LINUX KERNEL CAMP 2017

일시 : 2017년 8월 21일 (월) – 2017년 8월 23일 (수)

장소 : 선릉역 HJ 컨벤션센터

주최 : 한국정보과학회 컴퓨터시스템 소사이어티

Web : http://lkc2017.css.or.kr

한국정보과학회

KOREAN INSTITUTE OF INFORMATION SCIENTISTS AND ENGINEERS

행사개요

일 시 : 2017년 8월 21일 (월) – 2017년 8월 23일 (수)

장 소 : 선릉역 HJ컨벤션센터

주 최 : 한국정보과학회 컴퓨터시스템 소사이어티

모시는 글

한국정보과학회 컴퓨터시스템 소사이어티에서 제4회 리눅스 커널 캠프를 개최합니다. 리눅스 커널 캠프는 시스템 소프트웨어의 핵심이 되는 리눅스 커널에 대한 고급 수준의 단기 교육 프로그램으로 리눅스의 프로세스 관리 및 스케줄링, 메모리 관리, 파일시스템에 대한 체계적이고 심도 있는 강의를 진행할 예정입니다. 해당 분야에서 수십 년 경력을 자랑하는 국내 최고의 전문가들이 직접 강의를 진행하며 3일 동안 이론과 실습을 병행합니다. 이번 캠프가 교육기관, 연구소, 산업체에서 시스템 소프트웨어 관련 분야 종사자들에게 리눅스 커널에 대한 최신 정보를 획득할 좋은 기회가 되길 기원합니다.

2017년 8월

리눅스 커널캠프 프로그램 위원장 **남 범 석**

프로그램

1일차, 2017. 08. 21(월), 정진규 교수(성균관대), 서의성 교수(성균관대)

시간	프로그램
	정진규 교수(성균관대)
09:00 – 09:50	**메모리 관리 기본 개념** – 가상 메모리 및 페이징 – MMU 및 TLB
10:00 – 10:50	**물리 메모리 관리** – 물리 메모리 관리(zone, buddy system, etc) – 단편화 완화 및 개선 기법
11:00 – 11:50	**메모리 주소공간 관리** – 프로세스 및 커널 주소 공간 – 리눅스 페이지 테이블
12:00 – 13:00	**Lunch**
	서의성 교수(성균관대)
13:00 – 13:50	**CPU 스케줄링 개요** – 스케줄링 이론 – 다양한 스케줄링 알고리즘 소개
14:00 – 14:50	**리눅스 스케줄러 분석** – 리눅스 스케줄러 구조 – CFS 알고리즘 및 구현
15:00 – 15:50	**리눅스 로드밸런싱** – 로드밸런싱 개요 – 리눅스에서의 로드밸런싱 구현 방법
16:00 – 16:50	**리눅스 커널 동기화 I** – 리눅스 커널 내부에서의 동기화 개념 소개 – 각종 동기화 기법

2일차, 2017. 08. 22(화), 최종무 교수(단국대), 정진규 교수(성균관대)

시간	프로그램
	최종무 교수(단국대)
09:00 – 09:50	**파일 시스템 코드 수준 분석 필요성 (Case Study)** – IRON file system, SOSP, 2005 – F2FS(Flash Friendly File System), Fast, 2015
10:00 – 10:50	**파일시스템 개념적 구조** – Layered Architecture : System Call, VFS, Specific FS, Block Device Driver – FS Internals : Allocation, Metadata Management, Reliability
11:00 – 11:50	**파일시스템 추상화** – 단일한 사용자 인터페이스와 다양한 파일시스템 지원 – 가상 파일 시스템(Virtual File System) – From task_struct to inode
12:00 – 13:00	**Lunch**
	정진규 교수(성균관대)
13:00 – 13:50	**커널 메모리 관리** – 커널 동적 메모리 할당/해제 및 관리 – 페이지 캐시 관리
14:00 – 14:50	**프로세스 가상 메모리 관리** – 프로세스 가상메모리 생성 및 관리 – 요구 페이징 및 Copy-on-Write

15:00 – 15:50	**페이지 회수 기법**
	– LRU 및 working-set 기반 메모리 관리 – 동기적/비동기적 페이지 프레임 회수
16:00 – 16:50	**[실습] 페이지 컬러링**
	– 캐시 파티셔닝의 소프트웨어 기법인 페이지 컬러링 기법 구현 및 성능 측정

3일차, 2017. 08. 23(수), 서의성 교수(성균관대), 최종무 교수(단국대)

시간	프로그램
	서의성 교수(성균관대)
09:00 – 09:50	**리눅스 커널 동기화 II**
	– 리눅스 커널 내부에서의 동기화 개념 소개 – 각종 동기화 기법
10:00 – 10:50	**[실습1] 스케줄러 구현 I**
	– 리눅스에서의 커스텀 스케줄러 구현
11:00 – 11:50	**[실습2] 스케줄러 구현 II**
	– 리눅스에서의 커스텀 스케줄러 구현
12:00 – 13:00	**Lunch**
	최종무 교수(단국대)
13:00 –13:50	**파일시스템 소스 수준 구조**
	– 파일 접근과 file_operations, inode_operations 역할 – 파일시스템 접근과 file system type, super operations 역할
14:00 – 14:50	**[실습1] 파일 시스템**
	– FAT 소스 코드 분석 – 램디스크 상에서 FAT 구동
15:00 – 15:50	**[실습2] 파일 시스템**
	– F2FS 소스 코드 분석 – MDT 상에서 F2FS 구동실습
16:00 – 16:50	**[실습3] 파일 시스템**
	– 성능/신뢰성 향상 기술 (스토리지 특성인지 기술) – 파일 시스템 성능 비교 방법

CPU Scheduling, Linux Load Balancing, Kernel Sychronization

- CPU 스케쥴링 이론 및 알고리즘 소개
- 리눅스 스케쥴러 분석 및 CFS 알고리즘 구현
- 리눅스에서의 로드밸런싱 구현 방법 소개
- 리눅스 커널 동기화 개념 소개
- [실습] 스케쥴러 구현

서의성 교수

(성균관대학교)

Fundamental Concepts of CPU Scheduling

Euiseong Seo

College of Software
Sungkyunkwan University

* This material was originally written by Prof. Minsoo Ryu at Hanyang University for System Society of KIISE

Outline

1 CPU Scheduling — Page 3
2 Priority-Based Scheduling — Page 5
3 Proportional Share Scheduling — Page 15
4 Multiprocessor Scheduling — Page 24

CPU Scheduling

- **CPU scheduling is to decide when and which process to run among ready processes**

- **Scheduling criteria**
  - **Performance objectives**
    - Maximize CPU utilization or throughput
    - Minimize completion time, waiting time, or response time
  - **Real-time constraints**
    - Satisfy deadlines
  - **Fairness**
    - Provide CPU cycles proportional to weights

Classification of Scheduling Policies

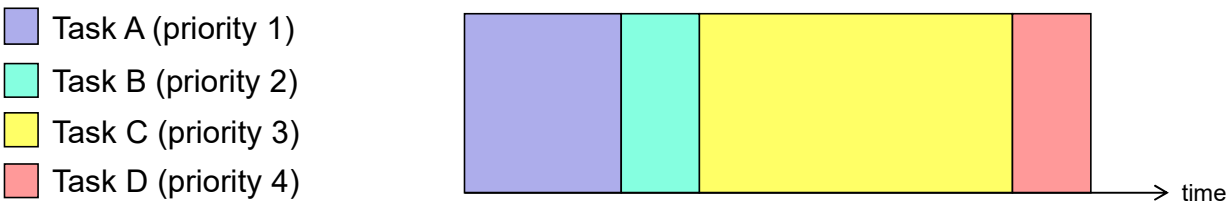
- **Two paradigms**
  - **Priority-based scheduling**
  - **Proportional share scheduling**

- **Number of processors**
  - **Uniprocessor scheduling**
  - **Multiprocessor scheduling**

Priority-Based Scheduling

Priority-Based Scheduling

- A priority number (integer) is associated with each process



- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive

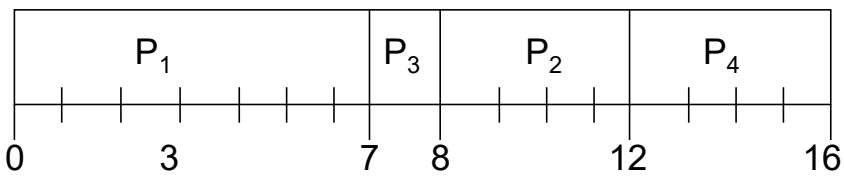
Priority-Based Scheduling Policies

- **Non-real-time policies**
  - FCFS (First-Come, First-Served)
  - SJF (Shortest-Job-First)
  - SRTF (Shortest-Remaining-Time-First)

- **Real-time policies**
  - RM (Rate Monotonic)
  - EDF (Earliest Deadline First)

SJF (Shortest-Job-First)

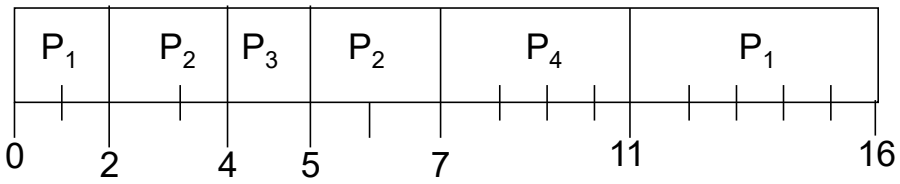
Process	Arrival Time	Execution Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



- **Average waiting time = (0 + 6 + 3 + 7)/4 = 4**

SRTF (Shortest-Remaining-Time-First)

Process	Arrival Time	Execution Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



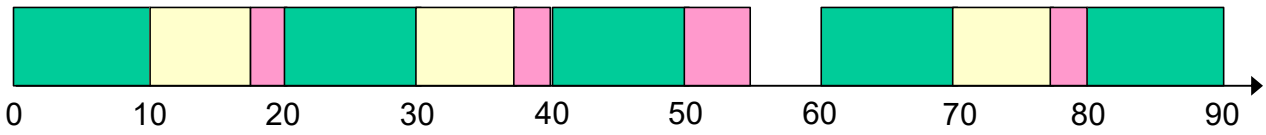
- **Average waiting time = (9 + 1 + 0 +2)/4 = 3**
  - **SRTF is optimal in that it achieves minimum average waiting time for a given set of processes**

RM (Rate Monotonic)

- **Assumptions**
  - **Processes have periods, worst-case execution times (WCETs), and deadlines**
- **Scheduling policy**
  - **Give higher priorities to tasks with shorter periods**
  - **Preemptive static priority scheduling**
- **Optimality**
  - **If a feasible static priority assignment exists for some process set, the RM priority assignment is feasible for that process set**

RM (Rate Monotonic)

- **Consider the following tasks**
  - **Process X ■: period = 20, WCET = 10, deadline = 20**
  - **Process Y □: period = 30, WCET = 8, deadline = 30**
  - **Process Z ■: period = 40, WCET = 4, deadline = 40**



- **Schedulability test**
  - **CPU utilization:** $U = \sum_{i=1}^{m} e_i / p_i$
  - **A set of m processes is schedulable if** $U \le m(2^{1/m} - 1)$
    - For large m, $m(2^{1/m} - 1) \approx \ln 2 \approx 0.69$

EDF (Earliest Deadline First)

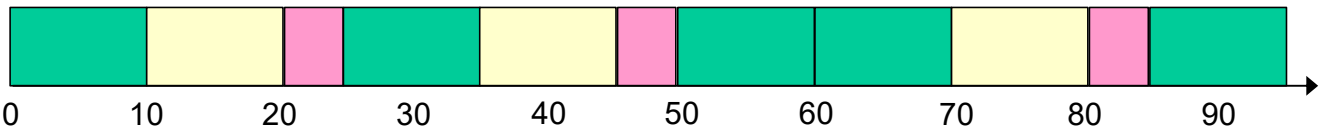
- **Scheduling policy**
  - **Give higher priorities to tasks with earlier absolute deadlines**
  - **Preemptive dynamic priority scheduling**

- **Optimality**
  - **If a feasible dynamic priority assignment exists for some process set, the EDF priority assignment is feasible for that process set**

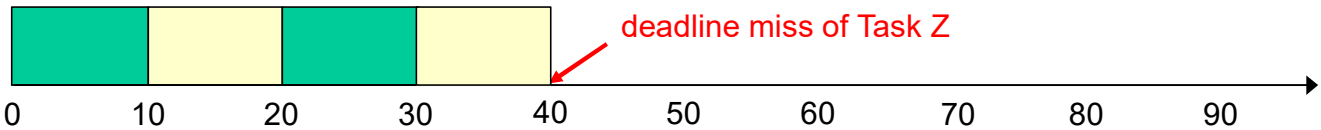
- **Schedulability test**
  - **A set of m processes is schedulable if and only if** $U \le 1$

EDF (Earliest Deadline First)

- Consider the following tasks
  - Process X ■: period = 20, WCET = 10, deadline = 20
  - Process Y □: period = 30, WCET = 10, deadline = 30
  - Process Z ■: period = 40, WCET = 5, deadline = 40



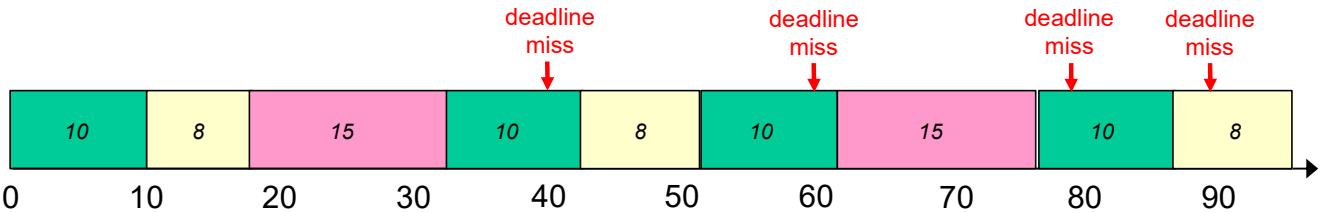
EDF schedule



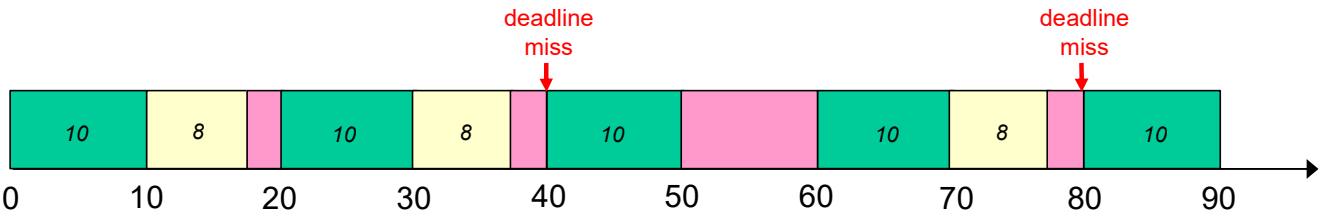
RM schedule

Non-schedulable Behavior

- Consider the following tasks
  - Process X ■: period = 20, WCET = 10, deadline = 20
  - Process Y □: period = 30, WCET = 8, deadline = 30
  - Process Z ■: period = 40, WCET = 15, deadline = 40



EDF schedule

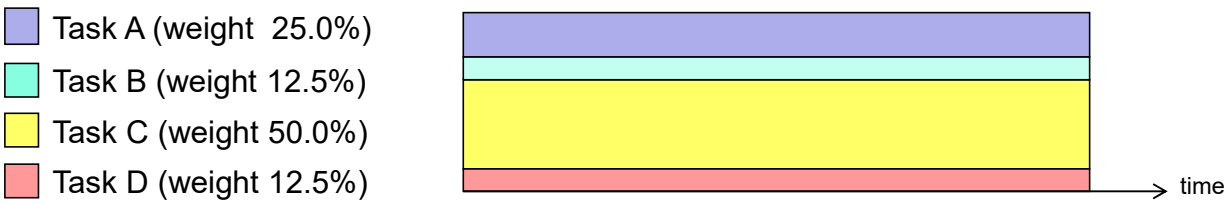


RM schedule

Proportional Share Scheduling

Proportional Share Scheduling

- **Basic concept**
  - **A weight value is associated with each process**
  - **The CPU is allocated to the process in proportion to its weight**



- **Two contexts**
  - **Fair queueing (in the context of communication networks)**
    - Packet scheduling
  - **Proportional share (in the context of operating systems)**
    - Process scheduling

Scheduling Algorithms

- **Network scheduling**
  - PGPS (= WFQ), Demers *et al.*, 1989
  - Virtual Clock, Lixia Zhang, 1990
  - SCFQ, Golestani, 1994
  - SFQ, Goyal *et al.*, 1996
  - WF^{2}Q, Bennett *et al.*, 1996

- **CPU scheduling**
  - Lottery and Stride, Waldspurger, 1995
  - Hierarchical SFQ, Goyal *et al.*, 1996
  - BVT, Duda *et al.*, 1999
  - VTRR, Nieh *et al.*, 2001

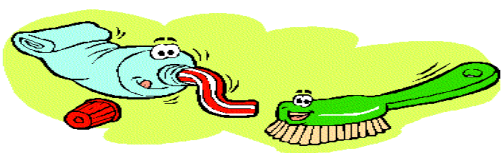
GPS (Generalized Processor Sharing)

- **A GPS server is defined by (Kleinrock, 1976)**

$$\frac{W_i(t_1,t_2)}{W_j(t_1,t_2)} \geq \frac{r_i}{r_j}, j = 1,2,\ldots,N$$

  - $W_i(t_1,t_2)$: the amount of session i traffic served in an $(t_1,t_2]$ interval
  - r_i : weight of session i

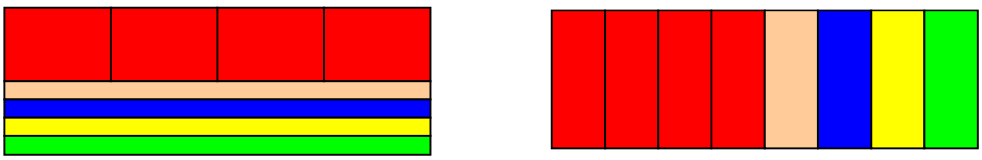
- **Packets of all sessions are served simultaneously**
  - Idealized fluid-flow system or bit-by-bit weighted round-robin



PGPS (Packet-by-Packet GPS)

- **PGPS is an approximation to GPS**
  - **Fair queueing on a packet-by-packet basis**
  - **Also known as WFQ (Weighted Fair Queueing)**

- **Algorithm**
  - F_p **: the time at which packet** p **will depart (finish service) under GPS**
  - **Serve packets in increasing order of** F_p



GPS PGPS

Fairness Bounds of PGPS

- **Bound on lag**
  - $\hat{F}_p - F_p \le \frac{L_{\max}}{r}$
    - $\hat{F}_p$: the time at which packet p departs under PGPS
    - F_p : the time at which packet p will depart under GPS
    - $L_{\max}$: the maximum packet length
    - If $r = 1Gbps$ and $L_{\max} = 1Kb$, then the lag is 1 ms

- **Bound on difference of services received**
  - $W_i(0,\tau) - \hat{W}_i(0,\tau) \le L_{\max}$

Worst-case Fair WFQ

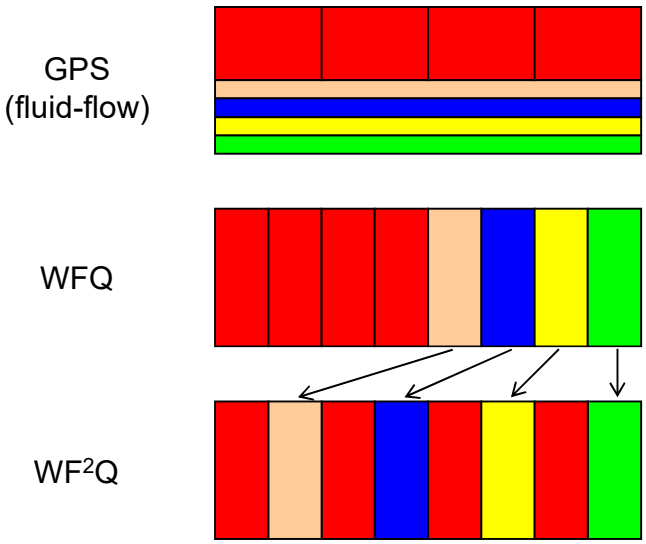
- **WF²Q only considers**
  - **Packets that have started receiving service under GPS**

WFQ satisfies only

$$W_p(t_1,t_2) - \hat{W}_p(t_1,t_2) \le L_{\max}$$

not

$$|W_p(t_1,t_2) - \hat{W}_p(t_1,t_2)| \le L_{\max}$$



Lottery and Stride Scheduling

- **Tickets: relative resource rights**
  - **Task τ_i has m_i tickets**
  - **A total of M tickets**
- **Lottery scheduling: probabilistic algorithm**
  - **Use random number generator to select a winning ticket**
  - **Task τ_i is probabilistically guaranteed a rate of $p = m_i / M$**
- **Stride scheduling: deterministic algorithm**
  - **"Stride" is inversely proportional to tickets**
  - **Task with minimum "pass" value is selected and its pass is advanced by its stride**

Stride Scheduling

- Task τ_1 : tickets = 3, stride = 2
- Task τ_2 : tickets = 2, stride = 3
- Task τ_3 : tickets = 1, stride = 6

Initial pass values are set to stride values

pass \ time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
τ_1's pass value	2	4	4	6	8	8	8	10	10	12	14	14	14	16	16
τ_2's pass value	3	3	6	6	6	9	9	9	12	12	12	15	15	15	18
τ_3's pass value	6	6	6	6	6	6	12	12	12	12	12	12	18	18	18

Smallest value is chosen
Pass is advanced by stride 3

Ties are broken arbitrarily

Multiprocessor Scheduling

Uniprocessor vs. Multiprocessor Scheduling

- **Uniprocessor scheduling**
  - **It is to decide when and which job will run**

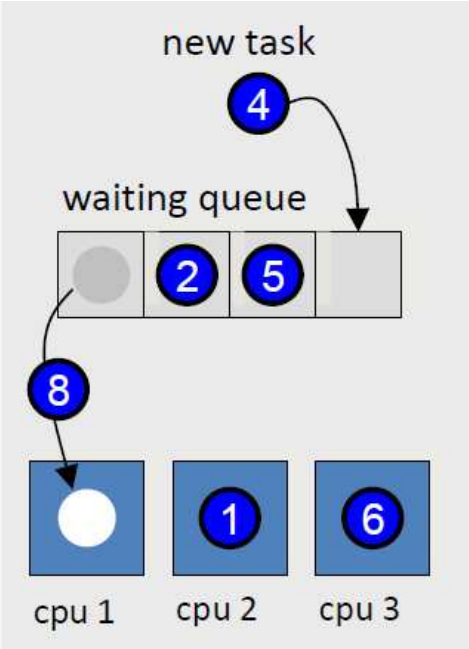
- **Multiprocessor scheduling**
  - **It is to decide not only when but also where a job will run**
  - **Almost the same goals as those of uniprocessor scheduling**
  - **But it raises new issues**
    - How to assign applications to multiple processors?
    - How to balance workload among processors?
    - How to define and exploit affinity?
    - How to manage processor heterogeneity?

Multiprocessor Scheduling Policies

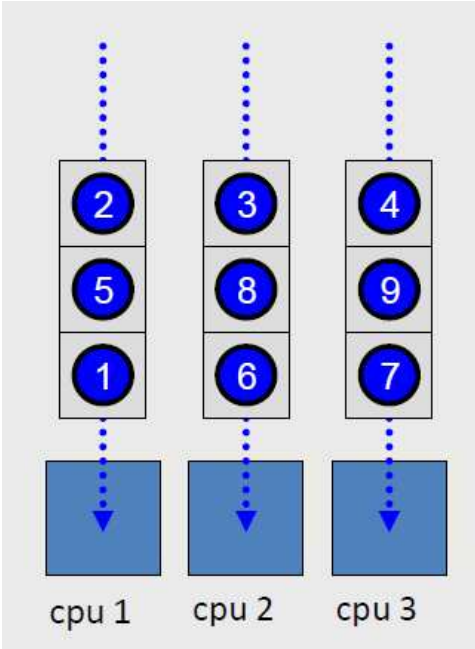
- **The same policies as uniprocessor policies**
  - **Priority-based scheduling: FCFS, SJF, SRTF, RM, EDF**
  - **Proportional share scheduling: PGPS, SFQ, WF^2Q, Lottery and Stride, BVT, VTRR**

- **Two approaches**
  - **Global scheduling**
    - The system has a single global process queue
    - Processes are dispatched to any available processors
  - **Partitioned scheduling**
    - Each processor has a separate process queue
    - Each queue is scheduled by an independent scheduler
    - Process migration may be allowed or not

Global vs. Partitioned Scheduling



Global Scheduling



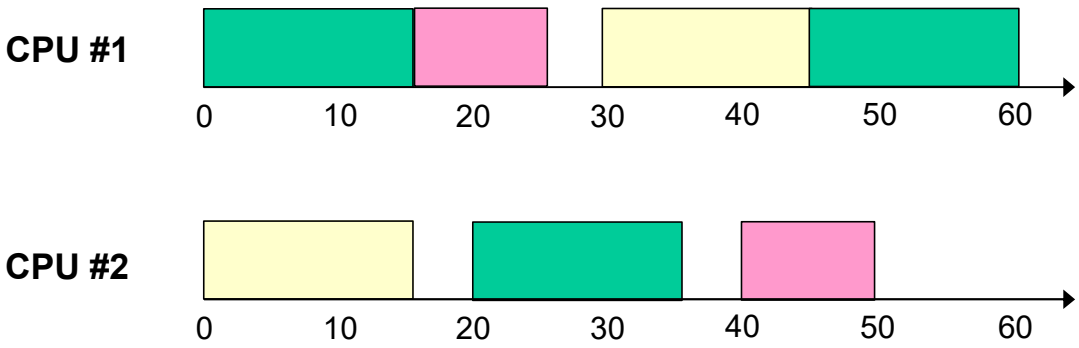
Partitioned Scheduling

Global vs. Partitioned Scheduling

- **Global scheduling**
  - **It is generally believed that global scheduling can achieve better performance**
  - **However, it can be inefficient due to the contention at the single queue and increased cache misses**
- **Partitioned scheduling**
  - **Performance can vary depending on the initial distribution of processes, i.e., a bin-packing problem**
  - **Different scheduling policies can be employed across processors**
  - **We can use the rich and extensive results from the uniprocessor scheduling theory**

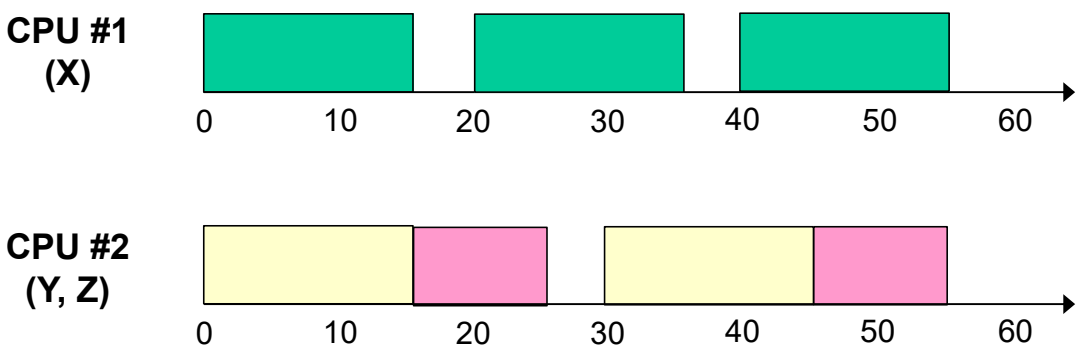
Global EDF

- Consider the following tasks
  - Process X ■: period = 20, WCET = 15, deadline = 20
  - Process Y □: period = 30, WCET = 15, deadline = 30
  - Process Z ■: period = 40, WCET = 10, deadline = 40



Partitioned EDF

- Consider the following tasks
  - Process X ■: period = 20, WCET = 15, deadline = 20
  - Process Y □: period = 30, WCET = 15, deadline = 30
  - Process Z ■: period = 40, WCET = 10, deadline = 40



Schedulability Analysis

➢ Global EDF

- **There is no single efficient test**
- **Most tests are very complex**

Theorem 2 (*GFB*). *A task set τ is schedulable with global* EDF *if*

$$\lambda_{tot} \leqslant m(1-\lambda_{max}) + \lambda_{max}. \quad (1)$$

Theorem 3 (*BAK*, from [3]). *A task set τ is schedulable with global* EDF *if, for all $\tau_k \in \tau$, there is a $\lambda \in \{\lambda_k\} \cup \{U_\ell | U_\ell \geqslant \lambda_k, \ell < k\}$ such that*

$$\sum_{\tau_i \in \tau} \min(1, \beta_{i,k}(\lambda)) \leqslant m(1-\lambda) + \lambda, \quad (2)$$

where

$$\beta_{i,k}(\lambda) = \begin{cases} U_i\left(1 + \frac{\max(0, T_i - D_i)}{D_k}\right) & \text{if} \quad U_i \leqslant \lambda \\ U_i\left(1 + \frac{T_i}{D_k}\right) - \lambda \frac{D_i}{D_k} & \text{if} \quad U_i > \lambda. \end{cases}$$

Theorem 6 (*FFDBF from* [10]). *A task set τ is schedulable with global* EDF *if $\exists \sigma | \lambda_{max} \leqslant \sigma < \frac{m - U_{tot}}{m-1} - \epsilon$ (with an arbitrarily small ϵ), such that $\forall t \geqslant 0$,*

$$ffdbf(t, \sigma) \leqslant (m - (m-1)\sigma)t \quad (10)$$

It can be proved that it is sufficient to check only those values of t in $\{kT_i + D_i | k \in \mathbb{N}\}_{i=1}^{n}$ that are smaller than[2]

$$\frac{\sum_{\tau_i \in \tau} C_i\left(1 - \frac{D_i}{T_i}\right)}{m - (m-1)\sigma - U_{tot}}. \quad (11)$$

…

…

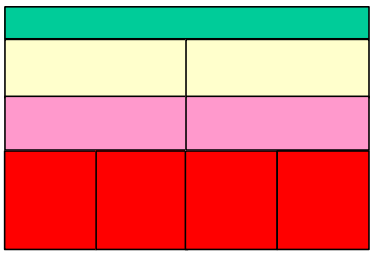
➢ Partitioned EDF

- **Sufficient to check if the CPU utilization does not exceed 100% for each processor**

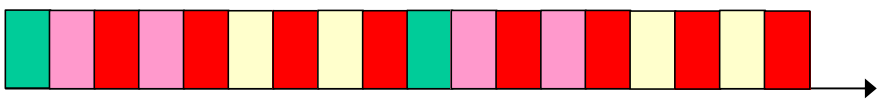
Global WFQ

➢ Consider the following tasks

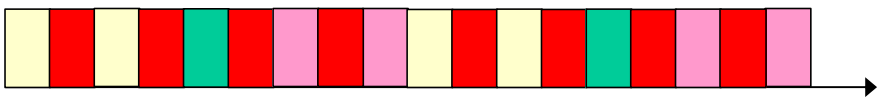
- **Process A ■: weight = 1**
- **Process B □: weight = 2**
- **Process C ■: weight = 2**
- **Process D ■: weight = 4**



CPU #1

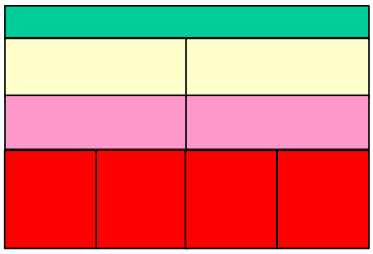


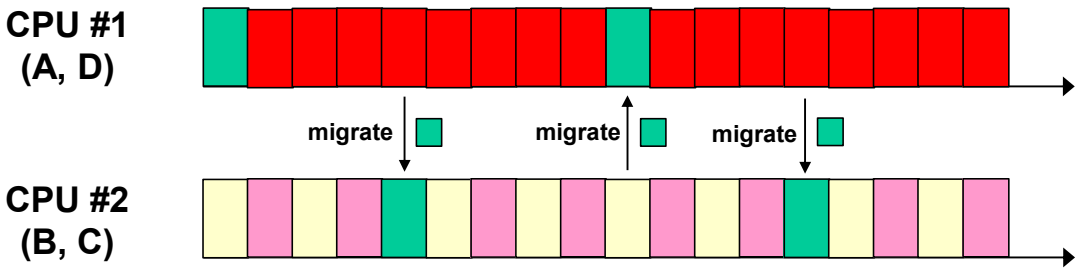
CPU #2



Partitioned WFQ with Load Balancing

- Consider the following tasks
  - Process A ■: weight = 1
  - Process B □: weight = 2
  - Process C ■: weight = 2
  - Process D ■: weight = 4





Linux Scheduler

Euiseong Seo

College of Software
Sungkyunkwan University

* This material was originally written by Prof. Minsoo Ryu at Hanyang University for System Society of KIISE

Outline

1 Introduction to Linux Scheduler — Page 3
2 Completely Fair Scheduler (CFS) — Page 14
3 CFS Implementation in the Kernel — Page 24
4 Deadline Scheduler — Page 38
4 Q & A — Page 45

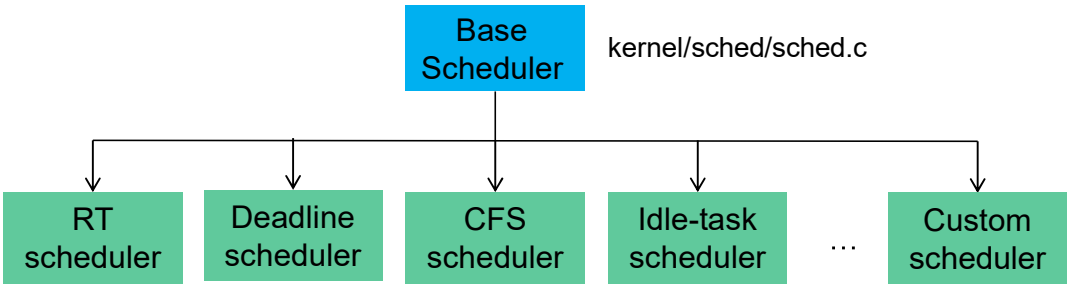
Introduction to Linux Scheduler

Evolution of Linux Scheduler

Version	CPU Scheduler
~ Linux 2.4	• Priority-based scheduling and Round-robin scheduling • Multilevel feedback queue • SMP support was introduced in Linux 2.2
Linux 2.4 ~ 2.6	• O(n) scheduler
Linux 2.6 ~ 2.6.22	• O(1) scheduler introduced by Ingo Molnar
Linux 2.6.23 ~	• CFS (Completely Fair Scheduler) by Ingo Molnar • O(log N) complexity

Scheduling Class

- **Since 2.6.23 kernel, the Linux scheduler has been made modular**
  - **The modularity is called scheduler classes**
  - **Scheduler classes enable different, pluggable algorithms to coexist, scheduling their own types of processes**
  - **Each scheduler class has a priority**
    - The base scheduler code iterates over each scheduler class in order of priority



Scheduling Class

- **Each task belongs to one specific scheduling class**

Class	Description	Policy
dl_sched_class	for real-time task with deadline	SCHED_DEADLINE
rt_sched_class	for real-time task	SCHED_FIFO SCHED_RR
fair_sched_class	for time-sharing task	SCHED_NORMAL SCHED_BATCH
idle_sched_class	for tasks that avoid to disturb other tasks	SCHED_IDLE

  - **Each scheduling class is linked with one another in a singly linked list, allowing the classes to be iterated**

```
#define sched_class_highest (&stop_sched_class)
#define for_each_class(class) \
   for (class = sched_class_highest; class; class = class->next)
```

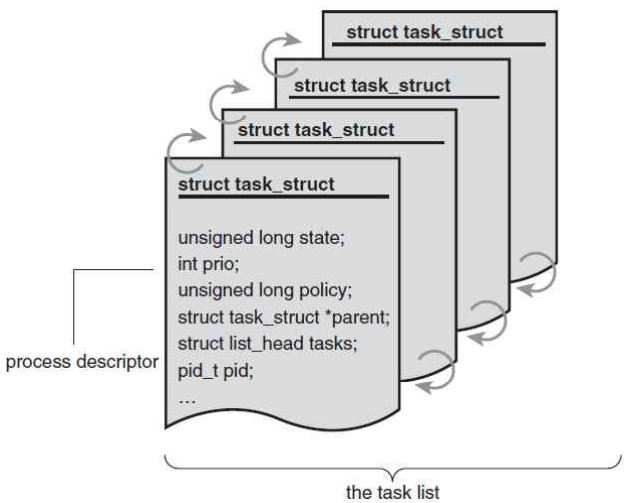
```
sched_class_highest
↳ stop_sched_class .next → dl_sched_class .next → rt_sched_class .next → fair_sched_class .next → idle_sched_class .next = NULL
```

Functions in Scheduling Class

Function	Operations
enqueue_task	• Put the task into the run queue • Increment the nr_running variable
dequeue_task	• Remove the task from the run queue • Decrement the nr_running variable
yield_task	• Dequeue the task and then enqueue it
check_preempt_curr	• Check whether the currently running task can be preempted by a new task
pick_next_task	• Choose the most appropriated task eligible to run next
set_curr_task	• Change task's scheduling class or group
load_balance	• Trigger load balancing code

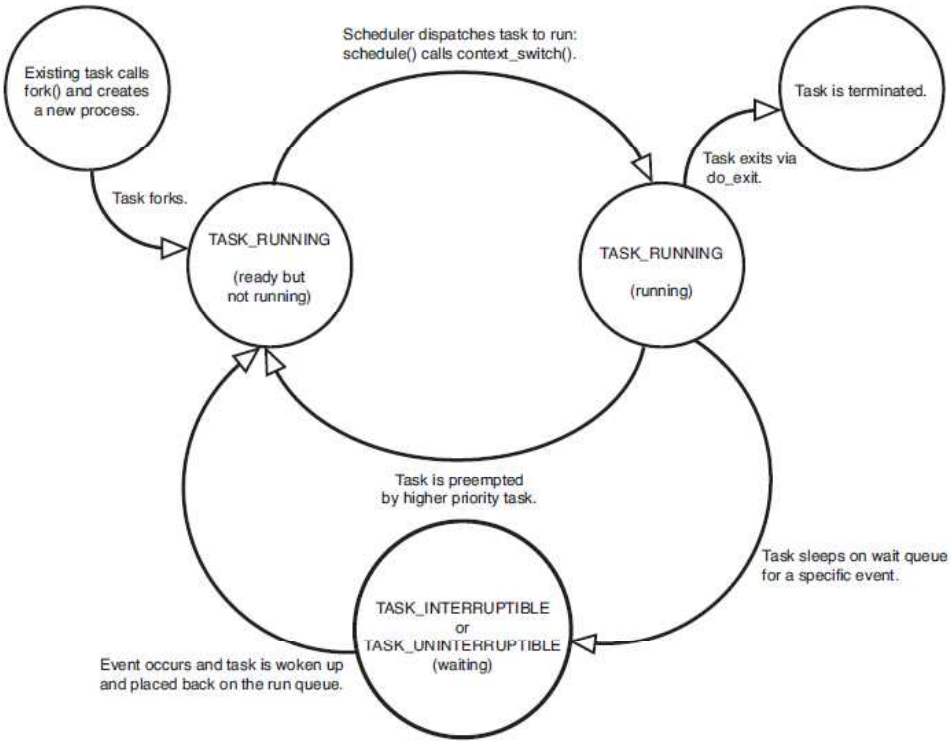
Tasks in Linux

- Linux internally refers to processes as tasks
- Process descriptor: task_struct



The kernel stores the list of tasks in a circular doubly linked list called the task list

Task State in Linux



Threads in Linux

- **There is no concept of a thread**
  - **Linux implements all threads as standard tasks**
- **The Linux kernel does not provide any special scheduling semantics or data structures to represent threads**
  - **A thread is merely a task that shares certain resources with other tasks**
  - **Each thread has a unique task_struct and appears to the kernel as a normal process—threads just happen to share resources, such as an address space, with other processes**

Task Structure

- **All tasks in Linux are represented by task_struct**
  - **Describes the task's state, its stack, process flags, priority, and much more**

include/linux/sched.h

```
struct task_struct {
    volatile long state;
    void *stack;
    atomic_t usage;
    unsigned int flags;
    unsigned int ptrace;
    ...
    int prio, static_prio, normal_prio;
    unsigned int rt_priority;
    const struct sched_class *sched_class;
    struct sched_entity se;
    struct sched_rt_entity rt;
    ...
}
```

kernel/sched/rt.c

```
const struct sched_class rt_sched_class = {
    .next           = &fair_sched_class,
    .enqueue_task   = enqueue_task_rt,
    .dequeue_task   = dequeue_task_rt,
    .yield_task     = yield_task_rt,
    .check_preempt_curr = check_preempt_curr_rt,
    ...
}
```

kernel/sched/fair.c

```
const struct sched_class fair_sched_class = {
    .next           = &idle_sched_class,
    .enqueue_task   = enqueue_task_fair,
    .dequeue_task   = dequeue_task_fair,
    .yield_task     = yield_task_fair,
    .yield_to_task  = yield_to_task_fair,
    .check_preempt_curr = check_preempt_wakeup,
    ...
}
```

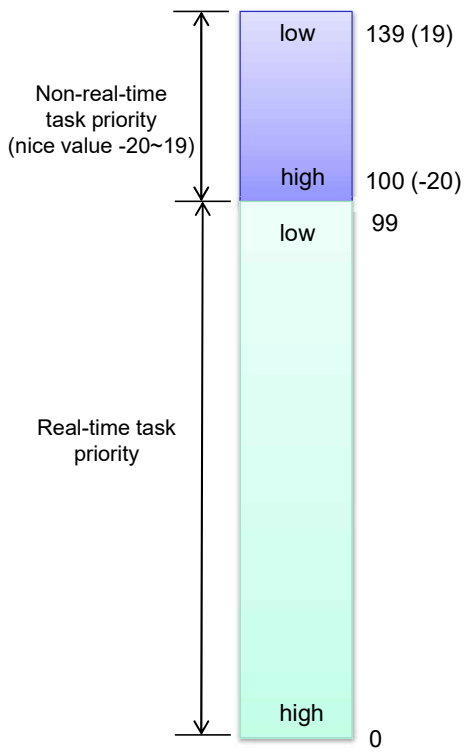
kernel/sched/idle_task.c

```
const struct sched_class idle_sched_class = {
    /* .next is NULL */
    /* no enqueue/yield_task for idle tasks */
    /* dequeue is not valid, we print a debug message there: */
    .dequeue_task       = dequeue_task_idle,
    .check_preempt_curr = check_preempt_curr_idle,
    ...
}
```


Task Priority

- **Non-real-time priority**
  - **Priority range: 100 ~ 139**
    - Nice value (-20 ~ 19, default of 0)
  - **Priority for normal time-sharing tasks**
    - Formerly, SCHED_OTHER
  - **A large nice value corresponds to a lower priority**

- **Real-time priority**
  - **Priority range: 0 ~ 99**
  - **Priority for real-time tasks**
    - SCHED_FIFO, SCHED_RR
  - **A smaller value corresponds to a higher priority**



Nice Values

- Nice values are the standard priority range used in all Unix systems
  - Different Unix systems apply them in different ways
  - In Mac OS X, the nice value is a control over the absolute time slice allotted to a process
- Linux uses nice values to control the proportion of time slice (weight)
  - Larger nice values correspond to a lower priority—you are being "nice" to the other processes on the system
  - Processes with a lower nice value (higher priority) receive a larger proportion of the system's processor compared to processes with a higher nice value (lower priority)

Completely Fair Scheduler

Completely Fair Scheduler (CFS)

- **CFS is based on a simple concept**
  - **The goal is to achieve an ideal multitasking system**
  - **Provide each task CPU time proportional to its weight**

$$C_{\tau_i}(t_1, t_2) = \frac{W(\tau_i)}{S_\Phi} \times (t_2 - t_1)$$

CPU time — Weight of task — Total weight — Total CPU time

- **Inspired by Con Kolivas's work**
  - **RSD (Rotating Staircase Deadline) scheduler**
  - **Released BFS (Brain Fuck Scheduler) in August 2009**
    - O(1) complexity

Task Weight

- **include/linux/sched.h**

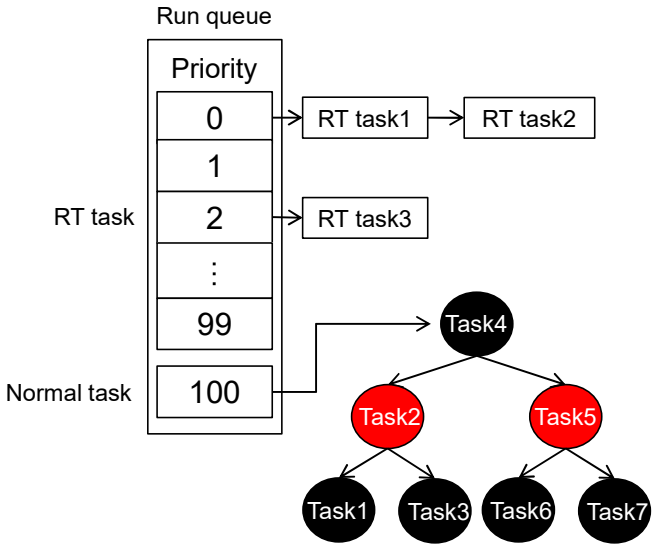
```
struct load_weight {
    unsigned long weight;
    u32 inv_weight;
};
```

- **kernel/sched/core.c**

```
const int sched_prio_to_weight[40] = {
/* -20 */    88761,    71755,    56483,    46273,    36291,
/* -15 */    29154,    23254,    18705,    14949,    11916,
/* -10 */     9548,     7620,     6100,     4904,     3906,
/*  -5 */     3121,     2501,     1991,     1586,     1277,
/*   0 */     1024,      820,      655,      526,      423,
/*   5 */      335,      272,      215,      172,      137,
/*  10 */      110,       87,       70,       56,       45,
/*  15 */       36,       29,       23,       18,       15,
};
```

Run Queue Structure

- **Each CPU owns its run queues**
  - **Partitioned scheduling with task migration**
  - **Red-black tree for the normal tasks, array for the RT tasks**




```
struct rt_rq {
    struct rt_prio_array active;
    unsigned int rt_nr_running;
    unsigned int rr_nr_running;
    ...
    int rt_throttled;
    u64 rt_time;
    u64 rt_runtime;
    ...
}

struct cfs_rq {
    struct load_weight load;
    unsigned int nr_running;
    u64 exec_clock;
    u64 min_vruntime;
    struct rb_root tasks_timeline;
    struct rb_node *rb_leftmost;
    ...
}
```


cfs_rq Structure

kernel/sched/sched.h

```
struct cfs_rq {
    struct load_weight load;
    unsigned int nr_running;
    u64 exec_clock;
    u64 min_vruntime;
    struct rb_root tasks_timeline;
    struct rb_node *rb_leftmost;

    struct sched_entity *curr, *next, *last, *skip;
#ifdef CONFIG_SMP
    struct sched_avg avg;
    u64 runnable_load_sum;
    unsigned long runnable_load_avg;
    atomic_long_t removed_load_avg, removed_util_avg;

#ifdef CONFIG_FAIR_GROUP_SCHED
    unsigned long h_load;
    u64 last_h_load_update;
    struct sched_entity *h_load_next;
#endif
#endif

#ifdef CONFIG_FAIR_GROUP_SCHED
    struct rq *rq;
    int on_list;
    struct list_head leaf_cfs_rq_list;
    struct task_group *tg;
#endif
    ...
}
```

- Load of the CFS run queue
- The number of tasks in the run queue
- Pointer to root node of red black tree
- Pointer to leftmost node of red black tree
- Per-CPU load information for CPU group scheduling
- Information for task group scheduling

CFS Algorithm: Virtual Runtime

- **Virtual Runtime**
  - **Approximate the "ideal multitasking" that CFS is modeling**
  - **To account for**
    - How long a process has run
    - How much longer it ought to run

$$VR(\tau, t) = \frac{W_0}{W(\tau)} \times PR(t)$$

- W_0: the weight of nice value 0
- $W(\tau)$: the weight of the task
- $PR(t)$: actual runtime received in time interval [0, t]

CFS Algorithm: Virtual Runtime

- **Meaning**
  - **Small virtual runtime**
    - The task has received less CPU time than what it should have received
  - **Large virtual runtime**
    - The task has received more CPU time than what it should have received

- **Scheduling decision**
  - **Choose the smallest virtual runtime (*vruntime*)**

CFS Algorithm: Timeslice

- **Timeslice is the time a task runs before it is preempted**
  - **It gives each runnable task a slice of the CPU's time**
  - **The length of timeslice of a taks is proportional to its weight**

constant

$$TS = \frac{weight}{runqueue's\ total\ weight} * P$$

Targeted preemption latency for CPU-bound tasks

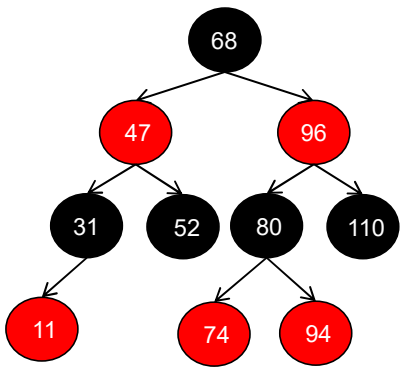
$$P = \begin{cases} \text{sched_latency}, & \text{if } n < \text{nr_latency} \\ \text{min_granularity} * n, & \text{otherwise} \end{cases}$$

Minimum preemption granularity for CPU-bound Tasks (default value is 1ms)

n = the number of tasks

CFS Algorithm: Red Black Tree

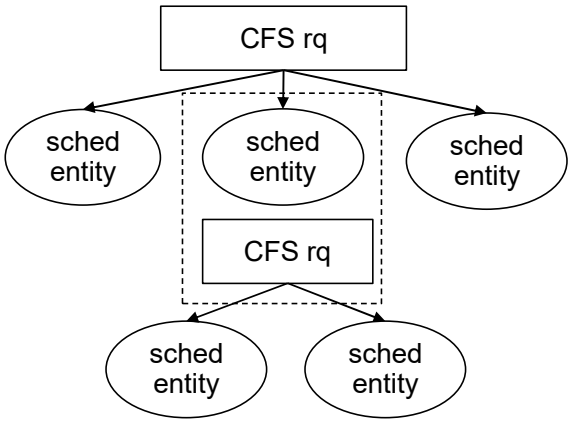
- **CFS maintains a time-ordered red black tree where all runnable tasks are sorted by virtual runtime**
  - **No path in the tree will ever be more than twice as long as any other (self-balancing)**
  - **Operations on the tree occur in O(log n) time**



Nodes are sorted by entity's virtual runtime

Group Scheduling and Processor Affinity

- **Group scheduling**
  - **A collection of tasks can share a virtual time**
  - **If a sched entity is a group, it has own run queue**
  - **Tasks in a group can be distributed on multiple CPUs**



- **Processor affinity**
  - **A task can run on all cpus by default**
  - **It can run only on a set of some cpus**
  - **Each task remembers its cpu running on**
    - (cpumask_t) task_struct->cpus_allowed

CFS Implementation in the Kernel

Time Management

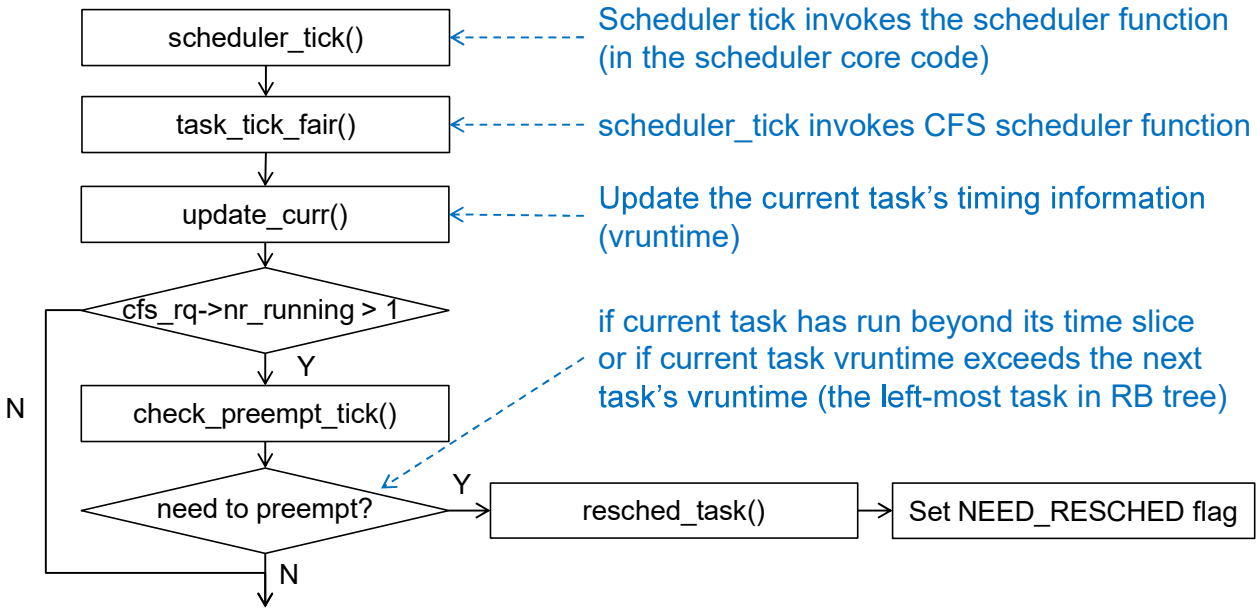
- **Timer interrupts are used to gauge the passing of time**
  - **The system timer goes off at a preprogrammed frequency, called the tick rate (HZ hertz defined in <asm/param.h>)**
  - **The time between any two successive timer interrupts is called a tick and is equal to 1/(tick rate) seconds**

- **With the initial version of Linux, the i386 has had a timer interrupt frequency of 100 Hz**
  - **During the 2.5 development series, however, the frequency was raised to 1000 Hz and was controversial**
  - **Although the frequency is again 100 Hz, it is now a configuration option**

Overview of Scheduling Flow

- **On each scheduling tick, CFS**
  - **Subtracts the currently running task's time slice by tick period**
    - When the time slice reaches 0, NEED_RESCHED flag is set
  - **Updates the virtual runtime of the currently running task**
  - **Checks NEED_RESCHED flag**
    - If set, schedules the task with the smallest virtual runtime in the run queue
    - Enqueue the currently running task into the RB tree
    - Dequeue the task at the left-most node in the RB tree

Setting NEED_RESCHED Flag



update_curr() in kernel/sched/fair.c

```
static void update_curr(struct cfs_rq *cfs_rq) {
    ...
    delta_exec = now - curr->exec_start;
    if (unlikely((s64)delta_exec <= 0))
        return;

    curr->exec_start = now;
    curr->sum_exec_runtime += delta_exec;
    curr->vruntime += calc_delta_fair(delta_exec, curr);
    ...
}

static inline u64 calc_delta_fair(u64 delta, struct sched_entity *se) {
    if (unlikely(se->load.weight != NICE_0_LOAD))
        delta = __calc_delta(delta, NICE_0_ LOAD, &se->load);

    return delta;
}
```

Update the current task's runtime statistics

Calculate the execution of the current task for one scheduling tick period

Calculate virtual runtime

= delta_exec * NICE_0_LOAD / weight

check_preempt_tick()

```
static void check_preempt_tick(struct cfs_rq *cfs_rq, struct sched_entity *curr) {
    ...
    ideal_runtime = sched_slice(cfs_rq, curr);
    delta_exec = curr->sum_exec_runtime - curr->prev_sum_exec_runtime;
    if (delta_exec > ideal_runtime) {
        resched_curr(rq_of(cfs_rq));
        clear_buddies(cfs_rq, curr);
        return;
    }
    ...
}
```

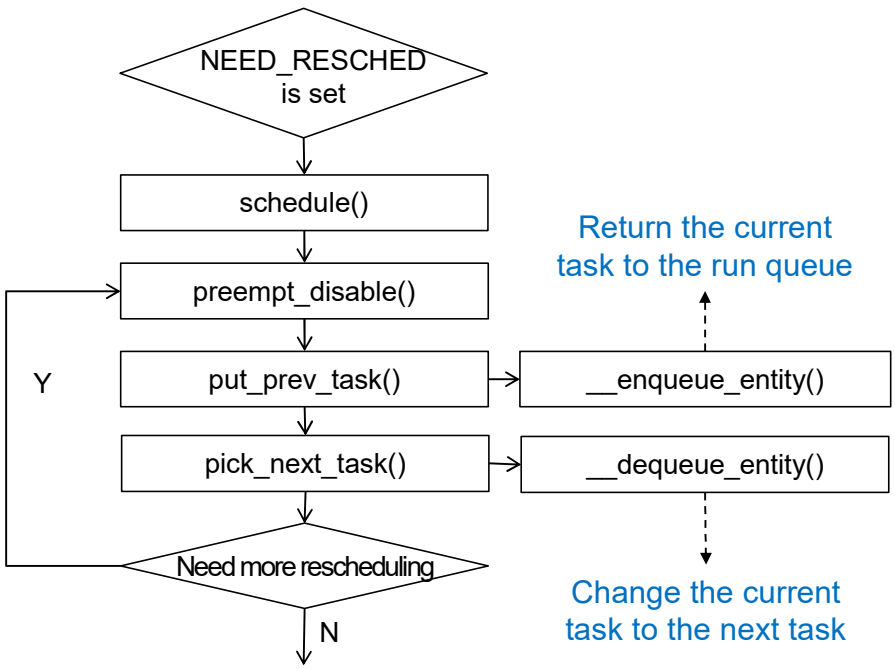
Set NEED_RESCHED flag

$$P = \begin{cases} \text{sched_latency}, & \text{if } n < \text{nr_latency} \\ \text{min_granularity} * n, & \text{otherwise} \end{cases}$$

```
static u64 sched_slice(struct cfs_rq *cfs_rq, struct sched_entity *se) {
    u64 slice = __sched_period(cfs_rq->nr_running + !se->on_rq);
    ...
        slice = __calc_delta(slice, se->load.weight, load);
    ...
    return slice;
}
```

$$TS = \frac{weight}{runqueue's\ weight} * P$$

Scheduling When NEED_RESCHED is Set



schedule()

<kernel/sched/core.c>

```
asmlinkage __visible void __sched schedule(void)
{
        struct task_struct *tsk = current;

        sched_submit_work(tsk);
        do {
                preempt_disable();
                __schedule(false);
                sched_preempt_enable_no_resched();
        } while (need_resched());
}
EXPORT_SYMBOL(schedule);
```

wrapper function for scheduling

main scheduling function is called after disabling preemption

<include/linux/sched.h>

```
static __always_inline bool need_resched(void)
{
        return unlikely(tif_need_resched());
}
```

<include/linux/thread_info.h>

```
#define tif_need_resched() test_thread_flag(TIF_NEED_RESCHED)
```


__schedule()

<kernel/sched/core.c>

```
static void __sched notrace __schedule(bool preempt)
{
        struct task_struct *prev, *next;
        unsigned long *switch_count;
        struct rq *rq;
        int cpu;

        cpu = smp_processor_id();
        rq = cpu_rq(cpu);
        prev = rq->curr;
        rcu_note_context_switch();
        …
        next = pick_next_task(rq, prev);
        clear_tsk_need_resched(prev);
        clear_preempt_need_resched();
        rq->clock_skip_update = 0;

        if (likely(prev != next)) {
                rq->nr_switches++;
                rq->curr = next;
                ++*switch_count;

                trace_sched_switch(preempt, prev, next);
                rq = context_switch(rq, prev, next);
        } else {
                lockdep_unpin_lock(&rq->lock);
                raw_spin_unlock_irq(&rq->lock);
        }

        balance_callback(rq);
}
```

Save current task as prev

Enqueue and dequeue task

pick_next_task_fair()

<kernel/sched/fair.c>

```
static struct task_struct *pick_next_task_fair(struct rq *rq, struct task_struct *prev)
{
        …
        put_prev_task(rq, prev);

        do {
                if (curr && curr->on_rq) {
                        update_curr(cfs_rq);
                } else {
                        curr = NULL;
                }
                se = pick_next_entity(cfs_rq, curr);
                cfs_rq = group_cfs_rq(se);
        } while (cfs_rq);

        p = task_of(se);

        if (hrtick_enabled(rq))
                hrtick_start_fair(rq, p);

        return p;
        …
}
```

Put previous task by enqueue

Pick and set next task for scheduling

Set high-resolution timer for setting time slice
(It allows for the task to check preemption point before 1/HZ seconds)

put_prev_entity()

- **CFS adds processes to the RB tree when a process becomes runnable (wake up) or is created via fork()**

```
static void put_prev_entity(struct cfs_rq *cfs_rq, struct sched_entity *prev)
{
        if (prev->on_rq)
                update_curr(cfs_rq);
        …
        if (prev->on_rq) {
                update_stats_wait_start(cfs_rq, prev);
                __enqueue_entity(cfs_rq, prev);
                update_entity_load_avg(prev, 0);
        }
        cfs_rq->curr = NULL;
}
```

Update the runtime and other statistics

Perform inserting the entry into the red black tree

__enqueue_entity()

```
static void __enqueue_entity(struct cfs_rq *cfs_rq, struct sched_entity *se) {
    struct rb_node **link = &cfs_rq->tasks_timeline.rb_node;
    struct rb_node *parent = NULL;
    struct sched_entity *entry;
    int leftmost = 1;
    while (*link) {
        parent = *link;
        entry = rb_entry(parent, struct sched_entity, run_node);
        if (entity_before(se, entry)) {
            link = &parent->rb_left;
        } else {
            link = &parent->rb_right;
            leftmost = 0;
        }
    }
    if (leftmost)
        cfs_rq->rb_leftmost = &se->run_node;
    rb_link_node(&se->run_node, parent, link);
    rb_insert_color(&se->run_node, &cfs_rq->tasks_timeline);
}
```

Find the right place to be located

Update rb_leftmost if the inserted task is located on the leftmost node

Insert the task

set_next_entity()

- **CFS removes processes from the RB tree when a process blocks (becomes unrunnable) or terminates**

```
static void set_next_entity(struct cfs_rq *cfs_rq, struct sched_entity *se)
{
        if (se->on_rq) {
                update_stats_wait_end(cfs_rq, se);
                __dequeue_entity(cfs_rq, se);
                update_load_avg(se, 1);
        }

        update_stats_curr_start(cfs_rq, se);
        cfs_rq->curr = se;
        …
        se->prev_sum_exec_runtime = se->sum_exec_runtime;
}
```

Perform removing the entry into the red black tree

__dequeue_entity()

```
static void __dequeue_entity(struct cfs_rq *cfs_rq, struct sched_entity *se) {
    if (cfs_rq->rb_leftmost == &se->run_node) {
        struct rb_node *next_node;

        next_node = rb_next(&se->run_node);
        cfs_rq->rb_leftmost = next_node;
    }

    rb_erase(&se->run_node, &cfs_rq->tasks_timeline);
}
```

Update the rb_leftmost if the task is the leftmost node

Remove the task from the red black tree

Deadline Scheduler

SCHED_DEADLINE

- **Scheduling class for real-time tasks with deadlines**
- **Earliest Deadline First (EDF) algorithm**
  - **Using three parameters**
    - Runtime, period, and deadline
  - **Augmented with Constant Bandwidth Server (CBS)**
    - Isolate behavior of tasks
- **Release**
  - **Added to version 3.14 of the Linux kernel mainline (March 30, 2014)**

Scheduling Parameters

- **A SCHED_DEADLINE task is guaranteed to receive `runtime` every `period`, and these `runtime` is available within `deadline` from the beginning of the `period`**
  - `Runtime`, `period`, and `deadline` are copied from `sched_attr` to `sched_dl_entity` during `sched_setattr()` system call

```
struct sched_dl_entity {
        struct rb_node          rb_node;    a node of red-black tree

        u64 dl_runtime;
        u64 dl_deadline;                    original scheduling parameters
        u64 dl_period;
        u64 dl_bw;                          dl_runtime / dl_deadline

        s64 runtime;                        remaining runtime
        u64 deadline;                       absolute deadline
        unsigned int flags;                 specifying the scheduler behavior
                                            actual scheduling parameters
                                            (continuously update during task execution)

        int dl_throttled, dl_new, dl_boosted, dl_yielded;    Some bool flags

        struct hrtimer dl_timer;            Per-task bandwidth enforcement timer
};
```

Budget Allocation

- **When a task wakes up,**
  - **If** $deadline < currentTime$ **or** $\frac{runtime}{deadline - currentTime} > \frac{dl_runtime}{dl_period}$**, then current budget is re-initialized**
    - $deadline = currentTime + dl_deadline$
    - $runtime = dl_runtime$
  - **Otherwise,** $deadline$ **and** $runtime$ **are left unchanged**

※ $deadline$ and $runtime$ are initialized as 0 at first

```
enqueue_task_dl()
→ enqueue_dl_entity()
      → update_dl_entity()
```

```
static inline void setup_new_dl_entity(struct sched_dl_entity *dl_se,
                                       struct sched_dl_entity *pi_se)
{
        struct dl_rq *dl_rq = dl_rq_of_se(dl_se);
        struct rq *rq = rq_of_dl_rq(dl_rq);

        WARN_ON(dl_time_before(rq_clock(rq), dl_se->deadline));

        if (dl_se->dl_throttled)
                return;

        dl_se->deadline = rq_clock(rq) + pi_se->dl_deadline;
        dl_se->runtime = pi_se->dl_runtime;
}
```


Earliest Deadline First

- **EDF algorithm selects the task with the smallest** $deadline$ **as the one to be executed first**

- **Linux sorts tasks by using red-black tree**
  - **__enqueue_dl_entity( )**
  - **Leftmost node has the smallest** $deadline$
  - **Scheduler just selects the leftmost node**
    - __dequeue_dl_entry( )

Update Runtime

- **When a task executes for an amount of time t, its $runtime$ is decreased as**
  - $runtime = runtime - t$

scheduler_tick()
→ task_tick_dl()
→ update_curr_dl()

```
static void update_curr_dl(struct rq *rq)
{
            struct task_struct *curr = rq->curr;
            struct sched_dl_entity *dl_se = &curr->dl;
            u64 delta_exec;

            if (!dl_task(curr) || !on_dl_rq(dl_se))
                        return;

            delta_exec = rq_clock_task(rq) - curr->se.exec_start;
            if (unlikely((s64)delta_exec <= 0)) {
                        if (unlikely(dl_se->dl_yielded))
                                    goto throttle;
                        return;
            }

            ...
            dl_se->runtime -= delta_exec;
throttle:
            ...
}
```


Throttling

- **When the $runtime$ becomes less than or equal to 0, the task cannot be scheduled until its $deadline$**
  - **It is called throttling**
  - **It sets timer for budget replenishment (to $deadline$)**
    - dl_task_timer( )
      → enqueue_task_dl( )
        → enqueue_dl_entity( )
          → replenish_dl_entity( )

When the replenish time is up (at $deadline$),
$deadline = deadline + dl_period$
$runtime = dl_runtime$

```
static void update_curr_dl(struct rq *rq)
{
            ...
            dl_se->runtime -= delta_exec;
throttle:
            if (dl_runtime_exceeded(dl_se) || dl_se->yielded) {
                        dl_se->dl_throttled = 1;
                        __dequeue_task_dl(rq, curr, 0);
                        if (unlikely(dl_se->dl_boosted || !start_dl_timer(curr)))
                                    enqueue_task_dl(rq, curr, ENQUEUE_REPLENISH);

                        if (!is_leftmost(curr, &rq->dl))
                                    resched_curr(rq);
            }
            ...
}
```

check $runtime \leq 0$

the task is throttled

set timer for replenishment

If the expiry time already passed

If leftmost task is changed by dequeue, then re-schedule to execute new task

Load Balancing

Euiseong Seo

College of Software
Sungkyunkwan University

* This material was originally written by Prof. Minsoo Ryu at Hanyang University for System Society of KIISE

Outline

1 Concepts of Load Balancing — Page 3

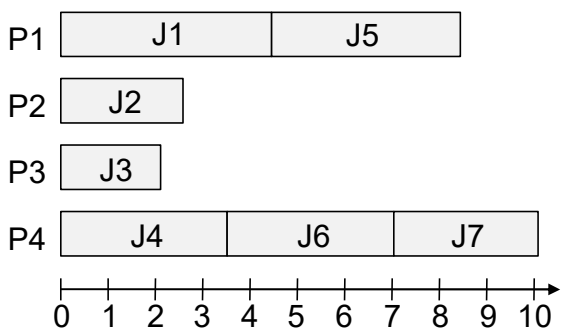
2 Load Balancing Algorithms — Page 8

3 Overhead of Load Balancing — Page 17

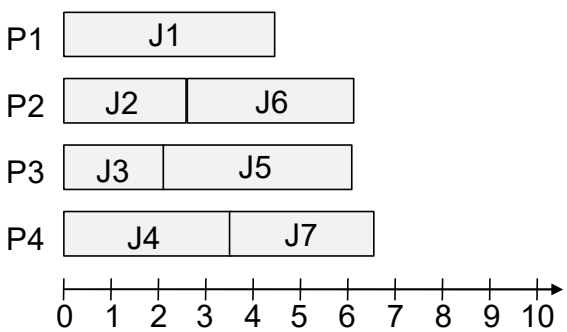
4 Load Balancing in Linux — Page 20

Goal of Load Balancing

- **Distribute workload evenly across processors**
  - **To get optimal resource utilization, maximize throughput, and minimize response time**



Unbalanced schedule



Well balanced schedule

Static Balancing vs. Dynamic Balancing

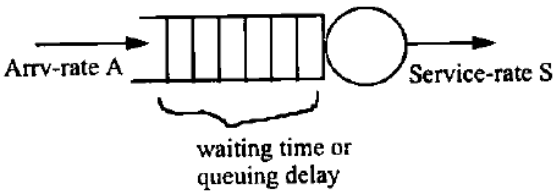
- Static balancing (partitioned sched. w/o migration)
  - Processes are statically assigned to processors during program compilation or loading
  - Works well when there is not much variation in the workload
- Dynamic balancing (partitioned sched. with migration)
  - Running processes are moved to remote processors
  - Works well when loads may vary significantly during runtime
  - But the cost of collecting and maintaining load information and process migration is high
- Adaptive balancing
  - Special type of dynamic load balancing
  - The algorithm may change depending on the system state
  - e.g.) If the system load is very high, it may not even attempt to collect load information

Load Balancing Policies

- Transfer policy
  - Decides whether a processor is eligible for load balancing
  - Usually based on a threshold such as queue length
- Selection policy
  - Decides which task should be moved
- Location policy
  - Decides where to send or receive the task
  - Typical approaches are polling, guessing, or random
- Information policy
  - Decide when (how often) trigger collection of system load information and where to collect them
  - On demand, periodic, or state change driven

Stability Condition

- **Without load balancing**
  - **If A < S, the system is stable**
  - **Otherwise, it is unstable**



- **With load balancing**
  - **If A+LB < S, the system is stable**
    - LB is the overhead due to load balancing
  - **Otherwise, it is unstable**
  - **If the algorithm leads to thrashing, the system is unstable**
    - When a task arrives at a processor 1, it gets transferred to processor 2
    - The task may get transferred to another processor 3
    - It may keep getting transferred from processor to processor

Diffusion Model and Rate of Convergence

- **We can use the following diffusion model to compute the rate of convergence to a uniform work distribution**

  - $w_i^{(t+1)} = w_i^{(t)} + \sum_j \alpha_{ij}(w_j^{(t)} - w_i^{(t)}) + \eta_i^{(t+1)} - c$

    - $w_i^{(t)}$: the number of tasks at processor i at time t
    - $\alpha_{ij}(w_j^{(t)} - w_i^{(t)})$: the amount of work that is exchanged
    - $\eta_i^{(t+1)}$: new work arrived at time t for processor i
    - c : a constant amount of work done between t and $t+1$

Sender-Initiated Algorithms

- Can be viewed as "work sharing"
- Transfer policy
  - Use a threshold policy on the ready queue length |Q|
  - If a task arrives and makes |Q| > T, the processor becomes a sender
- Selection policy
  - Consider only newly arrived tasks
- Location policy
  - Different algorithms (described on the next slide)
- Information policy
  - Demand driven

Location Policies

- Random LP
  - Choose a remote processor randomly
  - No overhead of collecting information
  - Still, better than no load balancing
- Threshold LP
  - Select remote processors randomly
  - But check if |Q| < T
- Shortest queue LP
  - Select *k* processors at random
  - Send to the processor with smallest |Q|
  - More overhead than threshold LP, but marginal improvement

Receiver-Initiated Algorithms

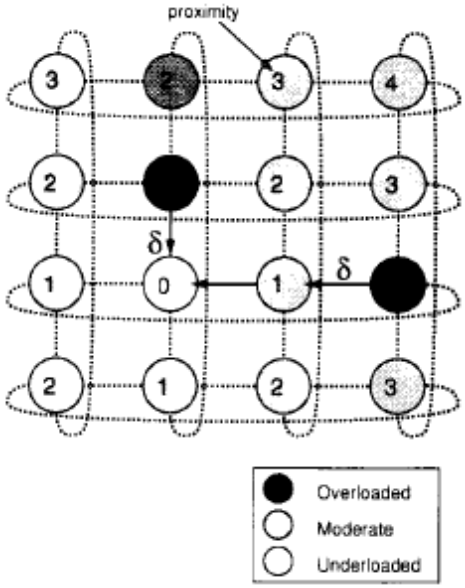
- Can be viewed as "work stealing"
- Transfer policy
  - Use a threshold policy on the ready queue length |Q|
  - If a task departs and makes |Q| < T, the processor becomes a receiver
- Selection policy
  - Consider only newly arrived tasks
- Location policy
  - Can choose randomly or based on the queue length |Q|
- Information policy
  - Demand driven

Symmetrically-Initiated Algorithms

- Combine the previous two algorithms
- Transfer policy
  - Use a double threshold <Lower, Upper>
  - If |Q| > Upper, the processor becomes a sender
  - If |Q| < Lower, the processor becomes a receiver
- Location policy
  - Sender-initiated: broadcasts a "too high" message, and waits for a reply from a receiver
  - Receiver-initiated: broadcasts a "too low" message, and waits for a reply from a sender
  - If no reply has been received within timeout, send a "system load is high/low" message and change the threshold

Gradient Model

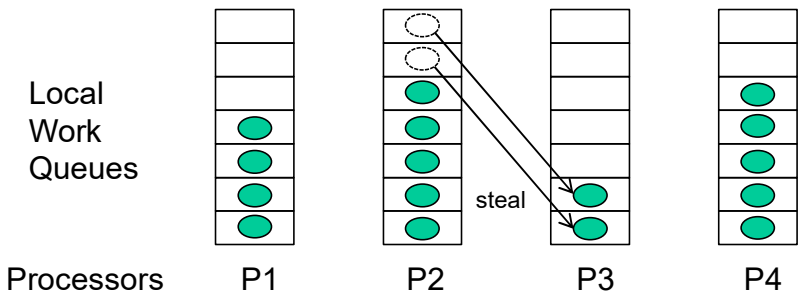
- A node's proximity is defined as the shortest distance from itself to the nearest lightly loaded node in the system



- Load balancing
  - The balancing process is initiated by lightly loaded processors reporting a proximity of 0
  - An overloaded processor sends a unit of its load to the neighbor of lowest proximity
  - Tasks are routed through the system in the direction of the nearest underloaded processors

Work Stealing

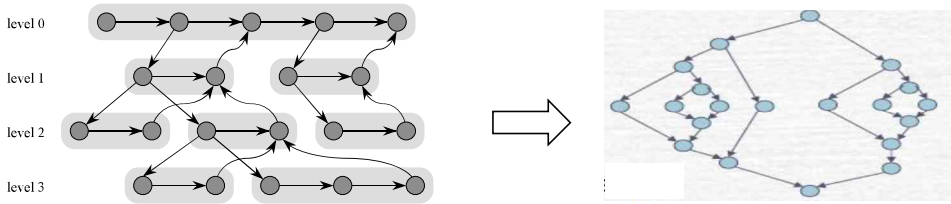
- Work stealing is known to be a simple but very effective approach to load balancing
  - Each processor maintains a local work queue
  - If a processor runs out of jobs, it steals from other processor



  - Work stealing is a receiver-initiated approach

Work Stealing

- **Target application: multithreaded computation that is fully strict**

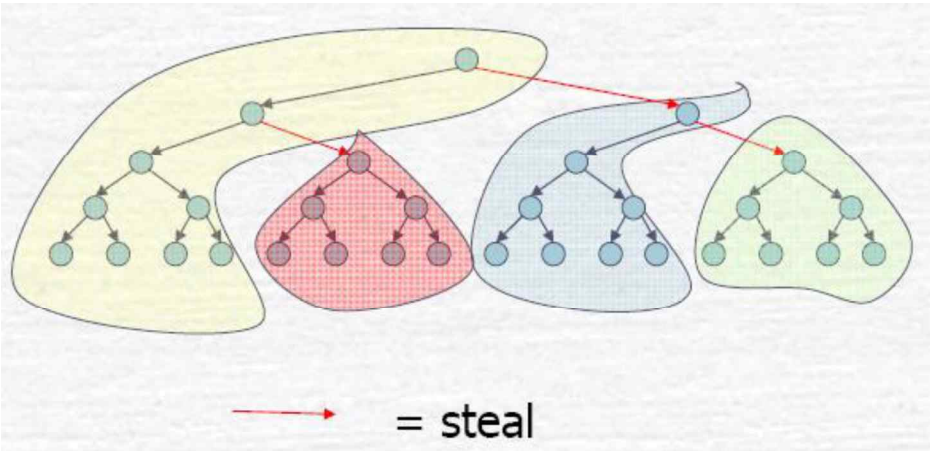


Multithreaded Computation Fully Strict (fork/join)

- **Basic mechanism**
  - **Each processor has a ready deque (double-ended queue)**
    - New thread is inserted and removed at the bottom
    - Thread to be migrated is removed from the top
  - **Processor steals from a randomly chosen processor if its deque is empty**

Work Stealing

- **Tends to schedule "sequential blocks" of tasks**



- **The proposed approach was used in Cilk**
  - **Cilk is a general-purpose programming language designed for multithreaded parallel computing (1994, MIT)**
  - **The commercial instantiation is Intel Cilk Plus (2010)**

Variations of Work Stealing

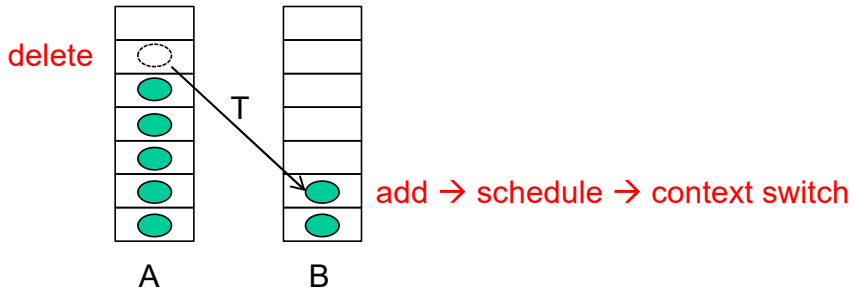
- **Threshold-based work stealing**
  - **When a processor becomes idle, it randomly chooses another processor and steals a job only if the victim's queue contains more than a threshold number of tasks**
- **Probabilistic work stealing**
  - **Whenever a processor accesses its local work queue, it performs a balancing operation with probability inversely proportional to the size of its work queue (with probability 1/m where m is the length of the work queue)**

Overhead of Load Balancing

- **Information collection**
  - **Can be done via either shared memory access or inter-processor communication**
  - **This causes minor overhead when compared to the task migration overhead**
- **Task migration**
  - **Scheduling**
  - **Cold cache effect**

Scheduling for Thread Migration

- **To move thread T from core A to core B**
  - **We need to access and update the kernel's scheduling data structures**
    - Deleting T from A's ready queue
    - Adding T to B's ready queue
  - **The scheduler on B needs to perform scheduling**
    - This would also involve context switching



Cold Cache Effect

- **Thread migration requires transferring program states**
  - **Register state, TLB state, and branch predictor state**

- **The largest overhead is caused by cache states**
  - **A primary mechanism is demand-fetching**
    - Executing the thread on the new core causes demand misses
    - This fills caches slowly (cold cache effect)

Load Balancing in Linux

Load Balancing in Linux

- Linux scheduler attempts to evenly distribute load across CPUs
  - Load of CPU (run queue): sum of task load
  - Load of task: $weight \times utilization(ready\&run)$

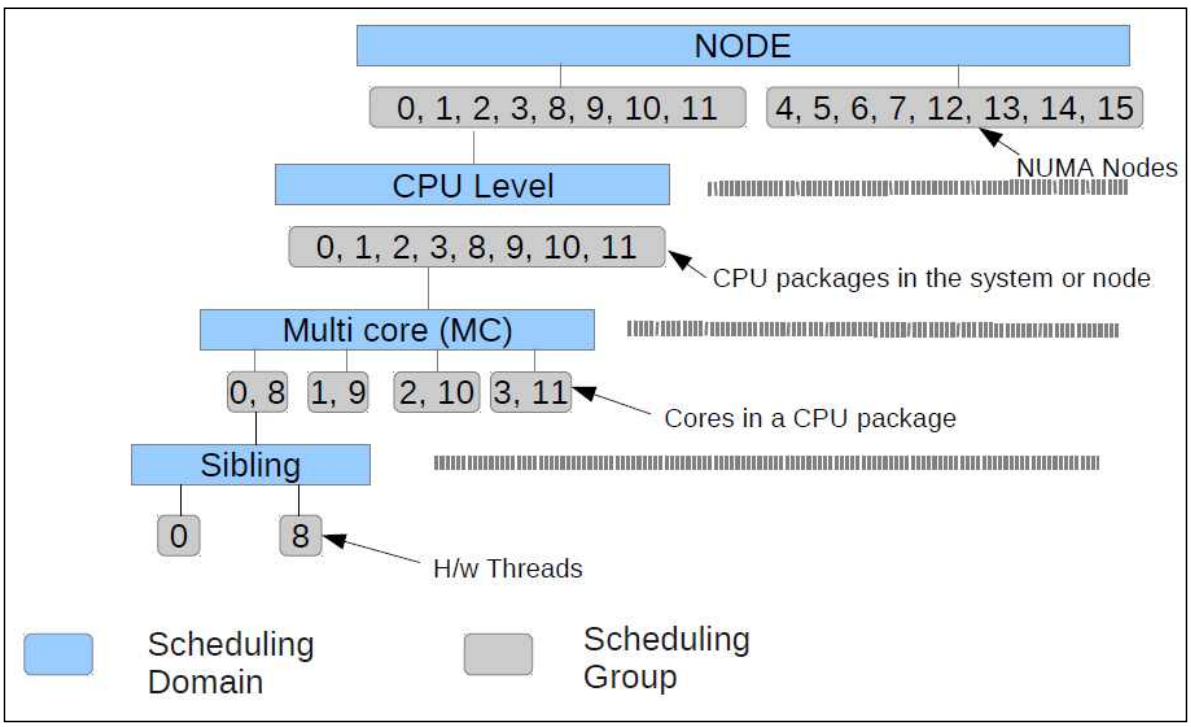
- Load balancing is triggered by
  - Timer interrupts
    - Load balancing code is invoked periodically
  - Scheduling events
    - Load balancing code is executed when
      - ✓ the CPU becomes idle
      - ✓ fork() has been executed
      - ✓ exec() has been executed
      - ✓ a task has waken up

Scheduling Domains and Groups

- **Scheduling domains**
  - **Load balancing takes place within a scheduling domain**
  - **Scheduling domains define the scheduling entities in a hierarchical fashion**
  - **Each scheduling domain spans a number of CPUs**
    - A domain's span MUST be a superset of it child's span
  - **Each scheduling domain must have one or more scheduling groups**

- **Scheduling groups**
  - **Each scheduling group contains one or more (virtual) CPUs**
  - **Load balancing takes place between scheduling groups**

Scheduling Domains and Groups



Linux Scheduling Domain

- **Default scheduling domain topology levels**
  - **SMT domain: for multi-threading in a package**
  - **MC domain: for multi-core in a package**
  - **DIE domain: for multi-package**

- **Domains have different configuration**
  - **Implies differences in flags**
    - SD_SHARE_PKG_RESOURCES
      - ✓ Sharing resources like the cache between groups
    - SD_SHARE_POWERDOMAIN
      - ✓ Reflect whether groups of CPUs in a sched_domain level can or not reach different power state

Timer-Driven Load Balancing

- **Load balancing is triggered by scheduling ticks**

kernel/sched/core.c

```
void scheduler_tick(void)
{
    int cpu = smp_processor_id();
    struct rq *rq = cpu_rq(cpu);
    struct task_struct *curr = rq->curr;

    ...
    curr->sched_class->task_tick(rq, curr, 0);
    ...
    rq->idle_balance = idle_cpu(cpu);
    trigger_load_balance(rq);
}
```

Invoked by a timer interrupt

Check if it is time to do load balancing

kernel/sched/fair.c

```
void trigger_load_balance(struct rq *rq)
{
    if (unlikely(on_null_domain(rq)))
        return;

    if (time_after_eq(jiffies, rq->next_balance))
        raise_softirq(SCHED_SOFTIRQ);
    ...
}

__init void init_sched_fair_class(void)
{
    open_softirq(SCHED_SOFTIRQ, run_rebalance_domains);
    ...
}
```

If it is time for load balancing, mark it for the softirq handler

Perform load balancing

run_rebalance_domains()

<kernel/sched/fair.c>

```
static void run_rebalance_domains(struct softirq_action *h)
{
        struct rq *this_rq = this_rq();
        enum cpu_idle_type idle = this_rq->idle_balance ? CPU_IDLE : CPU_NOT_IDLE;

        /*
         * If this cpu has a pending nohz_balance_kick, then do the
         * balancing on behalf of the other idle cpus whose ticks are
         * stopped. Do nohz_idle_balance *before* rebalance_domains to
         * give the idle cpus a chance to load balance. Else we may
         * load balance only within the local sched_domain hierarchy
         * and abort nohz_idle_balance altogether if we pull some load.
         */
        nohz_idle_balance(this_rq, idle);
        rebalance_domains(this_rq, idle);
}
```

Do balancing on behalf of the other idle CPUs whose ticks are stopped

Check each scheduling domain to see if it is due to be balanced and initiate a balancing operation if so.

rebalance_domains()

<kernel/sched/fair.c>

```
static void rebalance_domains(struct rq *rq, enum cpu_idle_type idle)
{
        int continue_balancing = 1;
        int cpu = rq->cpu;
        unsigned long next_balance = jiffies + 60*HZ;
        int update_next_balance = 0;
        …
        for_each_domain(cpu, sd) {
                …
                if (!(sd->flags & SD_LOAD_BALANCE))
                        continue;
                if (!continue_balancing) {
                        if (need_decay)
                                continue;
                        break;
                }
                …
                interval = get_sd_balance_interval(sd, idle != CPU_IDLE);
                …
                if (time_after_eq(jiffies, sd->last_balance + interval)) {
                        if (load_balance(cpu, rq, sd, idle, &continue_balancing)) {
                                idle = idle_cpu(cpu) ? CPU_IDLE : CPU_NOT_IDLE;
                        }
                        sd->last_balance = jiffies;
                        interval = get_sd_balance_interval(sd, idle != CPU_IDLE);
                }
                if (time_after(next_balance, sd->last_balance + interval)) {
                        next_balance = sd->last_balance + interval;
                        update_next_balance = 1;
                }
        }
        …
        if (likely(update_next_balance)) {
                rq->next_balance = next_balance;
                …
        }
}
```

1 m, the maximum value, $next_balance \leq 1\ m$

Starting from the current domain
All parent domains are visited before termination condition is satisfied

Variable interval, $interval \leq 0.1\ s$

Condition value for termination of the loop

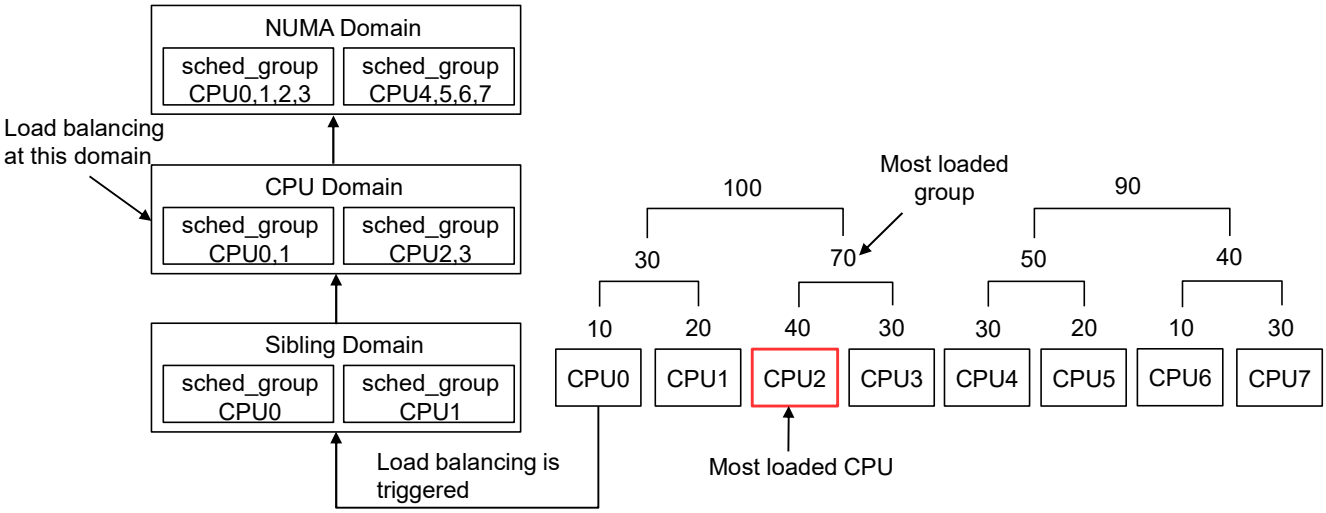
Load balancing

Earliest time will be selected for the next balance time

Applying earliest time

load_balance()

- When the sched domain should be rebalanced
  - Find the most loaded CPU of the most loaded group in the current domain
  - Move tasks from the CPU to the current CPU if some conditions are satisfied



calculate_imbalance()

```
static inline void calculate_imbalance(struct lb_env *env, struct sd_lb_stats *sds)
{
        unsigned long max_pull, load_above_capacity = ~0UL;
        struct sg_lb_stats *local, *busiest;

        local = &sds->local_stat;
        busiest = &sds->busiest_stat;
        ...
        if (busiest->avg_load <= sds->avg_load ||
            local->avg_load >= sds->avg_load) {
                env->imbalance = 0;
                return fix_small_imbalance(env, sds);
        }
        if (busiest->group_type == group_overloaded &&
                local->group_type   == group_overloaded) {
                load_above_capacity = busiest->sum_nr_running *
                            SCHED_LOAD_SCALE;
                if (load_above_capacity > busiest->group_capacity)
                            load_above_capacity -= busiest->group_capacity;
                else
                            load_above_capacity = ~0UL;
        }
        max_pull = min(busiest->avg_load - sds->avg_load, load_above_capacity);

        env->imbalance = min(
                max_pull * busiest->group_capacity,
                (sds->avg_load - local->avg_load) * local->group_capacity
        ) / SCHED_CAPACITY_SCALE;
        ...
}
```

If average load across all groups in current domain is greater than busiest group's or less than current group's, then balancer fixes small imbalance

Prevent making idle CPU although the current group can reduce the imbalance

Set imbalance nearly the average load of current domain

Conditions for Load Balancing

- This group is the most loaded group
- This group is more loaded than the average load of this domain
- The imbalance exceeds the specified limit

```
if (local->avg_load >= busiest->avg_load)
            goto out_balanced;

if (local->avg_load >= sds.avg_load)
            goto out_balanced;

if (env->idle == CPU_IDLE) {
            if ((busiest->group_type != group_overloaded) &&
                (local->idle_cpus <= (busiest->idle_cpus + 1)))
                        goto out_balanced;
} else {
            if (100 * busiest->avg_load <=
                env->sd->imbalance_pct * local->avg_load)
                        goto out_balanced;
}
```

Domain imbalance factor is determined by domain type:
If the domain shares CPU power, 110%
If the domain shares CPU package resources, 117%
If the domain is in NUMA, 112%

Choosing the Tasks

- *load_balance()* looks for tasks
  - Inactive (likely to be cache cold)
- *load_balance()* skips tasks that are
  - Likely to be cache warm
  - Currently running on a CPU
  - Not allowed to run on the current CPU (as indicated by the *cpus_allowed* bitmask in the *task_struct*)

Event-Driven Load Balancing

- **Event-driven load balancing is performed by setting flags**

include/linux/sched.h

```
#define SD_LOAD_BALANCE      0x0001  /* Do load balancing on this domain. */
#define SD_BALANCE_NEWIDLE   0x0002  /* Balance when about to become idle */
#define SD_BALANCE_EXEC      0x0004  /* Balance on exec */
#define SD_BALANCE_FORK      0x0008  /* Balance on fork, clone */
#define SD_BALANCE_WAKE      0x0010  /* Balance on wakeup */
#define SD_WAKE_AFFINE       0x0020  /* Wake task to waking CPU */
```

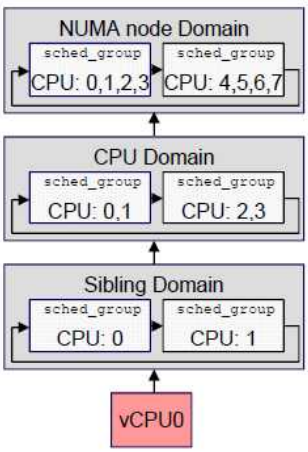
- **Example code: initialize common sched_domain**
  - **Load balancing will be triggered when**
    - The CPU becomes newly idle
    - New task is created

```
.flags  = 1*SD_LOAD_BALANCE
        | 1*SD_BALANCE_NEWIDLE
        | 1*SD_BALANCE_EXEC
        | 1*SD_BALANCE_FORK
        | 0*SD_BALANCE_WAKE,
```


Event-Driven Load Balancing

- **When one of the events (exec, fork, wakeup, exit) occurs, the CFS searches up the domain hierarchy**
  - **Perform load balancing at the highest possible sched domain with the corresponding flag set**

Policy flag	Sibling *sched domains*	CPU *sched domains*	NUMA node *sched domains*
SD_BALANCE_NEWIDLE	1	1	0
SD_BALANCE_EXEC	1	1	1
SD_BALANCE_FORK	1	1	1
SD_WAKE_IDLE	1	0	0
SD_WAKE_AFFINE	1	1	0
SD_WAKE_BALANCE	1	1	1
SD_SHARE_CPUPOWER	1	0	0



Event-Driven Load Balancing

- When a task is newly created or woke up via fork(), exec(), wakeup()
  - Select the least loaded group in its current domain
  - Move the task to the least loaded CPU
- When the CPU becomes newly idle
  - Select the most loaded group in its current domain
  - Move tasks from the most loaded CPU to this CPU

Linux Kernel Synchronization I

Euiseong Seo

College of Software
Sungkyunkwan University

* This material was originally written by Prof. Minsoo Ryu at Hanyang University for System Society of KIISE

Outline

1 Race Condition Scenarios in the Linux Kernel — Page 3

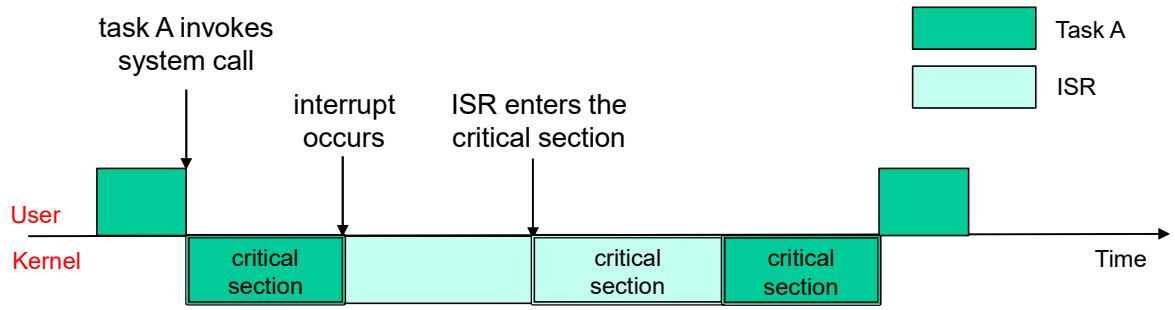
2 Synchronization for Single Core Hardware — Page 6

3 Synchronization for Multicore Hardware — Page 8

4 Q & A — Page 32

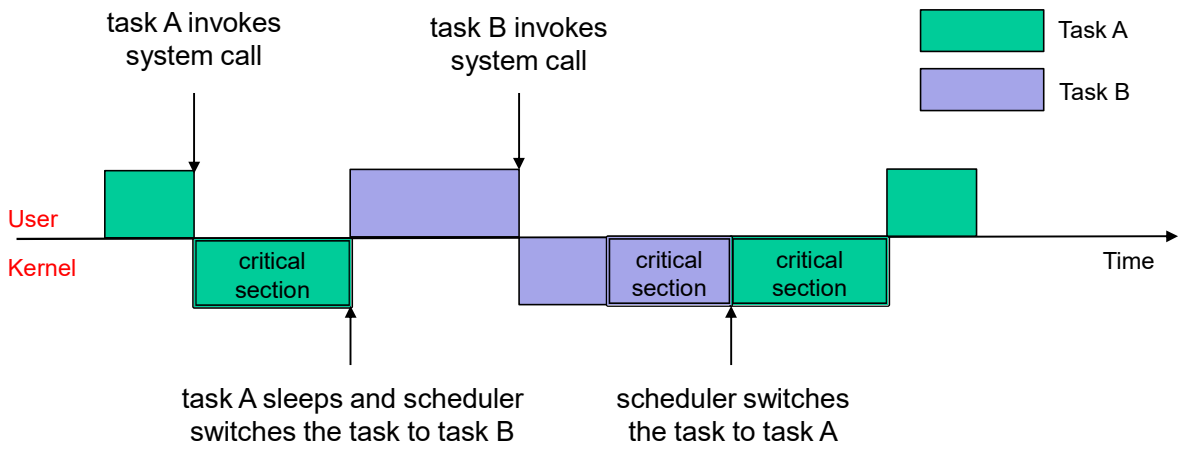
Race Condition Scenario #1 in the Kernel

➢ System call and interrupt



Race Condition Scenario #2 in the Kernel

➢ System call and preemption

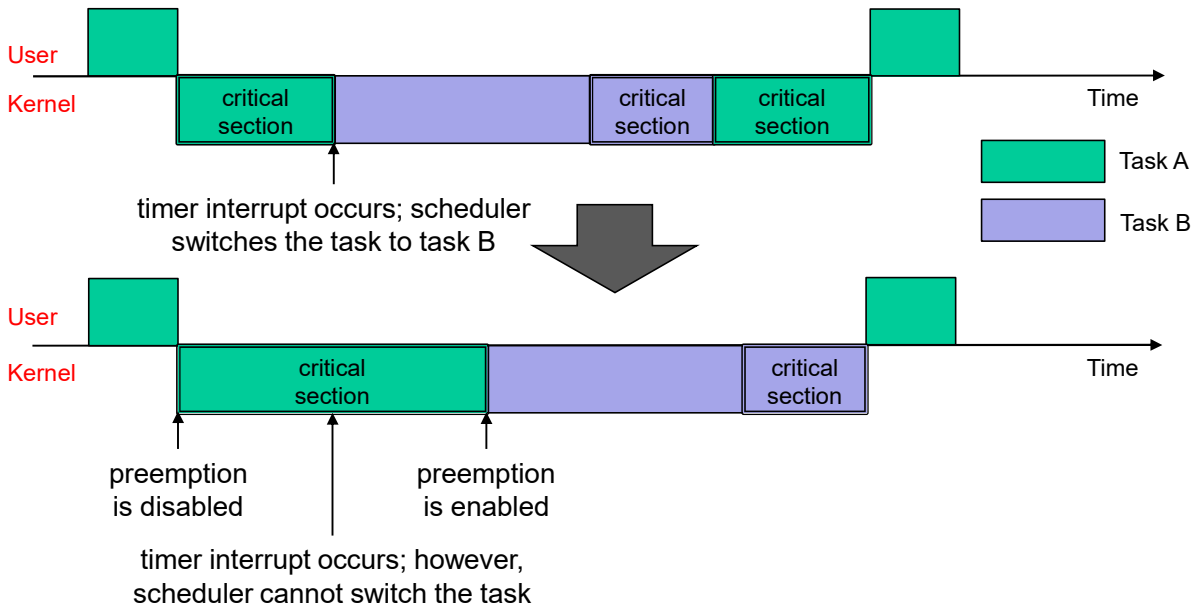


Synchronization Approaches in the Kernel

- For single core hardware
  - Disable preemption or interrupts to prevent other tasks or ISRs from running
- For multicore hardware
  - Atomic operations to perform multiple actions at once
  - Locking primitives to prevent other tasks from entering a critical section

Disabling Preemption

- Allow a task to complete its critical section without being interfered by other tasks



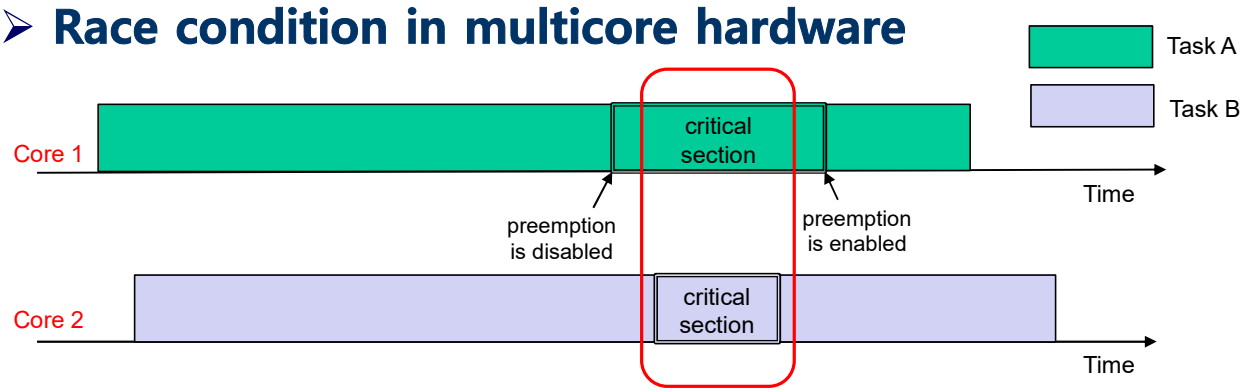
Disabling Preemption in the Kernel

- **Three functions**
  - **preempt_disable()**
    - Disable kernel preemption by incrementing the preemption counter
  - **preempt_enable()**
    - Decrement the preemption counter and checks and services any pending reschedules if the count is now zero
  - **preempt_count()**
    - Return the preemption count

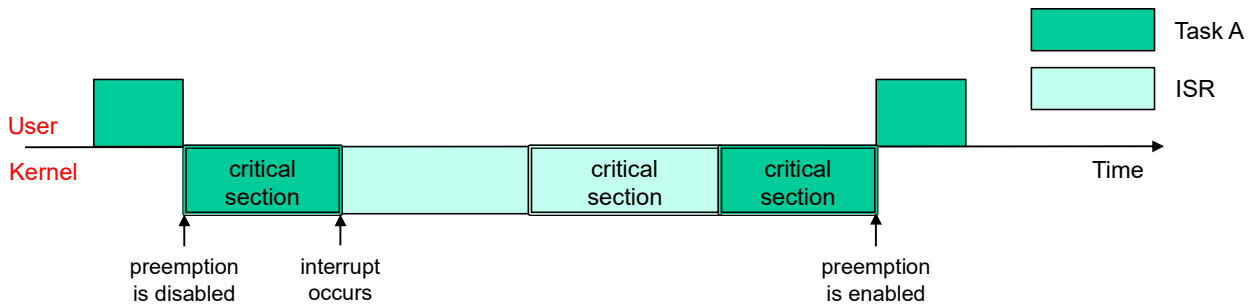
- **Preemption counter**
  - **preempt_count = 0 → preemptable**
  - **preempt_count > 0 → not preemptable**

Limitations of Disabling Preemption

- **Race condition in multicore hardware**

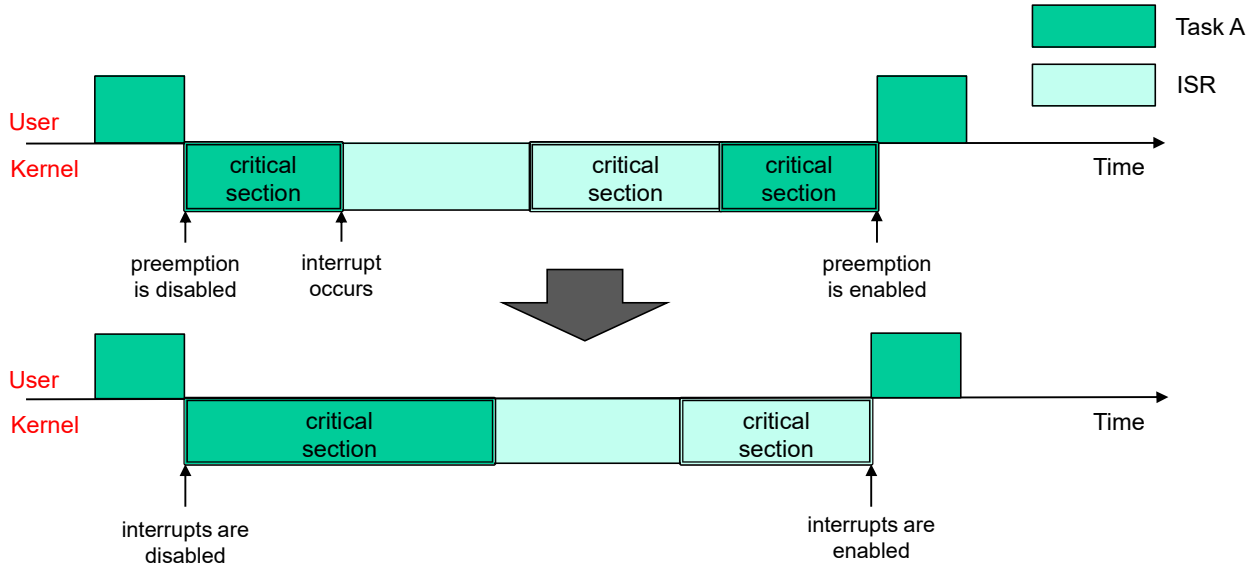


- **Race condition between a task and an ISR**



Disabling Interrupts

- Allow a task to complete its critical section without being interfered by interrupts
  - Disabling interrupts also disables kernel preemption



Disabling Interrupts in the Kernel

- Simply disable and enable interrupts for the current processor
  - Clear and set interrupt flags of the processor

```
local_irq_disable();
/* interrupts are disabled .. */
local_irq_enable();
```

- Disable and enable interrupts with saving and restoring the state of the interrupt system

```
local_irq_save(flags);
/* ... */
local_irq_restore(flags);
```

local_irq_disable( )

- The local_irq_disable( ) can be dangerous
  - If interrupts are already disabled prior to its invocation, local_irq_enable( ) unconditionally enables interrupts after they are off
- local_irq_save/restore() resolves the problem, to restore interrupts to a previous state

Disabling Interrupts in the Kernel

- Disable and enable only a specific interrupt line for the entire system
  - Mask out an interrupt line

```
void disable_irq(unsigned int irq);
void disable_irq_nosync(unsigned int irq);
void enable_irq(unsigned int irq);
void synchronize_irq(unsigned int irq);
```

  - disable_irq() does not return until any currently executing handler completes
  - disable_irq_nosync() does not wait for current handlers to complete
  - synchronize_irq() waits for a specific interrupt handler to exit

Atomic Operations

- **Atomic operations provide instructions that execute atomically—without interruption**

Thread 1	Thread 2
get, increment, and store i (7 -> 8)	—
—	get, increment, and store i (8 -> 9)

Thread 1	Thread 2
—	get, increment, and store i (7 -> 8)
get, increment, and store i (8 -> 9)	—

Atomic Integer Operations

- **A special data type <linux/types.h>**

```
typedef struct {
    int counter;
} atomic_t;
```

- **Usage**

```
atomic_t v = ATOMIC_INIT(0); /* define v and initialize it to zero */

atomic_set(&v, 4); /* v = 4 (atomically) */
atomic_add(3, &v); /* v = v + 3 (atomically) */
atomic_inc(&v);    /* v = v + 1 (atomically) */
```

Atomic Integer Operations

➢ Real code (ARM)

arch/arm/include/asm/atomic.h

```
ATOMIC_OP(add, +=, add)
ATOMIC_OP(sub, -=, sub)
ATOMIC_OP(and, &=, and)
ATOMIC_OP(andnot, &= ~, bic)
ATOMIC_OP(or,  |=, orr)
ATOMIC_OP(xor, ^=, eor)

#define ATOMIC_OP(op, c_op, asm_op)
static inline void atomic_##op(int i, atomic_t *v)
{
        unsigned long tmp;
        int result;

        prefetchw(&v->counter);
        __asm__ __volatile__("@ atomic_" #op "\n"
"1:     ldrex   %0, [%3]\n"
"       " #asm_op "     %0, %0, %4\n"
"       strex   %1, %0, [%3]\n"
"       teq     %1, #0\n"
"       bne     1b"
        : "=&r" (result), "=&r" (tmp), "+Qo" (v->counter)
        : "r" (&v->counter), "Ir" (i)
        : "cc");
}
```

unused; it is mnemonic

prefetches the cacheline at the given address for write

ldrex (load exclusive): load data from memory to register and set “exclusive” flag

strex (store exclusive): store register value to memory if “exclusive” flag is set

Atomic Integer Operations

Operations and Description
• ATOMIC_INIT(int i): At declaration, initialize to i
• int atomic_read(atomic_t *v): Atomically read the integer value of v
• void atomic_set(atomic_t *v, int i): Atomically set v equal to i
• void atomic_add(int i, atomic_t *v): Atomically add i to v
• void atomic_sub(int i, atomic_t *v): Atomically subtract i from v
• int atomic_sub_and_test(int i, atomic_t *v): Atomically subtract i from v and return true if the result is zero; otherwise false
• int atomic_add_negative(int i, atomic_t *v): Atomically add i to v and return true if the result is negative; otherwise false
• int atomic_add_return(int i, atomic_t *v): Atomically add i to v and return the result
• int atomic_sub_return(int i, atomic_t *v): Atomically subtract i from v and return the result

Spin Locks

- Spin lock is a mutual exclusion mechanism where a process busy-waits until the lock becomes available
- Spin locks are architecture-dependent and implemented in assembly
  - The architecture-dependent code is defined in <asm/spinlock.h>
  - The interfaces are defined in <linux/spinlock.h>
- Usage

```
DEFINE_SPINLOCK(lock);

spin_lock(&lock);
/* critical region */
spin_unlock(&lock);
```


Acquiring a Spin Lock in an ISR

- Spin locks can be used in interrupt handlers

```
DEFINE_SPINLOCK(isr_lock);
unsigned long flags;

spin_lock_irqsave(&isr_lock, flags);
/* critical region */
spin_unlock_irqrestore(&isr_lock, flags);
```

- If a lock is used in an interrupt handler, you must also disable local interrupts
  - Otherwise, it is possible for an interrupt handler to reacquire the lock

Spin Lock Implementation (Single Core)

➢ spin_lock(), if CONFIG_SMP is not set

include/linux/spinlock.h

```
static __always_inline void spin_lock(spinlock_t *lock)
{
    raw_spin_lock(&lock->rlock);
}

#define raw_spin_lock(lock)     _raw_spin_lock(lock)
```

include/linux/spinlock_api_up.h

```
#define _raw_spin_lock(lock)            __LOCK(lock)

#define __LOCK(lock) \
  do { preempt_disable(); ___LOCK(lock); } while (0)

#define ___LOCK(lock) \
  do { __acquire(lock); (void)(lock); } while (0)
```


Spin Lock Implementation (Single Core)

➢ spin_lock_irqsave(), if CONFIG_SMP is not set

include/linux/spinlock.h

```
#define spin_lock_irqsave(lock, flags)                          \
do {                                                            \
        raw_spin_lock_irqsave(spinlock_check(lock), flags);     \
} while (0)

#define raw_spin_lock_irqsave(lock, flags)          \
    do {                                            \
        typecheck(unsigned long, flags);            \
        _raw_spin_lock_irqsave(lock, flags);        \
    } while (0)
```

include/linux/spinlock_api_up.h

```
#define _raw_spin_lock_irqsave(lock, flags) __LOCK_IRQSAVE(lock, flags)

#define __LOCK_IRQSAVE(lock, flags) \
  do { local_irq_save(flags); __LOCK(lock); } while (0)

#define __LOCK(lock) \
  do { preempt_disable(); ___LOCK(lock); } while (0)
```

Spin Lock Implementation (Multicore)

➢ spin_lock(), if CONFIG_SMP is set

include/linux/spinlock_api_smp.h

```
#define _raw_spin_lock(lock) __raw_spin_lock(lock)

static inline void __raw_spin_lock(raw_spinlock_t *lock)
{
    preempt_disable();
    spin_acquire(&lock->dep_map, 0, 0, _RET_IP_);
    LOCK_CONTENDED(lock, do_raw_spin_trylock, do_raw_spin_lock);
}
```

include/linux/spinlock.h

```
static inline void do_raw_spin_lock(raw_spinlock_t *lock) __acquires(lock)
{
    __acquire(lock);
    arch_spin_lock(&lock->raw_lock);
}
```


Spin Lock Implementation (Multicore)

➢ spin_lock_irqsave(), if CONFIG_SMP is set

include/linux/spinlock_api_smp.h

```
#define _raw_spin_lock_irqsave(lock) __raw_spin_lock_irqsave(lock)

static inline unsigned long __raw_spin_lock_irqsave(raw_spinlock_t *lock)
{
    unsigned long flags;

    local_irq_save(flags);
    preempt_disable();
    spin_acquire(&lock->dep_map, 0, 0, _RET_IP_);

    LOCK_CONTENDED(lock, do_raw_spin_trylock, do_raw_spin_lock);

    return flags;
}
```

include/linux/spinlock.h

```
static inline void do_raw_spin_lock(raw_spinlock_t *lock) __acquires(lock)
{
    __acquire(lock);
    arch_spin_lock(&lock->raw_lock);
}
```

Architecture Specific Implementation

➢ Simple Test-and-Set spin lock

- **Lock is an integer (initialized to 0)**
  - 0: unlocked
  - 1: locked
- **Test-and-Set atomically sets lock to 1 and returns the previous value of the lock**

➢ Pseudo code

```
void arch_spin_lock(atomic_t *lock) {
    while (TestAndSet(lock) == 1);
}

void arch_spin_unlock(atomic_t *lock) {
    *lock = 0;
}
```


Architecture Specific Implementation

➢ Real code (ARM)

arch/arm/include/asm/spinlock.h

```
static inline void arch_spin_lock(arch_spinlock_t *lock)
{
    unsigned long tmp;
    u32 newval;
    arch_spinlock_t lockval;

    prefetchw(&lock->slock);
    __asm__ __volatile__(
"1: ldrex   %0, [%3]\n"
"   add %1, %0, %4\n"
"   strex   %2, %1, [%3]\n"
"   teq %2, #0\n"
"   bne 1b"
    : "=&r" (lockval), "=&r" (newval), "=&r" (tmp)
    : "r" (&lock->slock), "I" (1 << TICKET_SHIFT)
    : "cc");

    while (lockval.tickets.next != lockval.tickets.owner) {
        wfe();
        lockval.tickets.owner = ACCESS_ONCE(lock->tickets.owner);
    }

    smp_mb();
}
```

prefetches the cacheline at the given address for write

obtain a ticket atomically

wait for the order

Semaphores

➢ Semaphores are sleeping locks

Method	Description
`down_interruptible (struct semaphore *)`	Tries to acquire the given semaphore and enter interruptible sleep if it is contended
`down(struct semaphore *)`	Tries to acquire the given semaphore and enter uninterruptible sleep if it is contended
`down_trylock(struct semaphore *)`	Tries to acquire the given semaphore and immediately return nonzero if it is contended
`up(struct semaphore *)`	Releases the given semaphore and wakes a waiting task, if any

- **The function down_interruptible() attempts to acquire the given semaphore**
- **If the semaphore is unavailable, it places the calling process to sleep in the TASK_INTERRUPTIBLE state**
- **The task can be awakened with a signal**

Semaphore Data Structure

include/linux/semaphore.h

```
struct semaphore {
    raw_spinlock_t    lock;
    unsigned int      count;
    struct list_head  wait_list;
}
```

- lock → lock for wait queue
- count → the number of permissible simultaneous holders of semaphores
- wait_list → wait queue for tasks which failed to acquire the semaphore

The down() and up() Functions

- **Semaphore implementations rely on spinlocks**

kernel/locking/semaphore.c

```
void down(struct semaphore *sem)
{
    unsigned long flags;

    raw_spin_lock_irqsave(&sem->lock, flags);
    if (likely(sem->count > 0))
        sem->count--;
    else
        __down(sem);
    raw_spin_unlock_irqrestore(&sem->lock, flags);
}
```

In most cases, down() just decreases the counter because there is no competition

If other tasks try to acquire the same semaphore when its counter is not positive, put the task into the semaphore's wait list and sleep

```
void up(struct semaphore *sem)
{
    unsigned long flags;

    raw_spin_lock_irqsave(&sem->lock, flags);
    if (likely(list_empty(&sem->wait_list)))
        sem->count++;
    else
        __up(sem);
    raw_spin_unlock_irqrestore(&sem->lock, flags);
}
```

In most cases, up() just increases the counter because there is no competition

If there are tasks waiting for the same semaphore, wake up the oldest task from the semaphore's wait list

Constraint on Semaphores

- **Semaphore should not be used while holding a spin lock**
  - **Because spin lock disables preemption, if down() makes a task sleep, the task cannot wake**

Mutexes

- **Mutexes are simpler sleeping locks**
  - **A mutex is a binary semaphore**

Method	Description
`mutex_lock(struct mutex *)`	Locks the given mutex; sleeps if the lock is unavailable
`mutex_unlock(struct mutex *)`	Unlocks the given mutex
`mutex_trylock(struct mutex *)`	Tries to acquire the given mutex; returns one if successful and the lock is acquired and zero otherwise
`mutex_is_locked (struct mutex *)`	Returns one if the lock is locked and zero otherwise

Mutex Data Structure

include/linux/mutex.h

```
struct mutex {
    atomic_t          count;
    spinlock_t        wait_lock;
    struct list_head  wait_list;
}
```

- `count` → atomic counter
  1: unlocked
  0: locked
  < 0: locked, possible waiters
- `wait_lock` → lock for wait queue
- `wait_list` → wait queue for tasks which failed to acquire the mutex

Mutexes

- Semaphores are generic and do not impose many usage constraints
  - This makes them useful for managing exclusive access in unclear situations
- Mutexes are strict and narrower than semaphores
  - Whoever locked a mutex must unlock it
  - Recursive locks and unlocks are not allowed
  - A task cannot exit while holding a mutex
  - A mutex cannot be acquired by an interrupt handler or bottom half
  - A mutex can be managed only via the official API

Linux Kernel Synchronization II

Euiseong Seo

College of Software
Sungkyunkwan University

* This material was originally written by Prof. Minsoo Ryu at Hanyang University for System Society of KIISE

Outline

1 More Synchronization Mechanisms — Page 3

2 Read-Copy-Update — Page 27

3 Q & A — Page 36

Ticket Spinlocks

- **Ticket spin lock**
  - **Linux version >= 2.6.25**
  - **Lock contenders acquire the lock in FIFO manner**

- **spin_lock()**
  - **A task atomically reads and increases queue ticket by one**
  - **It then atomically compares the read value (the previous queue ticket) and de-queue ticket**
    - If they are the same, lock acquired and enters critical section
    - If they are not the same, the lock is held by another task
- **spin_unlock()**
  - **A task atomically increases the de-queue ticket**

Spinlocks

- **Lowest-level mutual exclusion mechanism**
  - **Spinlock is represented by an integer value**
  - **spin_lock()**
    - It acquires the lock, when lock is free
    - Otherwise, it spins ("busy-wait") until lock is free,
  - **spin_unlock()**
    - It releases the lock
- **Problem**
  - **Unfairness**
    - some threads may be starved when lock is highly contending

Ticket Spinlocks

- **Linux version >=2.6.25**
- **Fair lock to resolve starvation issue**
  - **Lock contenders acquire the lock in FIFO manner**

spinlock

Next	Owner
8bits	8bits

```
static inline void __raw_spin_lock(raw_spinlock_t *lock) {
        short inc = 0x0100;
        __asm__ __volatile__ (
            LOCK_PREFIX "xaddw %w0, %1\n"
            "1:\t"
            "cmpb %h0, %b0\n\t"
            "je 2f\n\t"
            "rep ; nop\n\t"
            "movb %1, %b0\n\t"
            "jmp 1b\n" "2:"
            :"+Q" (inc), "+m" (lock->slock) : :"memory", "cc");
}
```

```
static inline void __raw_spin_unlock(raw_spinlock_t *lock) {
        __asm__ __volatile__(
            UNLOCK_LOCK_PREFIX "incw %0"
            :"+m" (lock->slock) : :"memory", "cc");
}
```


Ticket Spinlocks

- **Linux version >=2.6.25**
- **Fair lock to resolve starvation issue**
  - **Lock contenders acquire the lock in FIFO manner**

spinlock

Next	Owner
8bits	8bits

```
void spin_lock (spinlock_t lock) {
            short inc = 0x0100;
            my_ticket = XADD(Next,inc);
            while(my_ticket != Owner) ;
}
```

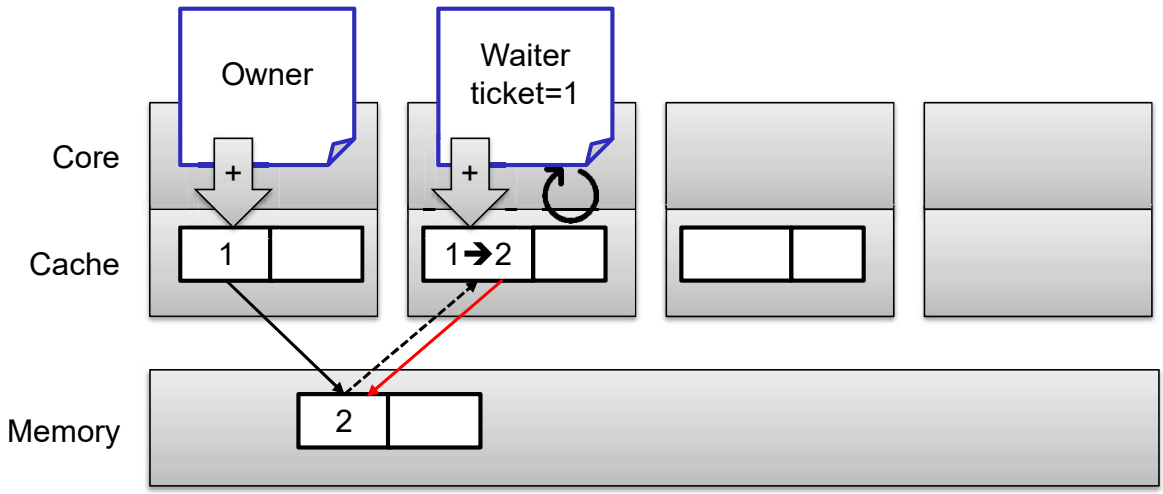
Exchange & Add

```
XADD(*addr, inc) {
    old = *addr;
    *addr++;
    return old;
}
```

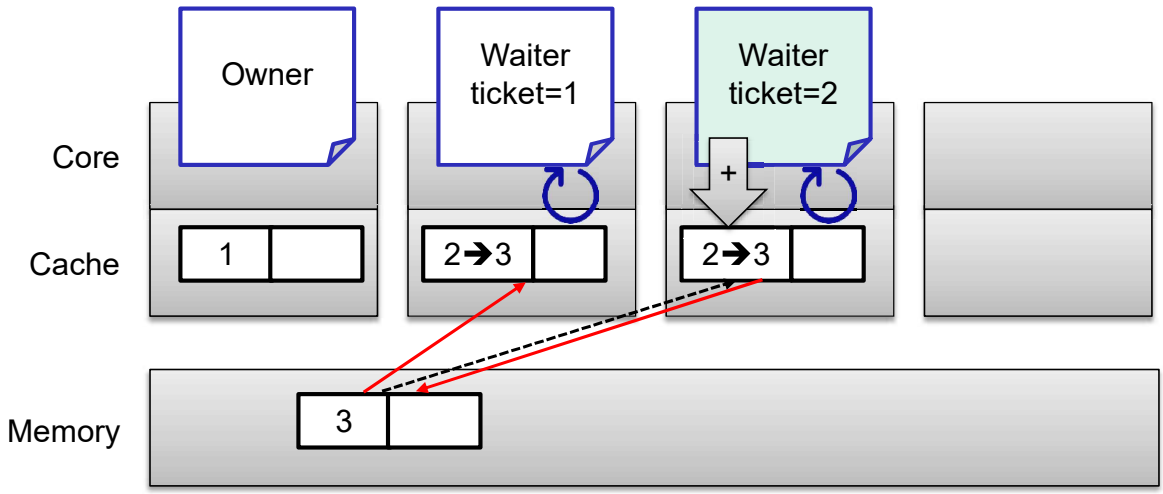
```
void spin_lock (spinlock_t lock) {
            Owner++;
}
```

- **Problem**
  - **High contention of interconnect**
    - spin_unlock invalidates cached copy of lock for waiter threads

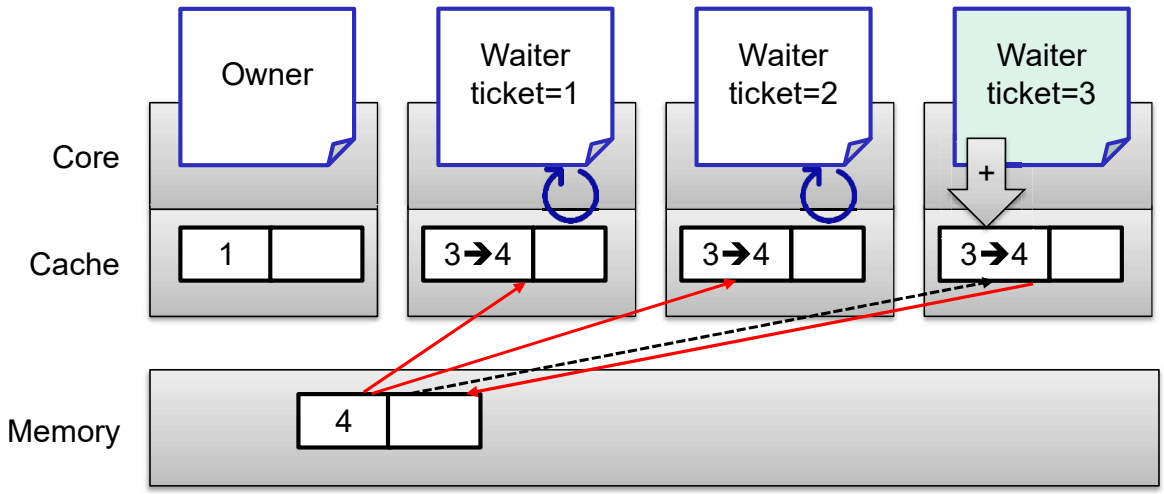
Interconnect Contention Problem



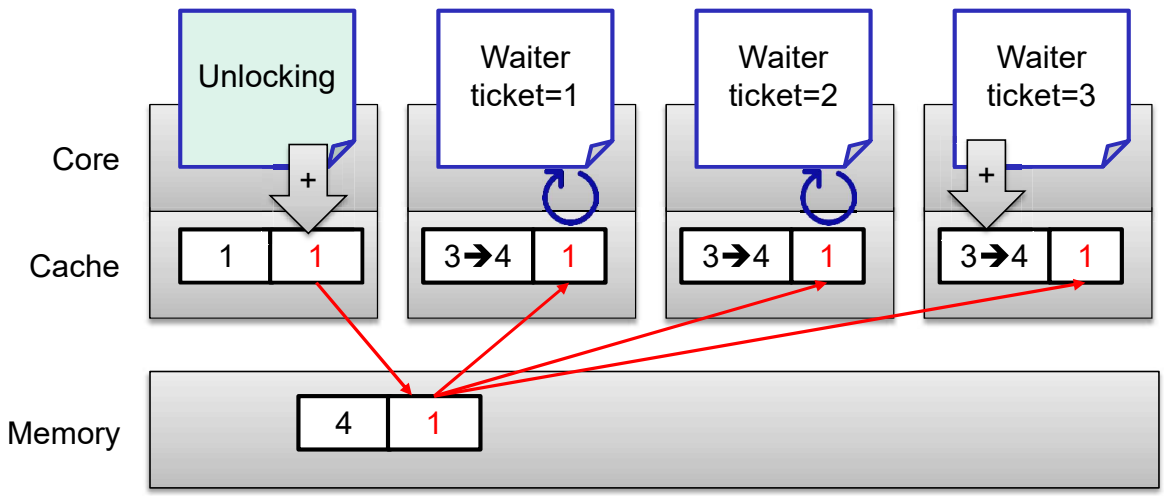
Interconnect Contention Problem



Interconnect Contention Problem

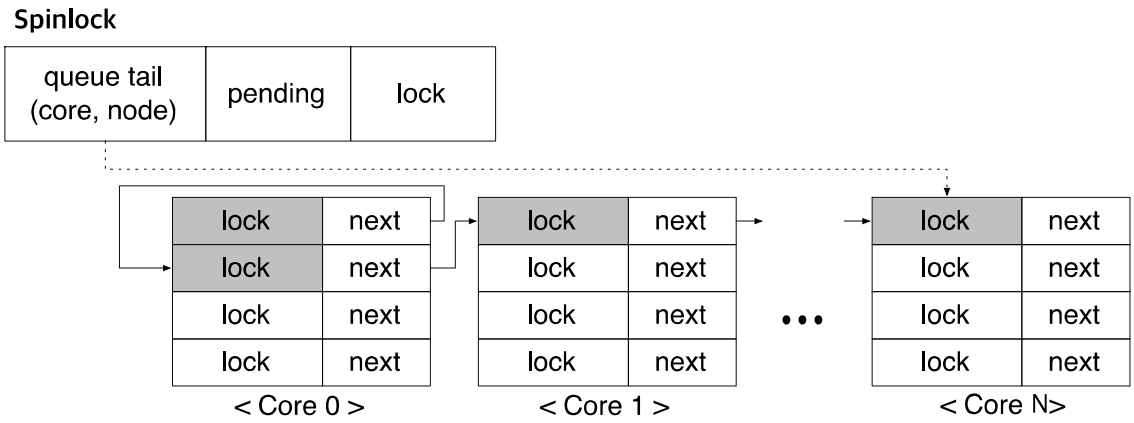


Interconnect Contention Problem

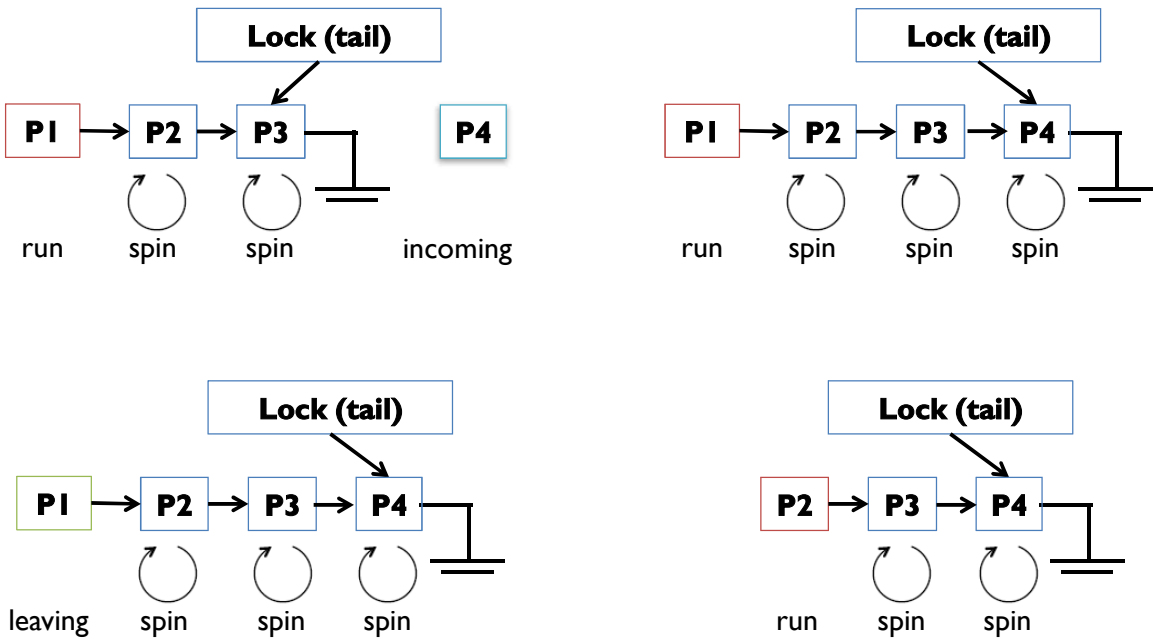


QSpinlocks (MCS locks)

- Linux version >= 3.15
- Queue-based spinlock can reduce cache invalidation
- Each waiter thread spin on a unique location
  - Data structure of qspinlock & waiter queue

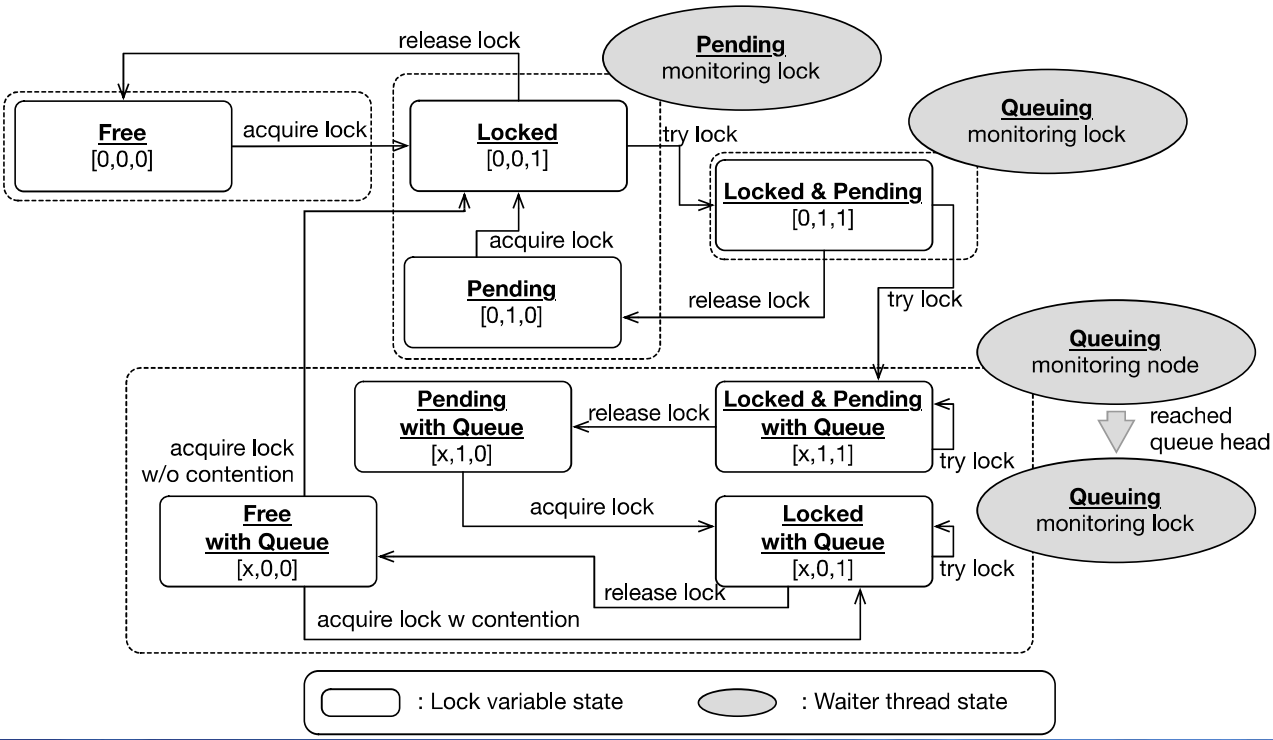


QSpinlocks (MCS locks)



QSpinlocks (MCS locks)

➢ State of spinlock variable & waiter threads



Reader-Writer Spin Locks

➢ In the reader code path

```
read_lock(&mr_rwlock);
/* critical section (read only) ... */
read_unlock(&mr_rwlock);
```

➢ In the writer code path

```
write_lock(&mr_rwlock);
/* critical section (read and write) ... */
write_unlock(&mr_lock);
```

➢ Notes

- **It is safe for multiple readers to obtain the same lock**
  - It is safe for the same thread to recursively obtain the same read lock
- **The Linux reader-writer spin locks "favor readers over writers"**

Readers-Writers Problem

- **Readers-Writers Problem**
  - **A single buffer with multiple readers and writers**
  - **Writers have exclusive access to the shared object**
    - "Single writer in CS" or "multiple readers in CS"

- **Two variants**
  - **First readers-writers problem**
    - No reader will be kept waiting until a writer has already obtained permission to use the shared object
    - No reader should wait for other readers
  - **Second readers-writers problem**
    - Once a writer is ready, that writer performs its write as soon as possible
    - If a writer is waiting to access the object, no new readers may start reading

The First Readers-Writers Problem

- **The first readers-writers problem**
  - **Give preferential treatment to readers**
  - **Writers may suffer unbounded waiting (starvation)**

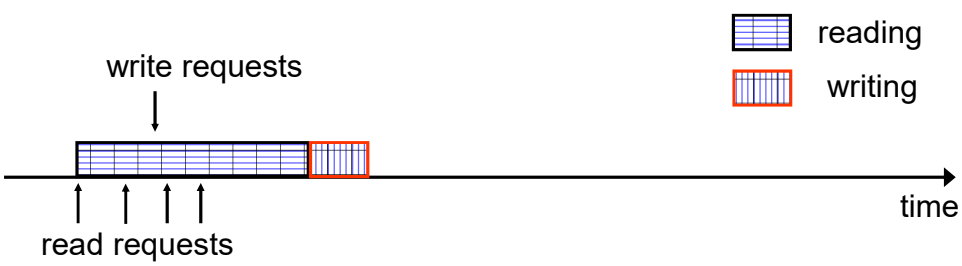
- **Shared data**
  - semaphore S, wrt;
  - **Initially,** S = 1, wrt = 1, readcount = 0

The First Readers-Writers Problem

```
/* Writer */
wait(wrt);
    …
    writing is performed
    …
signal(wrt);
```

```
/* Reader */
wait(S);
readcount++;
if (readcount == 1)
    wait(wrt);
signal(S);
    …
    reading is performed
    …
wait(S);
readcount--;
if (readcount == 0)
signal(wrt);
signal(S):
```


The First Readers-Writers Problem



The Second Readers-Writers Problem

- **The second readers-writers problem**
  - **Give preferential treatment to writers**
    - "write-preferring" or "write-biased"
  - **Readers may suffer unbounded waiting (starvation)**

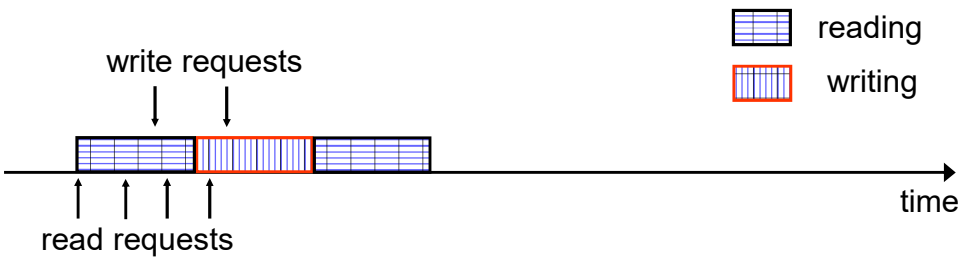
- **Shared data**
  - **semaphore S1, S2, wrt, rd, wrtpending;**
  - **Initially,**
    - **S1 = 1, S2 = 1, wrt = 1, rd = 1, wrtpending = 1,**
    - **readcount = 0, writecount = 0**

The Second Readers-Writers Problem

```
/* Writer */
wait(S2);
writecount++;
if (writecount == 1)
    wait(rd);
signal(S2);
wait(wrt)
    …
    writing is performed
    …
signal(wrt);
wait(S2);
writecount--;
if (writecount == 0)
    signal(rd);
signal(S2);
```

```
/* Reader */
wait(wrtpending);
wait(rd);
wait(S1);
readcount++;
if (readcount == 1)
    wait(wrt);
signal(S1);
signal(rd);
signal(wrtpending);
    …
    reading is performed
    …
wait(S1);
readcount--;
if (readcount == 0)
signal(wrt);
signal(S1):
```

The Second Readers-Writers Problem



Completion Variables

- **Used for synchronization between two tasks in the kernel**
  - **One task waits on the completion variable while another task performs some work**
  - **When the other task has completed the work, it uses the completion variable to wake up any waiting tasks**

Method	Description
`init_completion(struct completion *)`	Initializes the given dynamically created completion variable
`wait_for_completion(struct completion *)`	Waits for the given completion variable to be signaled
`complete(struct completion *)`	Signals any waiting tasks to wake up

  - **The idea is much the same as semaphores**

Big Kernel Lock

- BKL was used for the transition from kernel version 2.0 to 2.2
  - When SMP support was introduced in kernel version 2.0, only one task could be in the kernel at a time
  - It behaves like a spin lock
  - Now, the use of the BKL is discouraged

Function	Description
`lock_kernel ()`	Acquires the BKL.
`unlock_ kernel()`	Releases the BKL.
`kernel_ locked()`	Returns nonzero if the lock is held and zero otherwise. (UP always returns nonzero.)

Properties of BKL

- You can sleep while holding the BKL
  - The lock is automatically dropped when the task is unscheduled and reacquired when the task is rescheduled
- The BKL is a recursive lock
  - A single process can acquire the lock multiple times and not deadlock, as it would with a spin lock
  - A single thread of execution might acquire the lock recursively but must then call unlock_kernel() an equal number of times to release the lock
- You can use the BKL only in process context
  - Unlike spin locks, you cannot acquire the BKL in interrupt context

Sequential Locks

- **A newer type of lock introduced in the 2.6 kernel**
  - **Used for reading and writing shared data**
  - **Seq locks "favor writers over readers"**
- **Writers**
  - **After acquiring the seq lock, the writer increments the sequence number, both after acquiring the lock and before releasing the lock (0 → 1 → 2)**
- **Readers**
  - **Readers read the sequence number before and after reading the shared data**
    - If the seq number is odd, a writer had taken the lock
    - If the seq numbers differ, a writer has changed the data
    - In either case, readers simply retry

Sequential Locks

```
seqlock_t mr_seq_lock = DEFINE_SEQLOCK(mr_seq_lock);
```

Initially seq = 0

```
write_seqlock(&mr_seq_lock);
/* write lock is obtained... */
write_sequnlock(&mr_seq_lock);
```

writer code

```
do {
        seq = read_seqbegin(&mr_seq_lock);
        /* read data here ... */
} while (read_seqretry(&mr_seq_lock, seq));
```

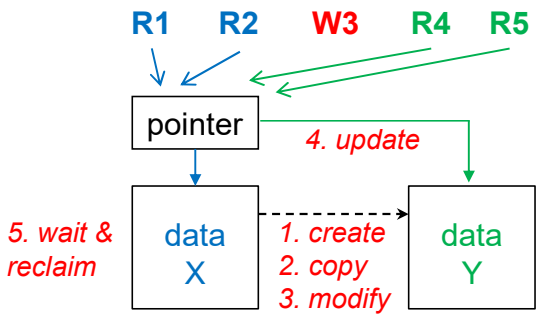
reader code

- **Seq locks are more efficient than read-write locks when there are many readers and few writers**
  - **Readers never block and writers do not wait for readers**
- **If there is too much write activity or the reader is too slow, they might livelock**

Read-Copy-Update

Read-Copy-Update

- **An alternative to reader-writer locks**
  - **Readers can access the shared data without taking a lock**
    - Even when the data is in the process of being updated
  - **A writer copies and updates the data, and changes the pointer**
    - Readers will see a valid data, but a different data
    - Pre-existing readers see the old data, new readers see the new one



RCU Interfaces

Operations	Description
rcu_read_lock()	• Mark an RCU-protected data structure so that it won't be reclaimed for the full duration of that critical section
rcu_read_unlock()	• Inform the reclaimer that the reader is exiting an RCU read-side critical section
rcu_dereference()	• The reader uses rcu_dereference to fetch an RCU-protected pointer • The value returned by rcu_dereference is valid only within the enclosing RCU read-side critical section
rcu_assign_pointer()	• The updater uses this function to “atomically assign” a new value to an RCU-protected pointer, in order to safely communicate the change in value from the updater to the reader
synchronize_rcu()	• It waits until all pre-existing RCU read-side critical sections on all CPUs have completed

Example

Reader

```
int id, price;
struct item *local_ptr;

rcu_read_lock();
local_ptr = rcu_dereference(shared_ptr);
id = local_ptr->id;
price = local_ptr->price;
rcu_read_unlock();

printf(“%d %d\n”, id, price);
```

Writer

```
struct item *local_ptr, *old_ptr;
local_ptr = kmalloc(...);
local_ptr->id = 46703;
local_ptr->price = 1500;

old_ptr = shared_ptr;
rcu_assign_pointer(shared_ptr, local_ptr);
synchronize_rcu();
free(old_ptr);
```

RCU Process

Reader 1	Writer	Reader 2
	assign(p, X)	
lock()		
	assign(p, Y)	
		lock()
		dereference(p) -> Y
dereference(p) -> X		
unlock()		
		unlock()

How to check there is no readers who want X?
In other words, how long writer maintains X and Y?

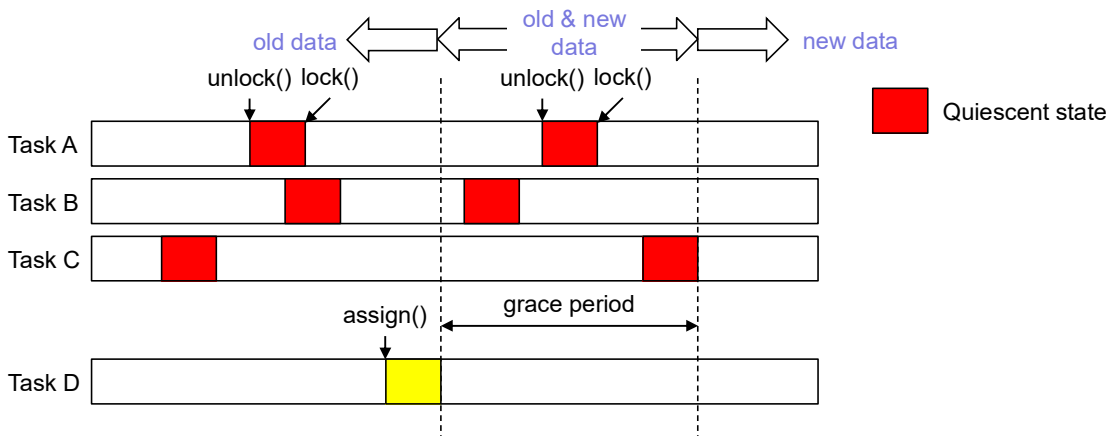
synchronize_rcu()

- **Two ways for writer**
  - **Simply blocks until when readers are done**
  - **Registers a callback to be invoked after all ongoing RCU read-side critical sections have completed (call_rcu())**

- **Deciding when to reclaim**
  - **Non-preemptive RCU**
    - Wait for at least one grace period to elapse
  - **Preemptive RCU**
    - Lock() increments the counter associated with RCU
    - Unlock() decrements the counter
    - Synchronize() checks whether the counter is zero

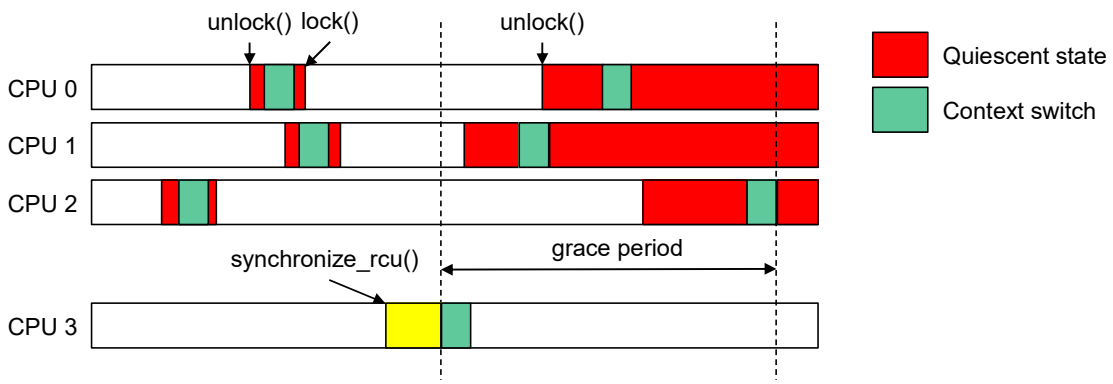
Quiescent State and Grace Period

- **Quiescent state**
  - Any statement that is not within an RCU read-side critical section
- **Grace period**
  - Any time period during which each thread resides at least once in a quiescent state is called a *grace period*



Grace Period with Nonpreemptive RCU

- **Detecting a grace period**
  - Readers cannot block, sleep or be preempted inside a critical section
  - Thus, context switch implies the task has experienced a quiescent state
  - Synchronize() waits for every other CPU passes through a context switch



Notes on RCUs

- RCUs allow readers to use much lighter-weight synchronization
- RCU works best in situations with mostly reads and few updates
- To use RCUs, data structure must be dynamically allocated and referenced by a single pointer

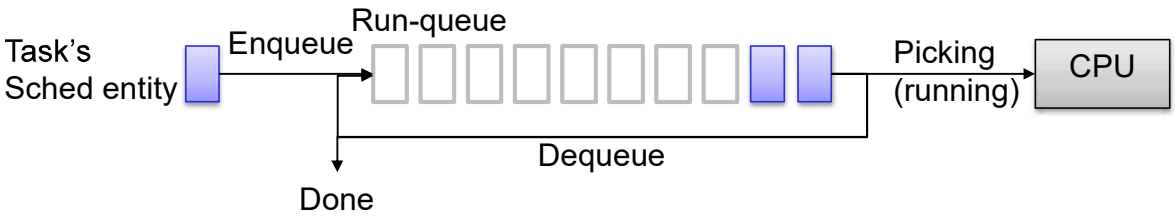
Scheduling Lab.
My Own Scheduler

Sungkyunkwan University
Woorak Park
(E-Mail: woorak.park@csl.skku.edu)

BACKGROUND

A New Scheduler in Linux Kernel

- Goal
  - Making a new scheduler
  - = designing scheduling policies, and implementing them
- What to implement
  - run queues
  - scheduling entities
  - scheduler classes and functions to fill them



Definition of a New Run-queue

```
<kernel/sched/sched.h>
…
/* Definition of our scheduling run-queue
struct mysched_rq {
        unsigned int nr_running;
        struct list_head queue;

        unsigned int prev_pick_time;
};
…
struct rq {
        …
        struct cfs_rq cfs;
        struct rt_rq rt;
        struct dl_rq dl;
        /* Declaring our scheduling run-
queue */
        struct mysched_rq mysched;
        …
}
…
```

```
<kernel/sched/core.c>
…
/* When booting system, our run-
queue should be initialized in
scheduling initialization function
*/
void __init sched_init(void) {
        …
        init_dl_rq(&rq->dl);
        init_mysched_rq(&rq-
>mysched);
        …
}
```

Definition of a Scheduling Entity

```
<include/linux/sched.h>
…
/* For Inserting scheduling entity into
our run-queue */
struct sched_mysched_entity {
        struct list_head run_list;
};
…
struct task_struct {
        …
        int on_rq;

        int prio, static_prio,
normal_prio;
        unsigned int rt_priority;
        const struct sched_class
*sched_class;
        struct sched_entity se;
        struct sched_rt_entity rt;
        /* Declaring our scheduling
entity */
        struct sched_mysched_entity
mysched;
```

```
<kernel/sched/core.c>

/* When forking new thread(task), our sched_entity
should be initialized */
static void __sched_fork(unsigned long clone_flags,
                         struct task_struct *p) {
        …
        INIT_LIST_HEAD(&p->rt.run_list);
        INIT_LIST_HEAD(&p->mysched.run_list);
        …
}
```


Defining and Attaching New Class

```
<include/uapi/linux/sched.h>
…
#define SCHED_IDLE      5
#define SCHED_DEADLINE  6
#define SCHED_MYSCHED   7
…
```

```
<kernel/sched/rt.c>
…
/*Connet RT scheduling class to our
scheduling class*/
const struct sched_class
rt_sched_class = {
        .next =
&mysched_sched_class,
        …
```

```
<kernel/sched/mysched.c>

/* This routine is called when a task migrates between
classes */
static void switched_to_mysched(struct rq *rq, struct
task_struct *p) {
        resched_curr(rq);
}
/* Our scheduling class is defined flowing below*/
const struct sched_class mysched_sched_class= {
/*Our scheduling class is connected to fair scheduling
class*/
        .next           = &fair_sched_class,
        .enqueue_task           =
enqueue_task_mysched,
        .dequeue_task           =
dequeue_task_mysched,

        .check_preempt_curr                     =
check_preempt_curr_mysched,

        .pick_next_task     = pick_next_task_mysched,
        .put_prev_task          =
put_prev_task_mysched,

#ifdef CONFIG_SMP
        .select_task_rq     = select_task_rq_mysched,
#endif

        .set_curr_task          =
set_curr_task_mysched,
        .task_tick              = task_tick_mysched,
        .prio_changed           =
prio_changed_mysched,
        .switched_to            =
switched_to_mysched,
```

sched_class_highest → stop_sched_class (.next) → dl_sched_class (.next) → rt_sched_class (.next) → mysched_sched_class (.next) → fair_sched_class (.next) → idle_sched_class (.next=NULL)

mysched Scheduling Class Function Form

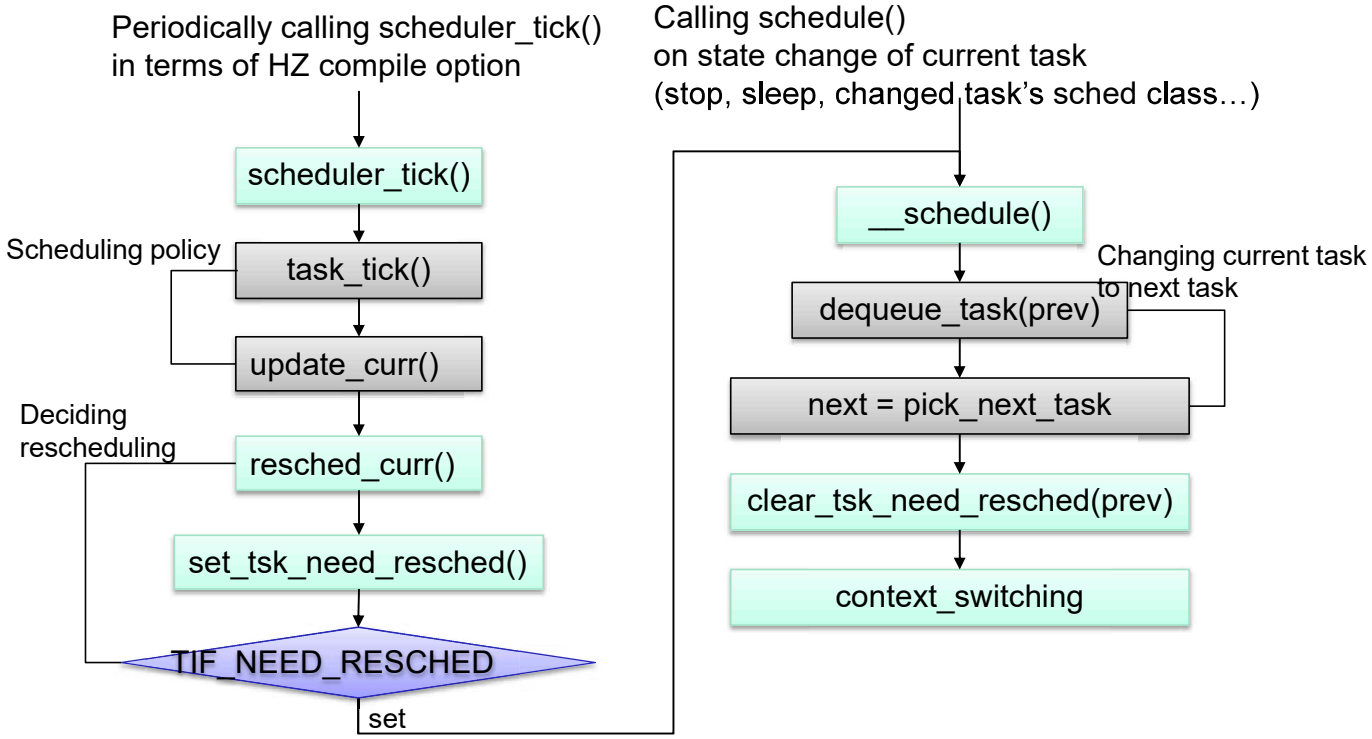
```
<kernel/sched/mysched.c>
/* … */
#include "sched.h"
#include <linux/slab.h>

/* For accessing variable in source code   */
// extern const struct sched_class mysched_sched_class;
/* For accessing function in source code   */
// extern void init_mysched_rq(struct mysched_rq *mysched_rq);

void init_mysched_rq(struct mysched_rq *mysched_rq)
{
      printk(KERN_INFO "***[MYSCHED] Mysched class is online\n");
}
static void update_curr_mysched(struct rq *rq){}
static void enqueue_task_mysched(struct rq *rq, struct task_struct *p, int
flags) {}
static void dequeue_task_mysched(struct rq *rq, struct task_struct *p, int
flags) {}
```

[continued at the following page…]

mysched Scheduling Class Function Form

```
continuing <kernel/sched/mysched.c>

static void check_preempt_curr_mysched(struct rq *rq, struct task_struct *p,
int flags) {}

struct task_struct *pick_next_task_mysched(struct rq *rq, struct task_struct
*prev) {
         put_prev_task(rq, prev);
         return NULL;
}
static void put_prev_task_mysched(struct rq *rq, struct task_struct *p) { }
static int select_task_rq_mysched(struct task_struct *p, int cpu, int sd_flag,
int flags) {
         return task_cpu(p);
}
static void set_curr_task_mysched(struct rq *rq) { }
static void task_tick_mysched(struct rq *rq, struct task_struct *p, int
queued) { }
static void prio_changed_mysched(struct rq *rq, struct task_struct *p, int
oldprio) { }
```

Approximately Scheduling Flow



Modify __setscheduler()

```
<kernel/sched/core.c>

static void __setscheduler (struct rq *rq, struct task_struct *p, const struct
sched_attr *attr, bool keep_boost)
{
        …
        if (dl_prio(p->prio))
                p->sched_class = &dl_sched_class;
        else if (rt_prio(p->prio))
                p->sched_class = &rt_sched_class;
        else if (mysched_policy(attr->sched_policy))
                p->sched_class = &mysched_sched_class;

                p->sched_class = &fair_sched_class;
}
```

```
<kernel/sched/sched.h>
…
static inline int mysched_policy(int
policy)
{
        return policy ==
SCHED_MYSCHED;
}
…
```

[continued at the following page…]

Modify __setscheduler()

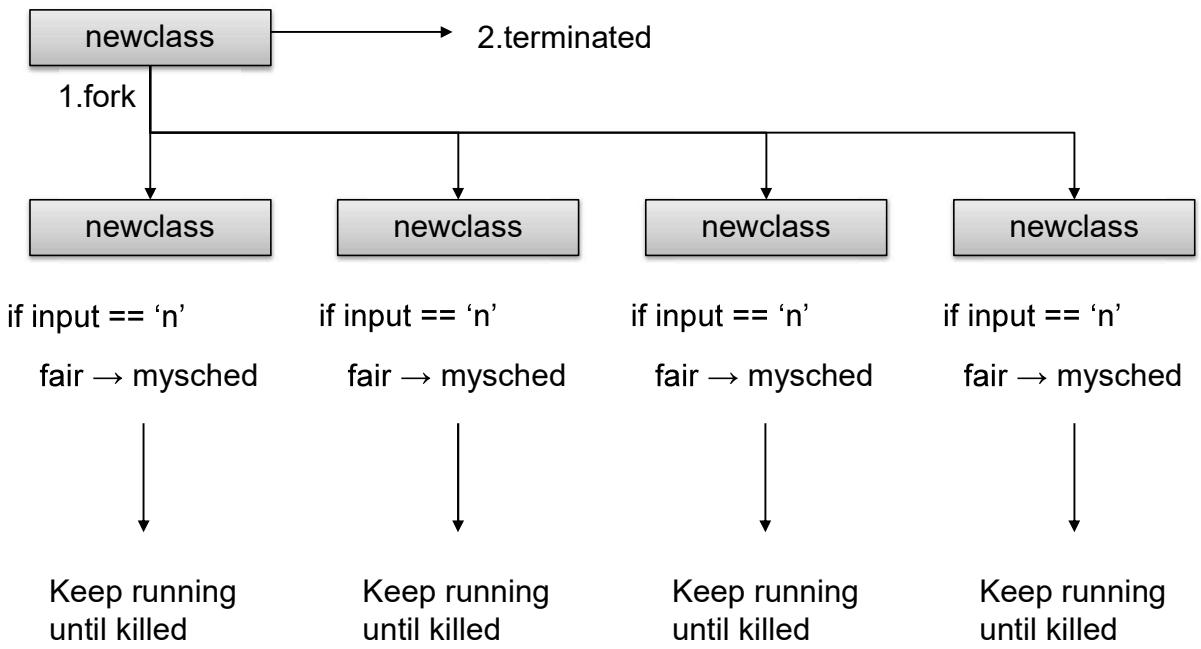
```
continuing <kernel/sched/core.c>

static int__sched_setscheduler(struct task_struct *p, const struct sched_attr
*attr, bool user, bool pi) {
        …
recheck:
        /* double check policy once rqlock held */
        if (policy < 0) {
                reset_on_fork = p->sched_reset_on_fork;
                policy = oldpolicy = p->policy;
        } else {
                reset_on_fork = !!(attr->sched_flags& SCHED_FLAG_RESET_ON_FORK);
                if (policy != SCHED_DEADLINE && policy != SCHED_FIFO
                        && policy != SCHED_RR && policy != SCHED_NORMAL
                        && policy != SCHED_BATCH && policy != SCHED_IDLE
                        && policy != SCHED_MYSCHED)
                        return -EINVAL;
        }
        …
}
```


Makefile

```
<kernel/sched/Makefile>

ifdef CONFIG_FUNCTION_TRACER
CFLAGS_REMOVE_clock.o = $(CC_FLAGS_FTRACE)
endif

ifneq ($(CONFIG_SCHED_OMIT_FRAME_POINTER),y)
# According to Alan Modra <alan@linuxcare.com.au>, the -fno-omit-
frame-pointer is
# needed for x86 only. Why this used to be enabled for all
architectures is beyond
# me. I suspect most platforms don't need this, but until we know
that for sure
# I turn this off for IA-64 only. Andreas Schwab says it's also
needed on m68k
# to get a correct value for the wait-channel (WCHAN in ps). --davidm
CFLAGS_core.o := $(PROFILING) -fno-omit-frame-pointer
endif

obj-y += core.o loadavg.o clock.o cputime.o
obj-y += idle_task.o fair.o rt.o deadline.o stop_task.o mysched.o
obj-y += wait.o completion.o idle.o
…
```

ASSIGNMENT

Introduction of Test Program (newclass)



How it Changes its Scheduling Class

```
…
if (c == 'n') {
        printf("***[NEWCLASS] Select new scheduling class \n");
        /* set attributes for SCHED_DEADLINE */
        attr.size = sizeof(attr);
        attr.sched_policy = SCHED_MYSCHED;
        attr.sched_flags = 0;

        attr.sched_period = 0;
        attr.sched_runtime = 0;
        attr.sched_deadline = 0;

        attr.sched_nice = 0;                    // for SCHED_NORMAL and SCHED_BATCH
        attr.sched_priority = 0;      // for SCHED_FIFO and SCHED_RR

        ret = sched_setattr(my_pid, &attr, flags);
        if (ret != 0) {
                perror("sched_setattr");
                exit(1);
        }
}
else if
…
```


Making Test Processes

- Command

```
:~/workspace/app$ make
gcc-g -static -o newclass newclass.c

:~/workspace/app$ ./newclass f
Child's PID = 1740
***[NEWCLASS] Select CFS class
Child's PID = 1741
***[NEWCLASS] Select CFS class
Child's PID = 1742
***[NEWCLASS] Select CFS class
Child's PID = 1743
***[NEWCLASS] Select CFS class
forking 4 tasks is completed
```

```
:~/workspace/app$ ps-ac
 PID  CLS  PRI  TTY     TIME       CMD
1740  TS   19   pts/2   00:00:00   newclass
1741  TS   19   pts/2   00:00:00   newclass
1742  TS   19   pts/2   00:00:00   newclass
1743  TS   19   pts/2   00:00:00   newclass

:~/workspace/app$ kill 'pidof newclass'
```

Assignment #1

➢ Implementation

- **enqueue_task_mysched 함수를 작성하고 다음의 결과를 출력**

➢ Checklist

```
:~/workspace/app$ dmesg -D
:~/workspace/app$ ./newclass n
Child's PID = 1740
***[NEWCLASS] Select new scheduling class
Child's PID = 1741
***[NEWCLASS] Select new scheduling class
Child's PID = 1742
***[NEWCLASS] Select new scheduling class
Child's PID = 1743
***[NEWCLASS] Select new scheduling class
forking 4 tasks is completed
```

```
:~/workspace/app$ ps-ac
 PID  CLS  PRI  TTY      TIME      CMD
1740  #7   19   pts/2   00:00:00  newclass
1741  #7   19   pts/2   00:00:00  newclass
1742  #7   19   pts/2   00:00:00  newclass
1743  #7   19   pts/2   00:00:00  newclass
1660  TS   19   pts/2   00:00:00     ps

:~/workspace/app$ dmesg | grep MYSCHED
[     0.000000] ***[MYSCHED] Mysched class is online
[4229.354982] ***[MYSCHED] Enqueue: nr_running = 1 pid = 1740
[4229.455901] ***[MYSCHED] Enqueue: nr_running = 2 pid = 1741
[4229.556478] ***[MYSCHED] Enqueue: nr_running = 3 pid = 1742
[4229.656907] ***[MYSCHED] Enqueue: nr_running = 4 pid = 1743
```


Hint #1

➢ Variables

```
struct mysched_rq {
        unsigned int nr_running;
        struct list_head queue;
};
```

```
struct sched_mysched_entity {
        struct list_head run_list;
};
```

➢ List

- **Initilization -> init_mysched_rq 함수**

  INIT_LIST_HEAD(…);

- **Addition**

  list_add_tail(…);

Assignment #2

- **Assignment #2**
  - **pick_next_task_mysched 함수 구현**
    - **MYSCHED 런큐 내의 태스크를 선택하여 실행하도록 구현**
  - **dequeue_task_mysched 함수 구현**
    - **태스크가 실행되어 sleep() 호출에 의해 런큐에서 제거될 수 있도록 구현**
    - **제거된 태스크는 다시 깨어나 런큐로 돌아올 수 있어야 함**
- **Checklist**

```
:~/workspace/app$ dmesg -D
:~/workspace/app$ ./newclass n
Child's PID = 1740
***[NEWCLASS] Select new scheduling class
Child's PID = 1741
***[NEWCLASS] Select new scheduling class
Child's PID = 1742
***[NEWCLASS] Select new scheduling class
Child's PID = 1743
***[NEWCLASS] Select new scheduling class
forking 4 tasks is completed
```

```
:~/workspace/app$ ps -ac
 PID   CLS  PRI  TTY    TIME       CMD
1740   #7   19   pts/2  00:00:00   newclass
1741   #7   19   pts/2  00:00:00   newclass
1742   #7   19   pts/2  00:00:00   newclass
1743   #7   19   pts/2  00:00:00   newclass
1660   TS   19   pts/2  00:00:00      ps
```

```
:~/workspace/app$ dmesg | grep MYSCHED
[ 0.000000] ***[MYSCHED] Myschedclass is online
[4229.354982] ***[MYSCHED] Enqueue: nr_running = 1pid = 1740
[4229.455901] ***[MYSCHED] Enqueue: nr_running = 2 pid = 1741
[4229.556478] ***[MYSCHED] Dequeue: nr_running = 1 pid = 1740
[4229.656907] ***[MYSCHED] Enqueue: nr_running = 2 pid = 1740
```


Hint #2

- **List**

  list_del_init(…);
  container_of(…);
  list_empty(…);

- **pick_next_task**
  - **It must call put_prev_task(…) function**
    - **Remind CFS put_prev_task_fair()**

- **FIFO generation**
  - **mysched_rq의 queue에 처음 들어온 것을 선택한다.**
    mysched_rq->queue.next

Assignment #3

- **Fixed-quantum Scheduler**
  - **테스트 프로그램 newclass에서 자식 프로세스가 sleep()하지 않도록 변경**
  - **MYSCHED 클래스 변경**
    - 30ms를 주기로 MYSCHED 태스크를 스케줄링 할 수 있도록 구현
    - MYSCHED 런큐 안에 태스크의 개수가 4개 미만일 경우 MYSCHED 태스크를 선택하지 않도록 변경
    - 100번의 Pick이 이루어진 이후 더 이상 MYSCHED 태스크를 선택하지 않도록 변경
    - 구현 함수
      - ✓ pick_next_task_mysched
      - ✓ check_preempt_mysched
      - ✓ task_tick_mysched

Assignment #3

- **Checklist**

```
:~/workspace/app$ dmesg -D
:~/workspace/app$ ./newclass n
Child's PID = 1740
***[NEWCLASS] Select new scheduling class
Child's PID = 1741
***[NEWCLASS] Select new scheduling class
Child's PID = 1742
***[NEWCLASS] Select new scheduling class
Child's PID = 1743
***[NEWCLASS] Select new scheduling class
forking 4 tasks is completed

:~/workspace/app$ ps –ac
 PID  CLS  PRI  TTY     TIME       CMD
1740  #7    19   pts/2   00:00:00   newclass
1741  #7    19   pts/2   00:00:00   newclass
1742  #7    19   pts/2   00:00:00   newclass
1743  #7    19   pts/2   00:00:00   newclass

:~/workspace/app$ dmesg | grep MYSCHED > ~/log

:~/workspace/app$ vim ~/log
```

```
[    0.000000] ***[MYSCHED] Mysched class is online
[ 3078.756401] ***[MYSCHED] Enqueue: nr_running = 1 pid = 288
[ 3078.857454] ***[MYSCHED] Enqueue: nr_running = 2 pid = 289
[ 3078.957924] ***[MYSCHED] Enqueue: nr_running = 3 pid = 290
[ 3079.058460] ***[MYSCHED] Enqueue: nr_running = 4 pid = 291
31ms {  [ 3079.058514] ***[MYSCHED] Pick new task: [1] pid = 290
29ms {  [ 3079.089603] ***[MYSCHED] Pick new task: [2] pid = 289
        [ 3079.119595] ***[MYSCHED] Pick new task: [3] pid = 288
        …
[ 3082.139499] ***[MYSCHED] Pick new task: [100] pid = 288
```

Memory, Virtual Memory, Kernel Memory

- 메모리 관리 기본 개념
- 물리 메모리 및 메모리 주소 공간 관리
- 커널 메모리 관리, 프로세스 가상 메모리 관리
- 프로세스 가상 메모리 관리 및 페이지 회수 기법
- [실습] 페이지 컬러링

정진규 교수

(성균관대학교)

Linux Memory Management

Jinkyu Jeong (정진규)
jinkyu@skku.edu
http://csl.skku.edu/People/Jinkyu
Dept. Semiconductor Systems Engineering,
Sungkyunkwan University

Agenda

- Generals in VM management
- Linux VM management
  - Physical memory organization and handling
  - Virtual address space management
  - Mapping (hat layer) in paging
  - Process virtual memory management
  - Dynamic memory allocation
  - Demand paging & memory sharing
  - Swapping & LRU management
- Practice – Page Coloring

References

- http://lxr.free-electrons.com
- Professional Linux® Kernel Architecture, Wolfgang Mauerer (2.6), Wiley Publishing, Inc.
- Understanding the Linux® Virtual Memory Manager, Mel Gorman, PRENTICE HALL (2.6)
- Slab allocators in the Linux Kernel: SLAB, SLOB, SLUB, Christoph Lameter, (Revision Oct 3, 2014)
- Slides are modified from the original ones by Jung Guk Kim @ Hankuk University of Foreign Studies, in Linux Kernel Camp 2015

Memory Management Generals

Types of Memory Management Systems

- Non-virtual memory system without MMU (Memory Management Unit)
  - Physical address = Logical (linear) address
    - **Logical addresses** are generated by CPU for instruction fetch & operand access.
    - **Physical addresses** are real memory address.
  - Contiguous loading
    - The entire program should be loaded in a contiguous area in order to be executed.

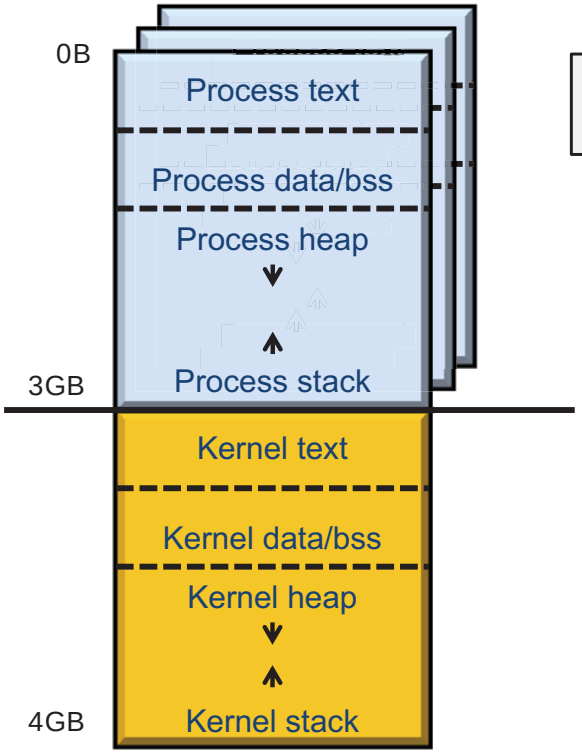
Types of Memory Management Systems

- Virtual memory system with MMU
  - Physical address ≠ Logical address
    - Mapping by **MMU** is necessary at each memory access of an instruction.
  - Scattered loading
    - A process is split into **pages** of equal length, loading unit: **page**
  - Partial loading
    - A process can be executed even if a small portion of a program has been loaded.
  - Demand paging
    - A page (pages) is loaded only when it is necessary during execution.

Paging

- Paging
  - The **logical address space** of a program is split into equal-sized pages.
  - Physical memory is also split into same sized page frames.
  - Scattered/partial/demand loading with pages
  - A pages can be loaded into any empty frame.
  - Most common in VM management.
- Segmentation & paging with segmentation
  - Segments: logical units with variable length. (function, text, stack, etc.)
  - Commonly used in early 16 bit machines. (AT)
  - Logical address space < physical space.
  - A program can have multiple segments to overcome the limitation of the virtual address space.

Process & Kernel Address Space (32bit)



Processes use the same overlapping virtual address spaces!

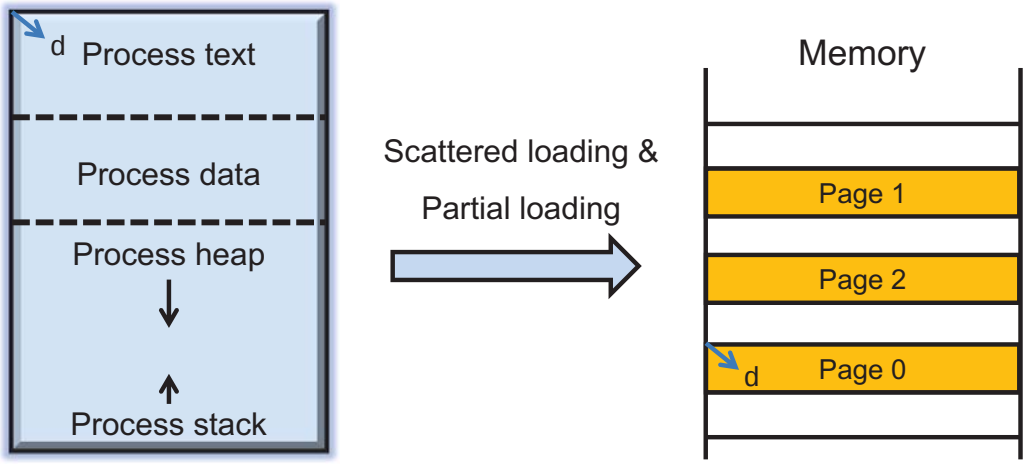
Text: instruction codes (read only)

Data/bss: global variables (initialized or not)

Heap: space for dynamic memory allocation

Stack: space for local variables, and function frames

Virtual Address vs. Physical Address



***Virtual address (Linear address)*:**

Relative address in the process address space, (p, d)

Address generated by CPU !

eg. &*i* is the virtual address!

→ mapping →

***Physical address*:**

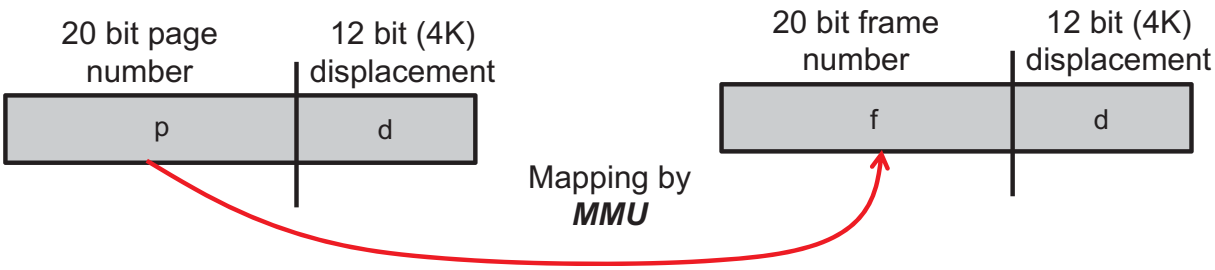
Address in memory (f,d)

MMU (Memory Management Unit)

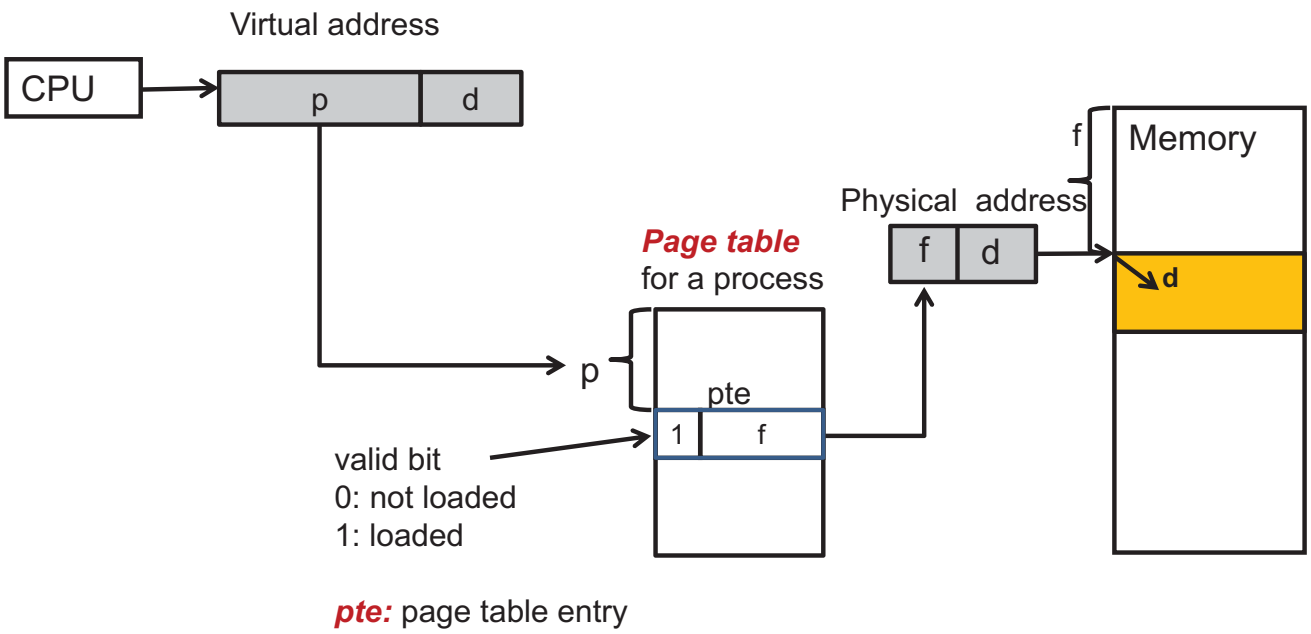
MMU cheats the CPU!: "***Execution time binding***"

Mapping in Paging (1-level mapping)

- Virtual address = (p, d) : Execution time binding is necessary.
  - p: page number
  - d: displacement(offset) in a page
- If the page has been loaded, the physical address will be (f, d).
  - f: frame number
  - d: displacement in a page is not changed because the loading unit is page.
- So it is sufficient to change p with f for mapping(MMU).
- Ex: 32 bit address, page size = 4KB (usually)

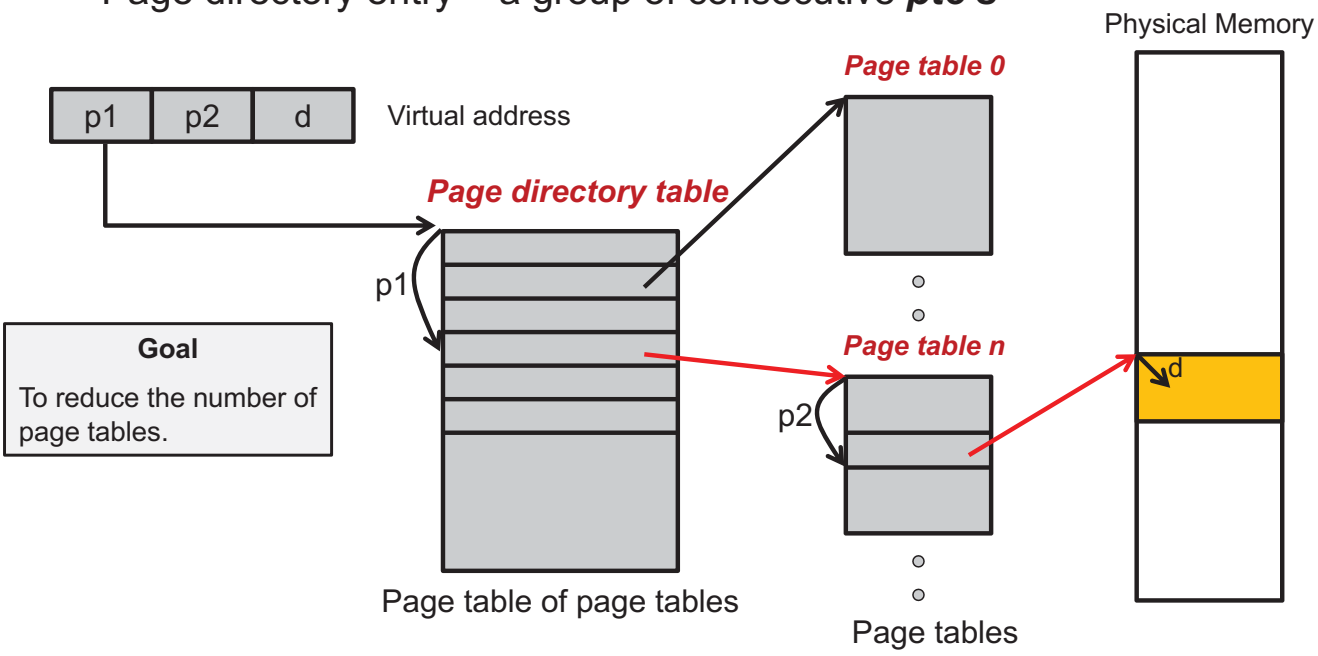


Mapping in Paging (1-level mapping)

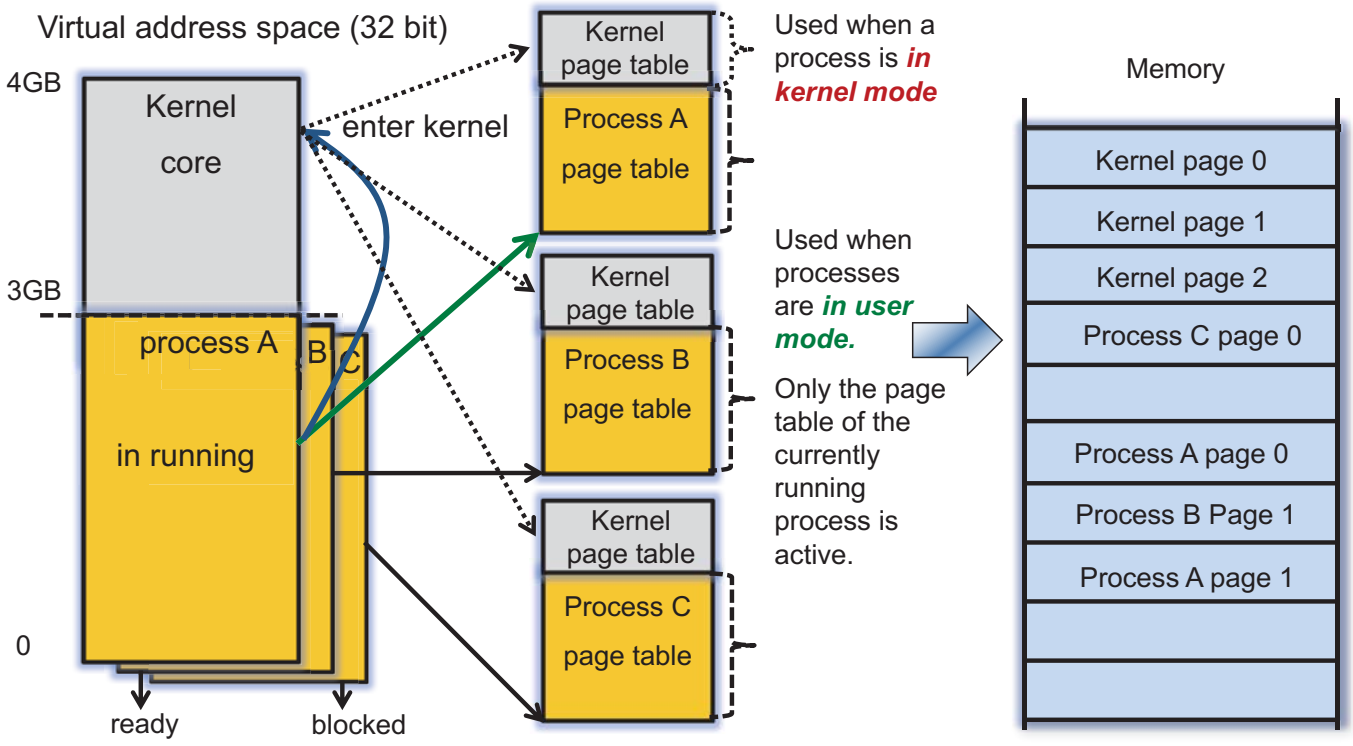


Two/Three/Four-level Mapping in x86

- A page number is split into a ***page directory number*** (p1) and a ***page number*** in the directory *(p2)*
- Page directory entry = a group of consecutive ***pte's***



VM & Multi-tasking



Where are page tables located?

- Commonly, page tables are located in the main memory.
- Where?
  - ***Kernel page tables*** are located at the specified kernel space. (boot time) : a kernel also runs in virtual memory mode.
  - ***User page tables*** are allocated dynamically at kernel space.
  - Because the kernel also runs in VM mode, the addresses of page tables for kernel accesses, are also virtual ones !
  - For mapping, MMU must know the physical address of page tables.
  - How can we solve this problem? (later)

More on Page table

- Mapping by MMU(Memory Management Unit) H/W
  - At least 2 mem. accesses are necessary
    - page table access, target memory access
  - 3(4) mem. accesses in two(three)-level mapping
  - To reduce page table access time, *x86* uses a H/W cache for page tables called ***TLB*** (Translation Look-aside Buffer) and an MMU cache
- PTBR(Page Table Base Register) : a process context that points to the page table (page directory table) of the currently running process.
  - PTBR switching must be included in context switching !
  - ***cr3*** in *x86* points to page table of current process.

Virtual vs. Physical Address Space

time ↓

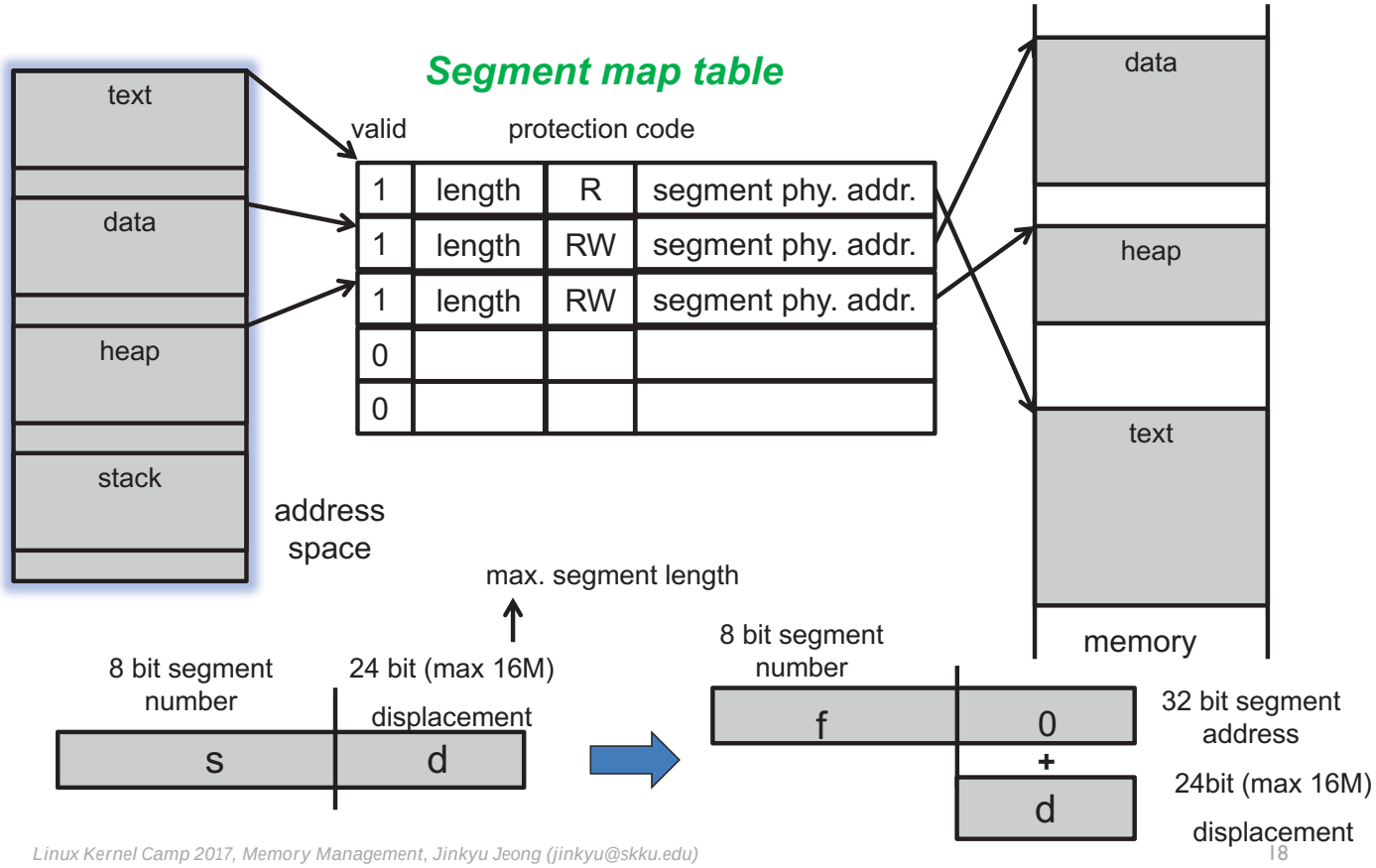
CPU address, Virtual space	***Memory bus/size***	***Solution***
16 bit (64K)	>= 16 bit	Segmentation
32 bit (4GB)	Under 4GB	Paging
32 bit (4GB)	>= 32 bit	Paging, Huge paging Each user: 3GB-limit, but many users
64 bit ()	2^{64} >> mem. size	Paging, Huge paging

Segmentation

- Segmentation
  - [Def] Segment
    - A variable-sized logical unit of a program
      - Text, text, data, stack, functions, etc.
    - Loading & swap-out unit
  - Mapping with a ***segment table*** instead of a page table
    - ***(s,d) -> (f,d)***
  - Segment table entry
    - valid, length, protection code, physical address of a segment
  - Sharing is easy but memory allocation is difficult (fragmentation).

- Segmentation is rarely used !

Mapping in Segmentation



Segmentation with Paging

- Paging + Segmentation
  - Loading unit: page
  - Mapping: segment table + page table
  - ***(s,p,d) -> (f,d)***

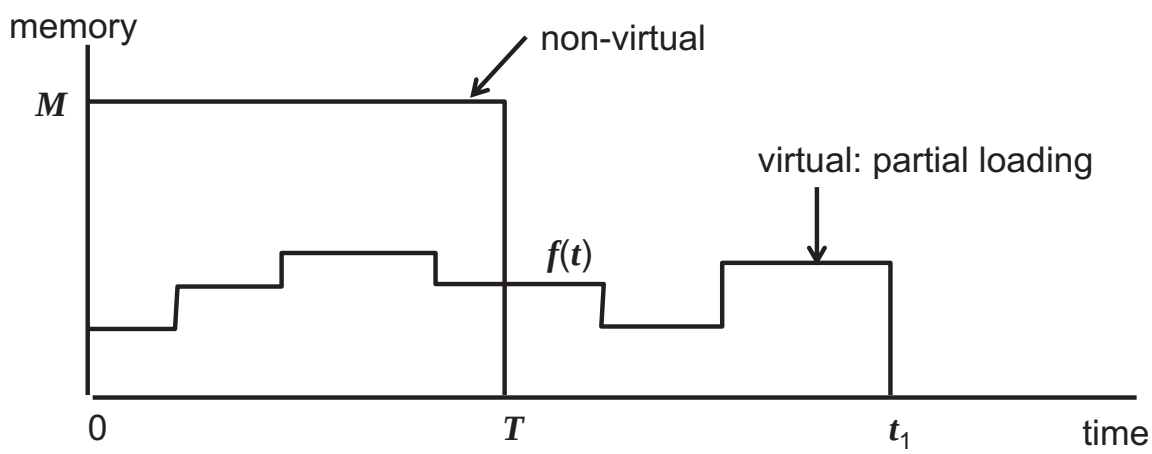
Paging

- Logical(Virtual) addr. space (264) >> physical memory size O(GB)
- Size of a program cannot be larger than the logical address space, however it can be larger that the size of physical memory.
- Sum of user spaces may be larger than the size of the memory.
- A program that is larger than the physical memory size can be executed by partial loading. (S/W caching of pages)
  - **Demand paging**: fetch a page from disk to memory when the page is referenced.
  - **Swapping**: rarely-referenced (data, heap, stack) pages in memory will be swapped out to the swap space in a disk to increase the size of free memory.

Motivations

- Why partial loading?
  - In a program, there is a core that is mainly executed. It's enough to load only the core for execution.
    - ex1) Error processing parts are rarely executed!
    - ex2) A large sparse matrix: only a portion of the matrix is used.
  - Size of a program may be much larger than the size of physical memory.
    - Large matrices, lists, tables
  - Decreased load time.
- So the partial loading increases the size of free memory and this allows more processes in a system. (an increased degree of multi-programming)

Total Memory Usage (Memory Efficiency)



$$\int_0^{t_1} f(t)\,dt \;<=\; M \cdot T \quad ; \quad M = \text{program size}$$

T = execution time in non-virtual

VM is more efficient in total memory usage !

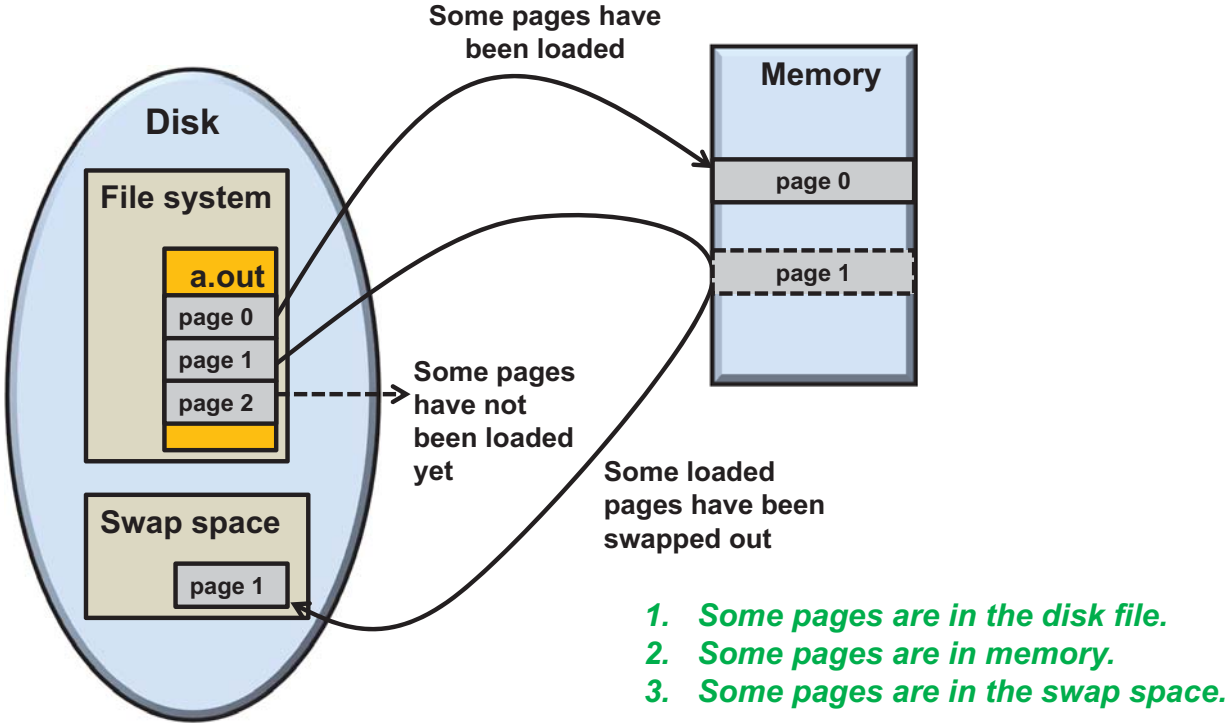
Paging Policies

- Policies for a paging algorithm
  1. Fetch policy
     - When and which pages are loaded?
  2. Placement policy
     - Where a loaded page should be located in memory?
     - In VM paging, a loaded page can be located at any empty frame. (no problem, a placement policy is not necessary.)
     - Linux has a placement policy, (Movable, Reclaimable, Unmovable) for zone compaction to reduce fragmentations.
  3. Replacement policy (swap-out policy)
     - Which pages must be swapped out (saved) to disk when free-memory is insufficient?

Locations of a Process in a Paging System

- File-backed pages
  - Memory-mapped file pages
    - Text pages
    - Clean Data pages
- Swap-backed pages (swapped out or not)
  - Anonymous (dirty-data / bss / heap / stack) pages
  - e.g) A new dirty data (becomes an anon page) that has not been swapped out yet, is also a swap-backed page.
- Others
  - Zero pages to be shared with clean heap/bss/stack pages.

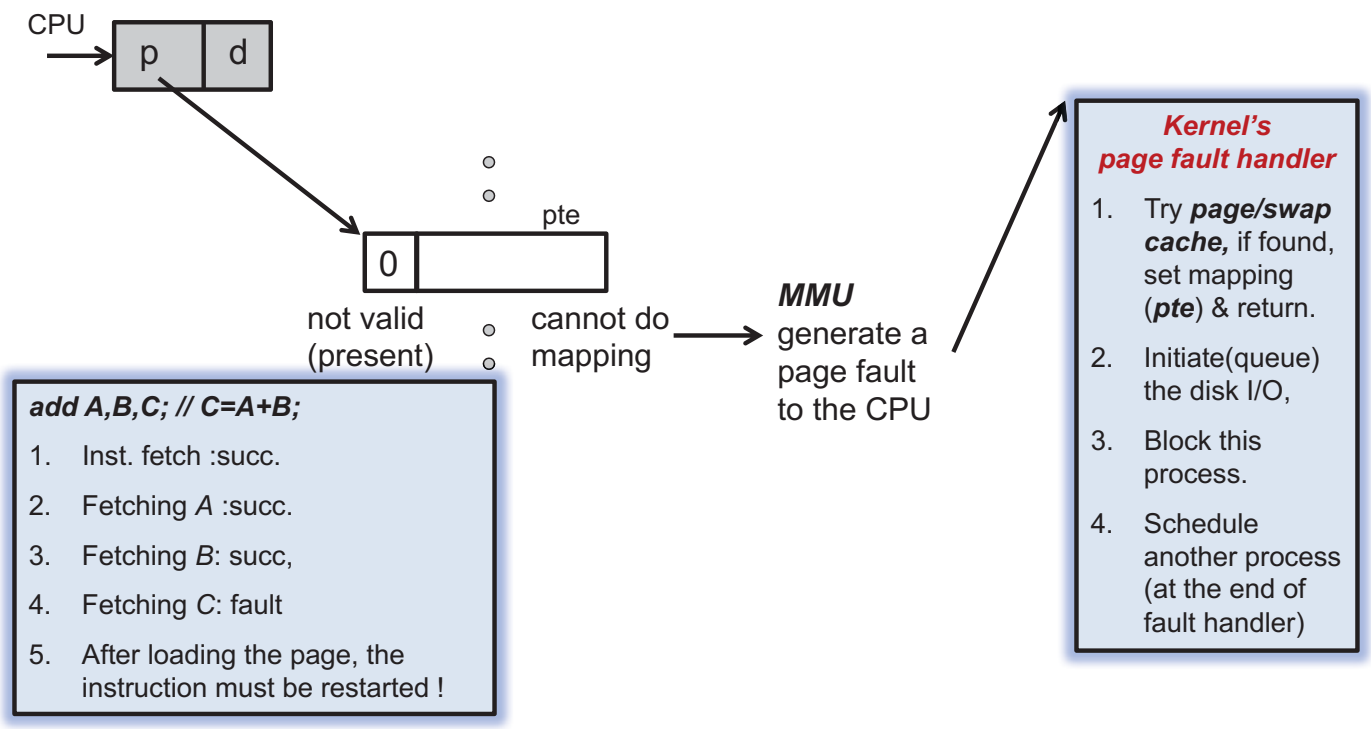
Locations of a Process in a Paging System



A Fetch Policy: Demand Paging

- Demand paging
  - A page is loaded when the **page fault** for the page occurs.
- Page fault
  - When the **MMU** tries to map a logical address to a physical address, it must refer the corresponding **pte** (page table entry) of the page table.
  - But if the **present (valid)** bit of the pte is **not set** or **no-pte** (not loaded yet), **page fault occurs** by MMU H/W.
  - Just like an interrupt handling, **page fault handler** is executed immediately for the current process.
  - **Page fault handler** initiates the loading (disk I/O) of the page and CPU control goes to another process. (Before the disk I/O, try S/W caches (page cache, swap cache) first to reduce disk I/Os.
  - After the page loading is finished, the old process cam restart.

A Fetch Policy: Demand Paging



Demand Paging Performance

- Demand Paging
  - Demand paging is common because it's simple and optimal.
  - However, demand paging needs a good page replacement policy.
    - A good page replacement policy.
      - ***Should not swap out a page which will be used in the near future!***

- Prepaging (Anticipatory paging)
  - To reduce the page fault processing time, fetch some pages by ***guessing*** before page faults.
  - Linux: when the kernel fetches a page as a result of page fault, it fetches the following pages together. This called ***"read-ahead"***
  - ***So sequential access is preferred!***

Page Fault Processing

- Page Fault Processing
  - A page reference occurred (instruction execution).
  - Page table mapping (not **valid**) → page fault (trap to kernel), → page fault handler.
  - Try to find a page at **page/swap cache**. If (found) map **pte** to the page.
  - If (not found), fetch the page from the **file/swap-device**. During the disk I/O, the kernel schedules another process.
  - After the disk I/O is finished,
    - Update the page table entry (set valid bit and frame number), make the process to be ready.
    - When scheduled, restart from the instruction.
- To save physical space, **memory sharing** is fully utilized.

Average Memory Access Time in VM

- Average memory access time in VM
  - $m_{vm} = M * (1 - p) + F * p;$ (1)
  - where, M is memory access time,
    F is the page fault processing time,
    p is the page fault rate.
  - In (1), the F (page-fault-processing-time) is a dominant factor because it may include disk I/O's.
  - So to reduce m_{vm} and the number of disk I/Os, the p (page fault rate) must be reduced by ***a good page replacement policy.***
  - Also a ***prefetching*** is supported to reduce disk I/O time.

Page Replacement Policy

- Page replacement in UNIX/Linux (actual)
  - Actually, pages are swapped out when the size of free memory becomes smaller than a specified ***watermark***.
  - This is to preserve a specific size of free memory.
  - This is done by a background ***kernel thread "kswpad"*** (async. reclaim),
  - or directly done by sync. page-reclaim (***try_to_free_pages()***) when doing the ***alloc_pages()*** kernel function.
  - When choosing ***victim pages***, an ***LRU approximation algorithm*** is used.

Page Replacement Algorithms

- Goal
  - Minimizing the ***page fault rate***.

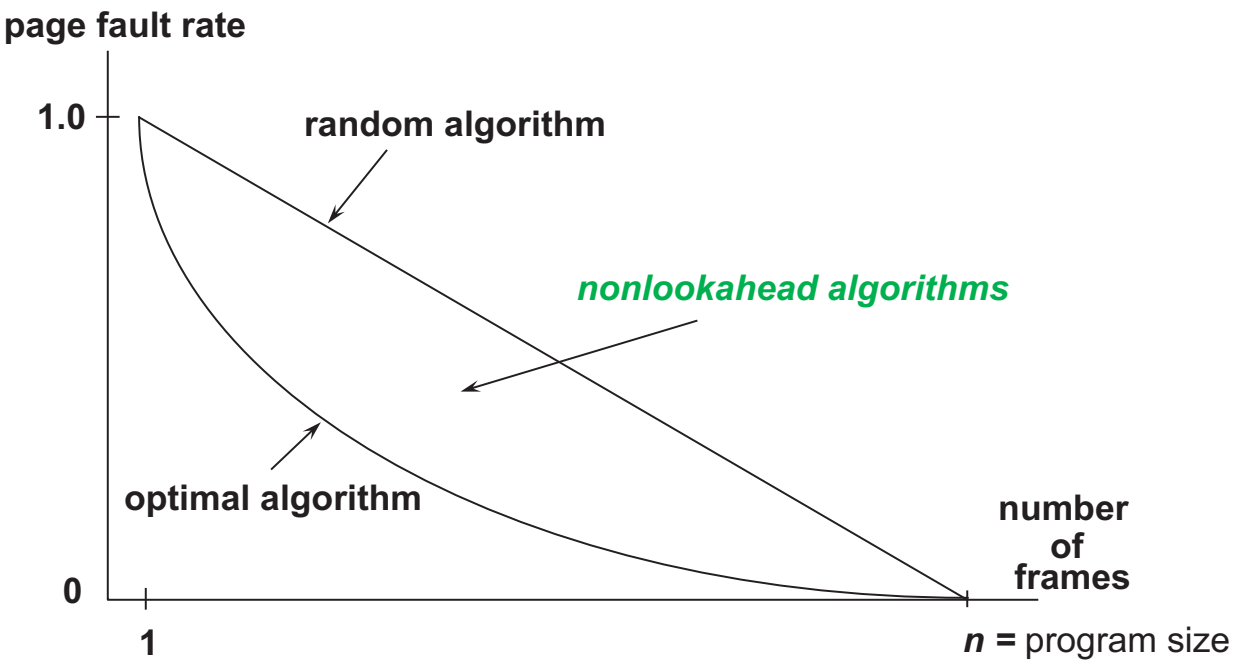
1. Belady's optimal algorithm (Principle)
   - Swap out the page that will be referenced most lately.
   - Ideal but not possible because we don't know the future.
2. FIFO
   - Swap out the page that has been loaded earliest.
   - FIFO anomaly => increased number of frames, however, but more page faults can arise at some time.
   - This alg. is intuitively no good from the fact that "an early-loaded page still remains in memory " means that it is a frequently used page.

Page Replacement Algorithms

3. LRU (Least Recently Used)
   - Swap out the memory-resident page that has been referenced least recently. (It has a big implementation overhead)
   - Look at the past instead of the future!
   - A most common background of all caching schemes. (H/W & S/W cache)
4. LRU approximation algorithms
   - BSD 4.3's LRU approximation
   - SECOND CHANCE/Clock Algorithm : LRU + Working-set alg. (old SysV)
   - Linux's scheme : Clock + 2-Q + Working-set alg.

- Trends: (a less optimal but a fast LRU alg. to reduce the overhead)

Replacement Algorithms



performance of page replacement algorithms

Belady's Optimal Principle

- Reference string : $\omega = r_1, r_2, r_3, \ldots, r_t, r_{t+1}, \ldots$
- m = # of frames in memory
- $S_n = \{ \ldots, p_i, \ldots \}$: set of pages in memory after r_n
- Forward distance of p_i at r_t is
  - The distance of the 1st occurrence of pi in the string "$r_{t+1}, r_{t+2}, \ldots$" from r_t

  - Ex) ω = 1, 1, 2, 3, 3, 4, 3, 2, 1
  - Current reference: rt = 4, memory = {1, 2, 3}
  - Forward distance of page 1 = 3,
  - Forward distance of page 2 = 2,
  - Forward distance of page 3 = 1.
  - This means that page 1 will be used most lately!
  - So if swap-out the page 1 => page fault will occur most lately.

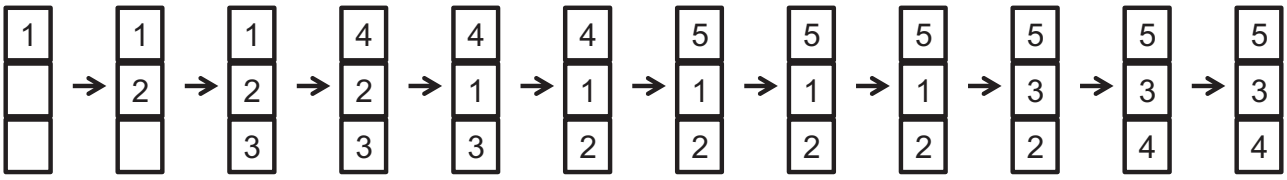
FIFO Algorithm

- Replace the page that is loaded earliest among memory-resident pages.
- This algorithm suffers from the Belay's anomaly
- Belady's anomaly (FIFO anomaly)
  - For a given reference string, the number of page faults must be reduced if the size of physical memory is extended.
  - But when the FIFO algorithm is used, there is a case that the number of page faults increases even though memory has been enlarged.

- Intuitively, FIFO algorithm is not good.
  - The fact that an early-loaded page is still in memory, means that this page has been used frequently.

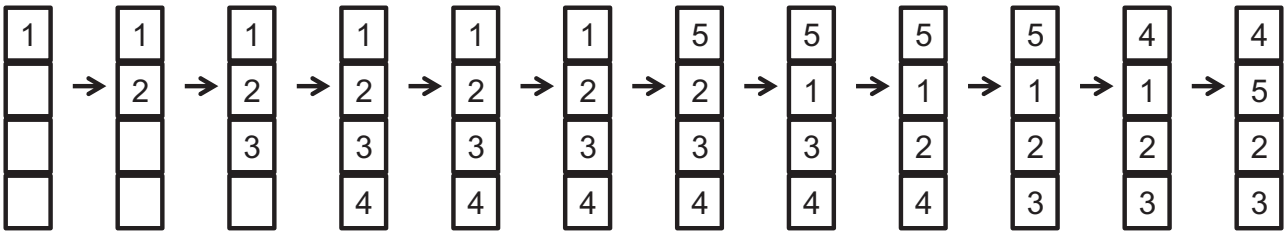
FIFO Alg. : Belady's Anomaly

- *Reference string* ω = 1* 2* 3* 4* 1* 2* 5* 1 2 3* 4* 5
- 3 memory frames:



9 page faults occurred (*)

- Same reference string ω = 1* 2* 3* 4* 1 2 5* 1* 2* 3* 4* 5*
- 4 memory frames: => 10 page faults occurred !



LRU (Least Recently Used)

- Among memory-resident pages, choose the page that has been referenced least recently as a victim.
- Backward distance of pi at rt is
  - The distance of the 1st occurrence of pi in the string "rt-1, rt-2, …" from rt
  - Ex) ω = 1, 1, 2, 3, 3, 4, 3, 2, 1
  - Current reference rt = 4, memory = {1, 2, 3}
  - Backward distance of page 1 = 4,
  - Backward distance of page 2 = 3,
  - Backward distance of page 3 = 1.
  - This means the page 1 has been referenced least recently.
- Remove the page which has the max. backward distance from S_{t-1} when r_t occurs. So the victim page is page 1.

LRU (Least Recently Used)

- The idea of LRU algorithm is based on that a page that is not used recently, is supposed not to be used in the near future.
- LRU is the best and popular algorithm, however,
  - It is hard to implement the LRU, because of some overheads;
    - Overhead 1: to maintain the past reference-list for every memory references.
    - Overhead 2: searching of the LRU page in the list.
- So, most of other practical algorithms are LRU-approximations.

Major Fields of a pte

pte (page table entry)

v: valid(***PRESENT***) bit

ref: reference bit (***ACCESSED*** in Linux)

mod: ***dirty***(modify) bit

prot: protection code (R, RW, EX)

frame/swap-entry: frame # if v is 1
swap-entry (swap-info) , if v is 0 & swapped out

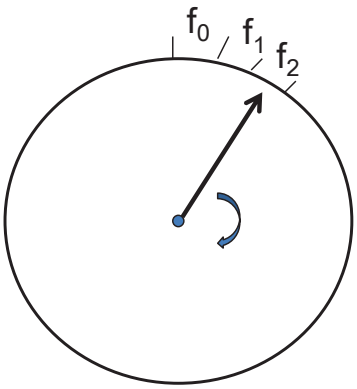
LRU Approximation in BSD 4.3 UNIX

- BSD 4.3 UNIX page replacement algorithm
  - A ***pte*** has ***ref., mod.***, and ***age*** fields
  - Timer interrupt handler clears all reference bits of all ***pte's*** periodically (***Δt***). When clear,
    - If ***ref.*** bit is 0, then ***age***++; // increases not-used time
    - If ***ref.*** bit is 1 then ***age*** = 0; and ref. bit = 0; (new-born child)
  - When swapping, the victim page will be the eldest page (max. age)
  - If ***Δt*** is larger, it becomes a rough approximation.
  - This algorithm has also an overhead in scanning ***pte's*** & choosing the eldest page, ***but closest to the LRU.***

Second Chance Alg. (Clock Alg.): SysV UNIX

❖ ***Second Chance Alg. (Clock Alg.): SysV UNIX***

Mem. resident pages



- Only when mem. lacks of free pages (# of free pages < watermark),
  - Scan the frames, (clockwise)
  - When clears the ref. bit
    - 0 : swapout this page
    - 1 → 0 : skip this page (give the 2nd chance to this page)
- Repeats until enough free frames

- ***Swap out any page that is not used during one turn of the clock hand.***
- ***When free memory is insufficient, the clock-hand turns fast, vice versa.***

Working-set Model

- A program has a locality of reference!
  - Localities of reference
    - ***Temporal locality***
      - If a page has been referenced now, it is highly probable that the page will be referenced in the near future.
        - » Ex) loop, sequential executions in a page
    - ***Spatial locality***
      - If a page has been referenced now, it is highly probable that the neighbor pages will be referenced in the near future.
        - » Ex) loop, sequential executions
  - Why?
    - Because we do ***structured programming.***
      - sequential, selection(if), iteration(loop) only
      - A goto-less program is preferred.

Working-set Model

- [Def] ***Working set***
  - $W_{\Delta t}(t)$: Set of pages referenced in $[t - \Delta t, t]$
  - Δt is called working set window.
- Working-set transfer
  - Working set is changed as the execution continues.
- Due to the locality of reference,
  - The number of page faults increases rapidly if some of the current working-set pages are not in memory.

Working-set Principle in Page Replacement

- ***A working-set page should not be a victim !***
  - A good page replacement algorithm is necessary to keep working-set pages in memory. (A working-set page should not be a victim!)
  - Also, an enough memory space must be given to a process for keeping its working set.
- Working set examples
  - ***Second chance algorithm***: referenced pages during one turn of the clock hand.
- Thrashing
  - When working-set pages are not reserved in memory => many disk I/O's => no progress because of huge swap-in & swap-outs.
    - Caused by shortage of the physical memory or poor replacement policies.

Other Considerations

- Page size
  - A smaller page size is better to construct a precise working set in demand paging scheme.
  - So usually page size is set as the size of a physical disk block.
  - Early system: 512 bytes
  - Today's system: usually 4KB,
  - Huge pages (2/4 MB) to increase TLB reach
  - Kernel: 4KB, 2MB, 4MB, or 1GB (h/w dependent)

Linux VM Management (~4.x)

Linux VM Overview (from 2.6 ~4.x)

- **Paging system**
  - Demand paging & pre-paging (file/swap read-ahead).
  - Scattered loading & partial loading.
  - H/W segmentation is not used actually.
- **Support 32 bit ~ 64 bit machine (mapping)**
  - Physically 2-level, 3-level, 4-level page tables (H/W dependent)
  - Linux 4-level page table (H/W independent)
  - TLB (page table cache) for fast address translation
  - MMU cache for fast address translation

Linux VM Overview (from 2.6 ~4.x)

- **Pageout (page replacement policy)**
  - LRU (best, but large overhead) → LRU approximation algs. (old BSD) → a second chance (clock alg.) (old SysV) → **2Q LRU + working set model** (Linux)
  - To reduce the number of page faults (major page faults = disk I/O)
- Memory sharing
  - **CoW**(Copy on Write) for **forked** processes & **anonymous** pages.
  - Address space sharing of **threads**.
  - Shared memory, memory-mapped files.
  - Shared library functions (dynamic binding).

Linux VM Overview (from 2.6 ~4.x)

- **Caches to reduce disk I/O**
  - Caches for pages
    - **Page cache** (pages fetched from files: text, early data)
    - **Swap cache** (pages swapped in/out from/to a swap device: dirty data, anonymous pages (bss, heap, stack))
    - Buffer cache or block cache (pages for low-level block device (/dev/sda1): FS metadata, …)
  - Other S/W caches
    - inode cache, dentry cache, slab cache, …

Linux VM Overview (from 2.6 ~4.x)

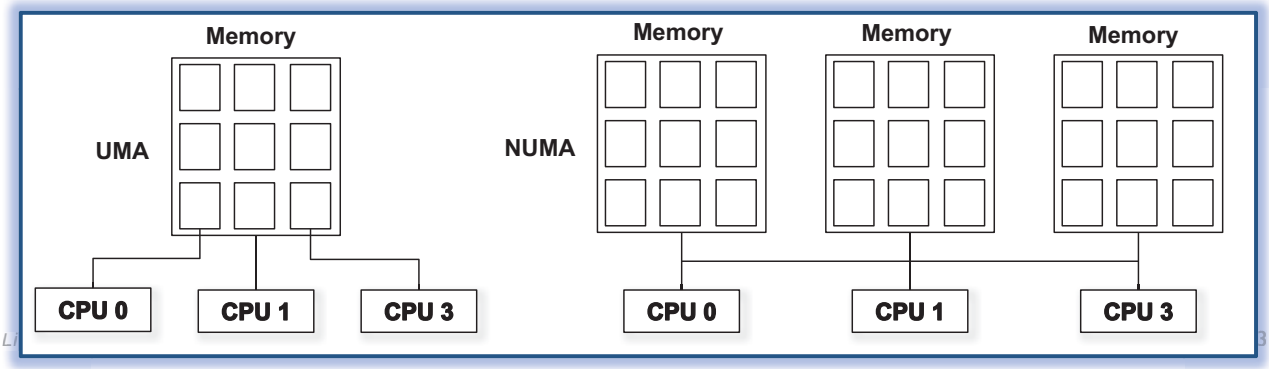
- **SMP (Symmetric Multi-Processor)**
  - Per CPU S/W **hot** and **cold** caches of 20 sized free pages.
  - Per CPU **slab** cache to enhance the H/W cache hit-ratio of local caches.
- **cgroup (mem_cgroup) support:** control RSS & page cache
  - A cgroup (task group) has limitations on CPU & memory usages.
- **MMU-less Architecture**
  - Comes from "µcLinux"

Linux VM Overview (from 2.6 ~4.x)

- **Reverse Mapping**
  - From **struct page**, find the **ptes** (sharing).
  - struct page has a reverse map to ptes **to look access/dirty bits of ptes**, & **to modify/set flags of pte's**.
  - Useful when **pageout** is frequent, otherwise it's an overhead.

Linux VM Overview (from 2.6 ~4.x)

- **NUMA(Non-Uniform Memory Access) & UMA support**
  - *NUMA* machines (non-uniform memory access) are always multiprocessor machines. Local RAM is available to each CPU of the system to support particularly fast access. The processors are linked via a bus to support access to the local RAM of other CPUs
  - Access costs are different from each other. (via fetch H/W or network)
  - Node list : *pgdat_list* (linked list of ***pg_data_t*** per node)
  - For ***UMA*** architecture (eg. PC), one static ***pg_data_t*** exists.
  - Node
    - A node is divided into a number of *zones*.
    - A zone: a range within memory (described by ***struct zone_struct (zone_t))***



Linux VM Overview (from 2.6 ~4.x)

- **KSM (Kernel Same-Page Merging) from 2.6.32 (CONFIG option)**
  - Lets the hypervisor system share identical memory pages amongst different processes or virtualized guests. KVM (Kernel-based Virtual Machine: Hypervisor = kernel module) can use KSM to merge pages of virtual machines. Hypervisor helps VMs to share single host H/Ws)
  - Can be used for both of VMs and general processes
  - This is done by scanning through the memory finding duplicate pages. The duplicate pair is then merged into a single page, and mapped into both original locations. The page is also marked as "copy-on-write", so the kernel will automatically separate them again should one process modify its data.
  - Red Hat found that 52 virtual instances of Windows XP with 1GB of memory, could run on a host computer that had only 16 GB of RAM.
  - Concerns: CPU usage ↑, Security risks 2.6.32 ↑

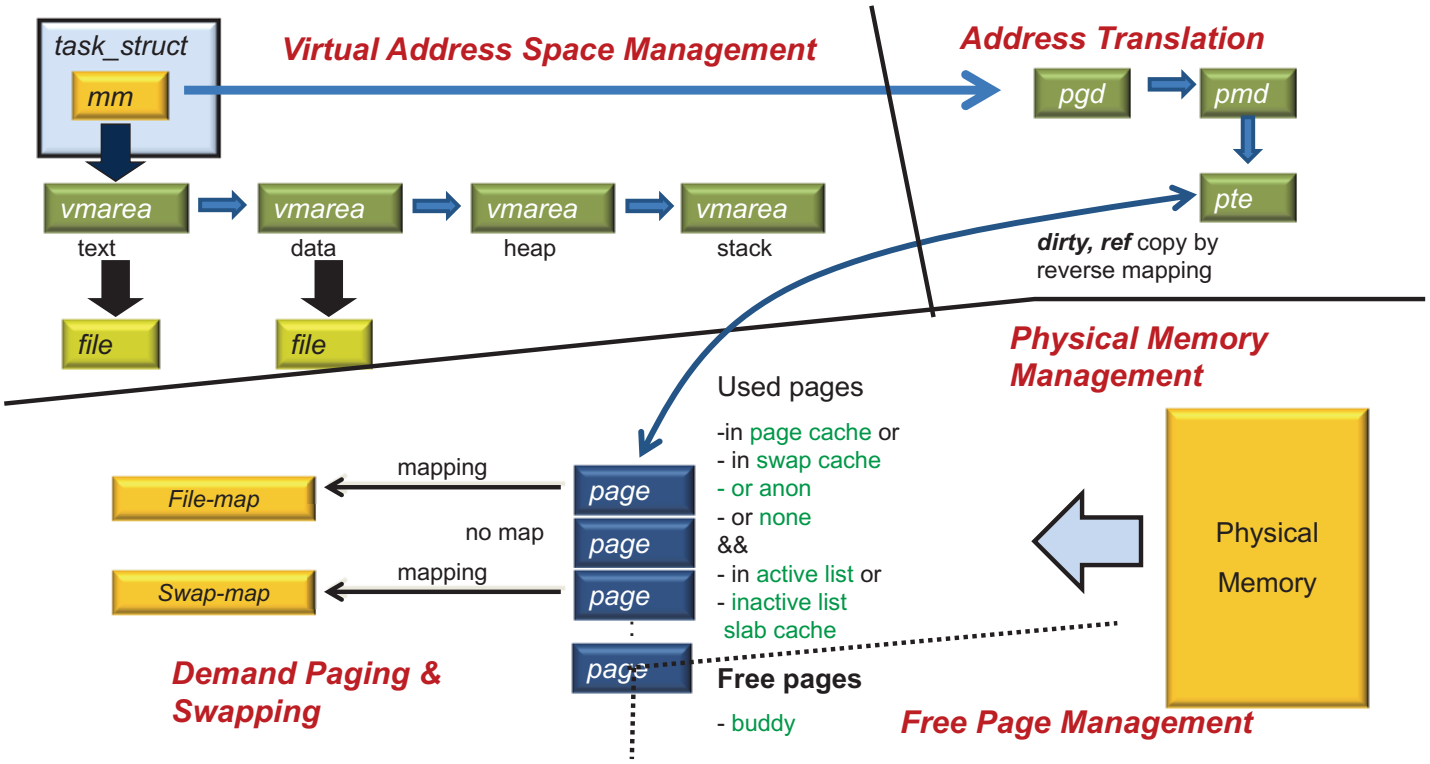
Linux VM Overview (from 2.6 ~4.x)

- **CMA (CONFIG option)**
  - For special I/O devices that needs bulky & physically contiguous pages.
  - Buffers for them is allocates in a zone and they are treated as used pages. (when page reclaim, they are excluded in counting nr_free_pages)
- **Memory hot plug-in (CONFIG option)**
  - Memory plug-in at run time
  - Scan and add
  - Launch a new kswapd;
- **Huge page**
  - ~ n MB page to enhance TLB performance. (x86: 2MB)

Linux VM Overview (from 2.6 ~4.x)

- Scope of this lecture: Main logic of VM
  - UMA.
  - MMU machine (VM).
  - mem_cgroup/ksm disabled: (left to readers).
- Goals
  - Understanding the main logics of VM.
  - To help further source navigation.
  - How? -> Why? -> Problems? -> You can enhance the core !

Linux VM Management Subsystems



Linux VM Subsystems

1. Virtual address sapce management
   - struct mm_struct (mm) in a task_struct (PCB) has virtual address space descriptions and a pointer to the page global directory (pgd)
   - Each struct vm_area_struct (vma) for text, data, bss, heap, stack, and mmapped regions, contains region-description. (file (executable) pointer, size, prot, start & end address, etc.)
2. Hat (H/W Address Translation) Layer
   - Handling of page global directory (PGD), page upper directory (PUD), page middle directory (PMD), and page table entries (PTEs).
   - Mapping by MMU

Linux VM Subsystems

3. Demand-paging / Swapping
   - Page fault handling
     - Fetch pages from file or swap devices or just allocate a zero page.
   - Swapping (Page Reclaim)
     - Reclaim free pages (LRU-approximation)
     - Swapping out anonymous pages, discarding or writing-back cache pages
4. Physical-memory Management
   - Physical page (frame) management by the array of struct page. (each contains a description of a physical frame.)
   - Dynamic memory allocation for the kernel.
   - Free page management.
   - Cooperation with the demand paging & swapping sub-systems.

Physical Memory Construction

Physical Memory Construction (32, 64 bit)

1. ***ZONE_DMA***: lower physical memory ranges for old (ISA) DMA devices. (x86: first 16MB)
2. ***ZONE_DMA32***: upper physical memory ranges for DMA devices supporting only 32bit physical address. (up to 4GB)
3. ***ZONE_NORMAL***: directly mapped into the upper region of the linear address space (x86_32: up to 896 MB), mapping = just shift, kernel's page table at a fixed location can map physical spaces up to ZONE_NORMAL.
   - If all DMA devices support this area, ZONE_DMA is empty.
   - Most of kernel memory are allocated from *ZONE_NORMAL*.
   - Usual page allocation order for user pages: ***ZONE_HIGHMEM -> ZONE_NORMAL -> ZONE_DMA***

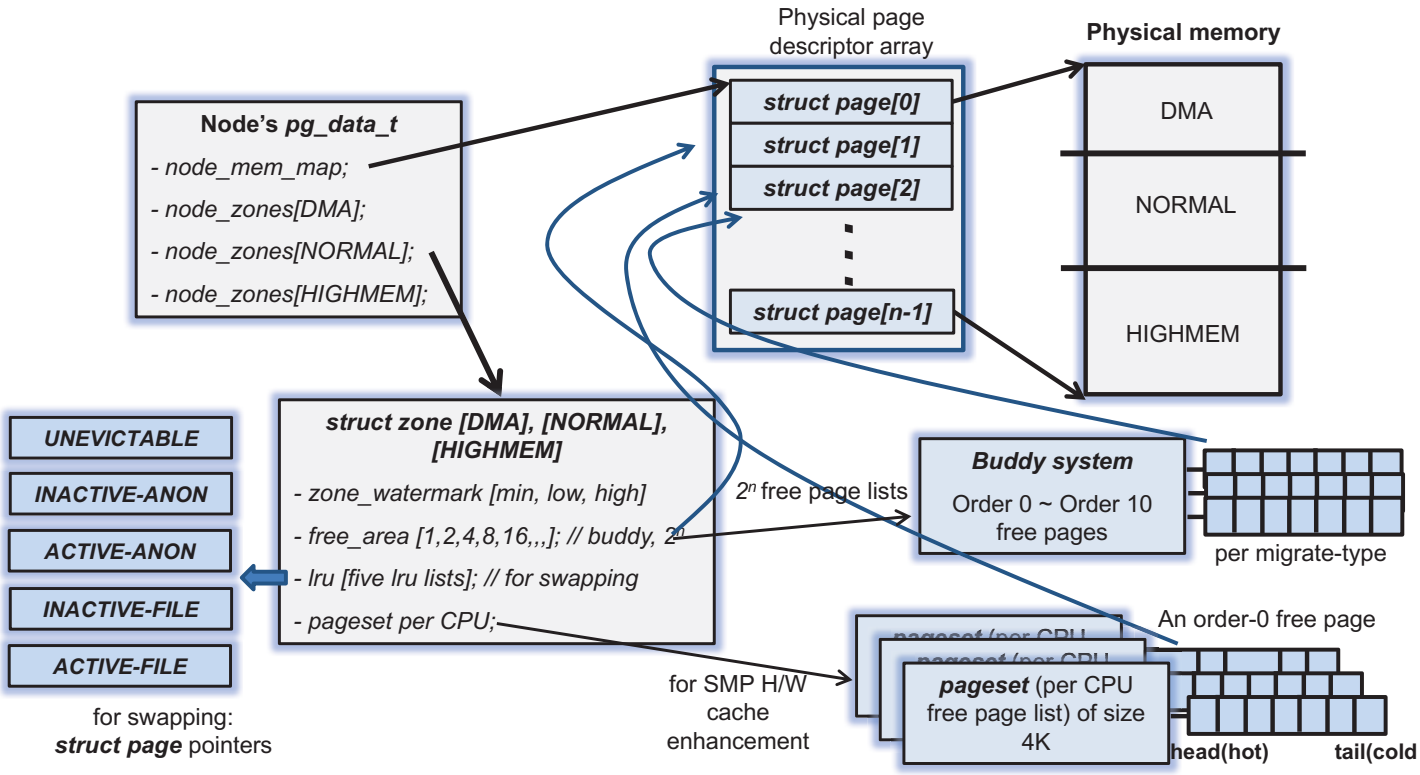
Lowmem: ZONE_DMA + ZONE_DMA32 + ZONE_NORMAL

Physical Memory Construction (32, 64 bit)

4. ***ZONE_HIGHMEM***: not mapped directly by kernel (x86_32: 896MB~)
   - For user pages, cache pages, and kernel dynamic pages (page tables, vmalloc pages)
   - Kernel cannot access this area because the kernel has no mappings for this zone. In this case, kernel maps a highmem page to kernel address space (**kmap area**). The mapping must be released after use as soon as possible.
   - However, user spaces can cover this zone with their virtual address mappings.
   - If a disk I/O to a highmem page is necessary, use a page in another zone and copied. (called "bounce buffer")
5. **ZONE_MOVABLE**: a pseudo (virtual) zone of movable pages for compaction.
   - Usually, a portion of ***HIGH_MEM*** is used. (optional)

	UMA		NUMA (64bit)	
	32bit	64bit	Node 0	Node 1~
DMA	o	o	o	
DMA32		o	o	
NORMAL	o	o	o	o
HIGHMEM	o	o		

Physical Memory Management



Node Memory Descriptor: *pg_data_t*

```
// On NUMA machines, each NUMA node would have a pg_data_t to describe it's memory layout.

typedef struct pglist_data {  // per NUMA node
        struct zone node_zones [MAX_NR_ZONES];  //DMA, DMA32, NORMAL, HIGHMEM, (MOVABLE)
        struct zonelist node_zonelists [MAX_ZONELISTS]; // fallback order of zones when page-alloc.
        int nr_zones;                           // number of zones
        struct page *node_mem_map;      // physical page (struct page array) pointer
         …
        unsigned long node_start_pfn;           // start physical frame number (= address)
        unsigned long node_present_pages;   // total number of physical pages
        unsigned long node_spanned_pages;  // total size of physical page range, including holes
        int node_id;
        wait_queue_head_t kswapd_wait;              // swapper (kswapd) thread sleep queue
        wait_queue_head_t pfmemalloc_wait;       // task queue (waiting alloc_pages())
        task_struct *kswapd;                    // kswapd task_struct pointer
        int kswapd_max_order;                   // explained later
        enum zone_type classzone_idx;           // explained later
         ::
} pg_data_t;
```

Zone data structure

```
struct zone {
  unsigned long watermark [NR_WMARK]; // for WMARK_MIN, WMARK_LOW, WMARK_HIGH,
                                      // 3 watermarks for the number of free pages
  long lowmem_reserve[MAX_NR_ZONES];
                                      // minimum number pages that must be reserved to avoid OOM,
      …
  unsigned int inactive_ratio;        // The target ratio of ACTIVE_ANON to INACTIVE_ANON
                                      // pages on this zone's LRU.
  struct pglist_data      *zone_pgdat;   // back pointer
  struct per_cpu_pageset  *pageset [nr_cpu];  // per CPU hot & cold cache in SMP (size 1 free pages)
  unsigned long    dirty_balance_reserve;  // per-zone reserve of non-dirtiable pages
      …
  unsigned long    *pageblock_flags;  // explained later used at zone compaction
          …
```


Zone data structure

```
  unsigned long   zone_start_pfn;  // zone start address
  unsigned long   managed_pages;  // buddy pages (reserved pages are excluded)
  unsigned long   spanned_pages;  // total pages (including hole(padding))
  unsigned long   present_pages;  // total pages – hole pages
  const char      *name;
      ::
  wait_queue_head_t  *wait_table;  //  task waiting hash queue at wait_on_page_locked()
   ::
  struct free_area       free_area [MAX_ORDER]; // free area lists of different sizes (buddy), 2^1~n
   ::
  struct lruvec    lruvec;  // lru array, and zone_reclaim_stat , five lru lists for reclaiming
  atomic_long_t inactive_age;  // explained later in demand paging & swapping
  unsigned long   percpu_drift_mark;
                  // desired number of free pages per CPU cache -> lower -> action
                                  // fill from buddy
  atomic_long_t vm_stat[NR_VM_ZONE_STAT_ITEMS]; // zone statistics
}
```

Several Zone State Flags

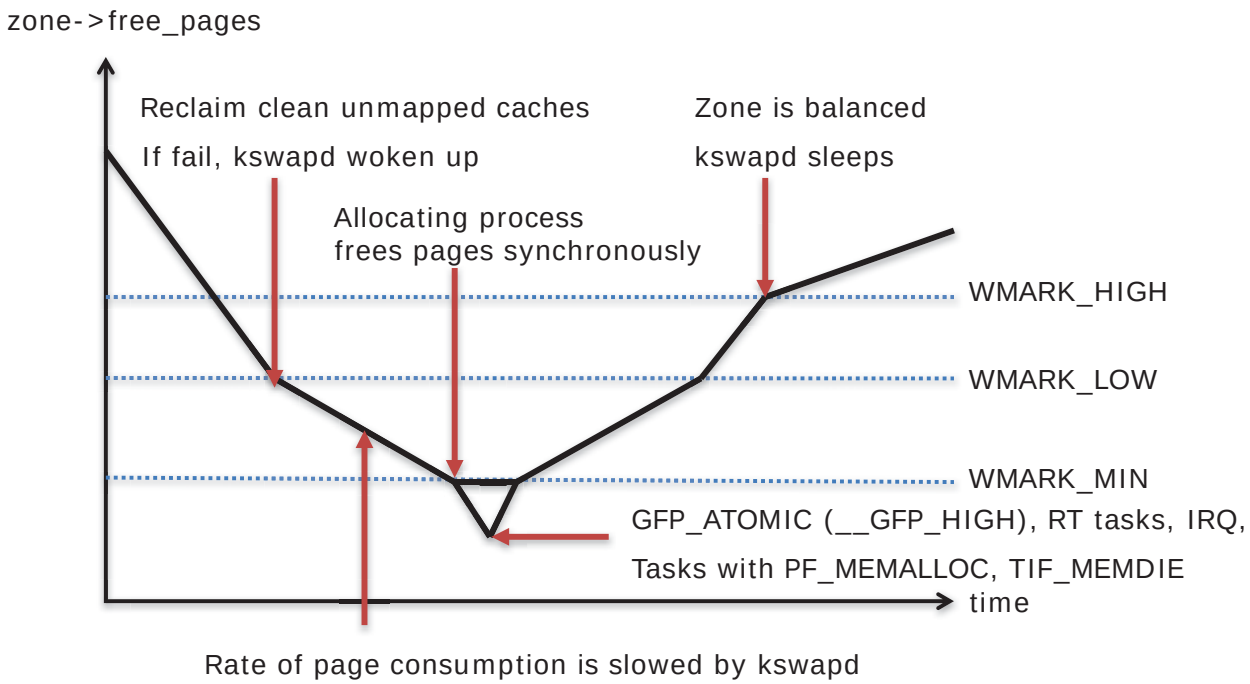
```
enum zone_flags {
 ZONE_RECLAIM_LOCKED,          // prevents concurrent reclaim
 ZONE_OOM_LOCKED,          // zone is in OOM killer zonelist
 ZONE_CONGESTED,          // zone has many dirty pages backed by a congested BDI
 ZONE_DIRTY,     // reclaim scanning has recently found many dirty file pages at the tail of the LRU.
 ZONE_WRITEBACK, // reclaim scanning has recently found many pages under writeback
 ZONE_FAIR_DEPLETED,    // zone fairness depleted
};
```


Zone Watermarks: Page Reclaim & Zone Balancing

Watermark level	***Lowmem (DMA, Normal)***	***Highmem***
WMARK_MIN	***min_free_kbytes*** * *zone_pages* / lowmem_pages	zone_pages / 1024 (32~128 pages)
WMARK_LOW	1.25 * min	*1.25 * min*
WMARK_HIGH:	1.5 * *min*	*1.5 * min*

- When alloc_pages() fails to allocate pages, do synchronous reclaim
  - **zone_reclaim()**: reclaim clean caches (clean unmapped file pages, clean swap caches)
  - **try_to_free_pages()**: reclaim pages, (writing out dirty pages, swapping out anon pages) if free pages is below **WMARK_MIN**
- kswapd perform asynchronous reclaim when # of free pages is below WMARK_LOW
  - Until, # of free pages is over ***WAMRK_HIGH***.
  - If # free pages per order 2^n lists are not well distributed, ***zone compaction*** may be called to remove fragmentations.

Zone Watermarks: Page Reclaim & Zone Balancing

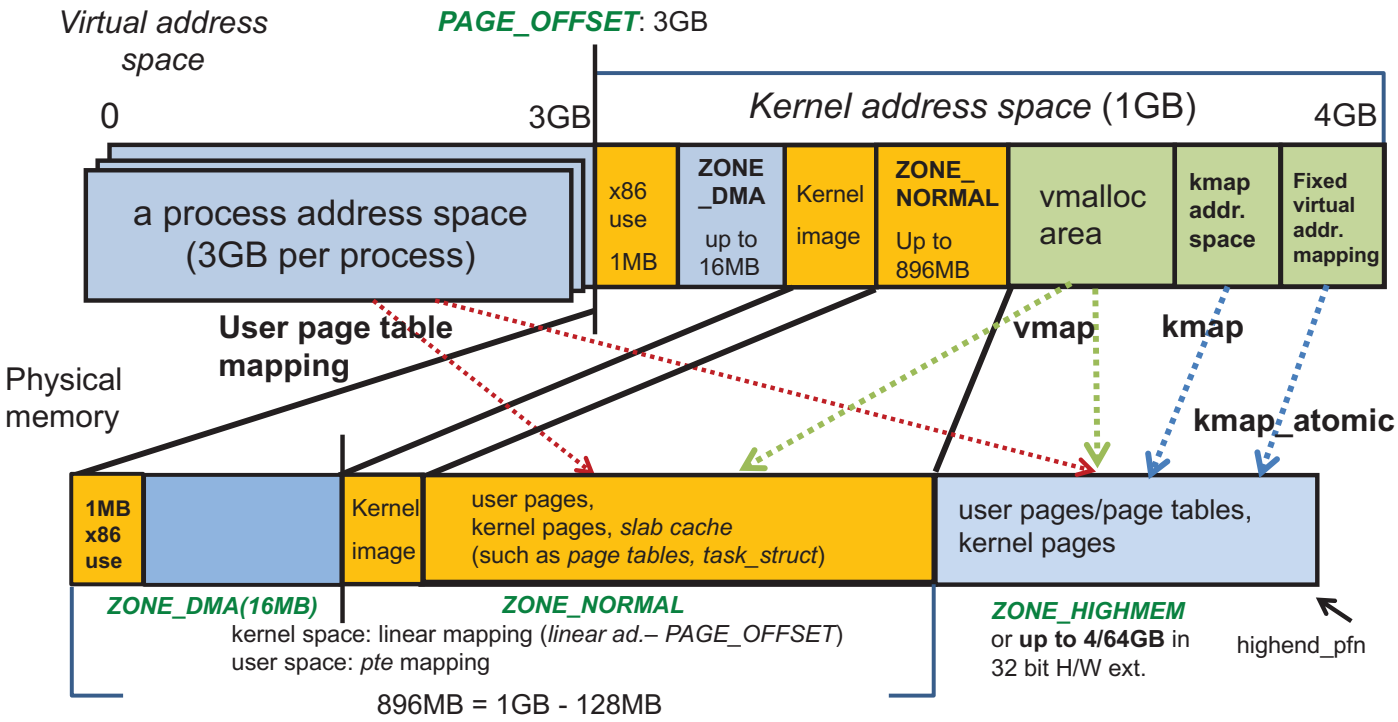


Source: Understanding the Linux® Virtual Memory Manager, Mel Gorman, PRENTICE HALL (2.6)

struct page: major members

struct page	Description		
u long flags	Atomic flags, some possibly updated asynchronously lock, **referenced, dirty, file-backed / swap-backed, lru, swapcache, active**, etc.		
*struct address_space *mapping*	File-backed	***address_space*** (for page cache) & **vma_interval_tree** in address_space for reverse mapping)	forma file (text, clean data)
	Swap-backed	***anon_vma*** for reverse mapping page_mapping() = ***swap address_space***	from a swap dev.
	Swap-backed, but not swapped out yet = anonymous pages	***anon_vma*** for reverse mapping	BSS, heap, stack, dirty data
	Free	NULL	
pgoff_t index	Our offset within mapping (eg. if file-backed, the offset in the file)		
atomic_t _map_count	number of ptes (that share this page), start at -1. If it is in swap_cache or in page-cache, 0		
atomic_t _count	Reference counting		
struct list_head lru	Have ***next & *prev**, used for ***INACTIVE_ANON, ACTIVE_ANON, INACTIVE_FILE, ACTIVE_FILE, UNEVICTABLE*** lru lists.		
u long private	Stores private data. Swap-cache page has **swap_entry** (block number in swap device)		
others	Information for managing Buddy/Slab-cache, mem-cgroup, kernel debugging		

Virtual & Physical Map (32bit machine)



Kernel Virtual Address Space

1. Kernel core
2. vmalloc area (virtual)
   - ***vmalloc():*** virtually contiguous but not in physical memory.
     - Physical: can be in any zone
   - But also used for physically contiguous allocation for kernel (virtual)
     - Physical : ***kmalloc()***, etc, : from slab/slub/slob allocators (slab cache)
     - Physical : ***alloc_pages():*** from buddy (buddy maintains physically contiguous page groups to minimize fragmentation)
3. kmap address space
   - A portion of kernel lower virtual address space to be mapped some pages in ***high-mem temporarily***. (explained later)
4. Fixed address mapping area
   - Reserved and used for mapping highmem pages in atomic manner (e.g., in IRQ)

Mapping

Linux Mapping

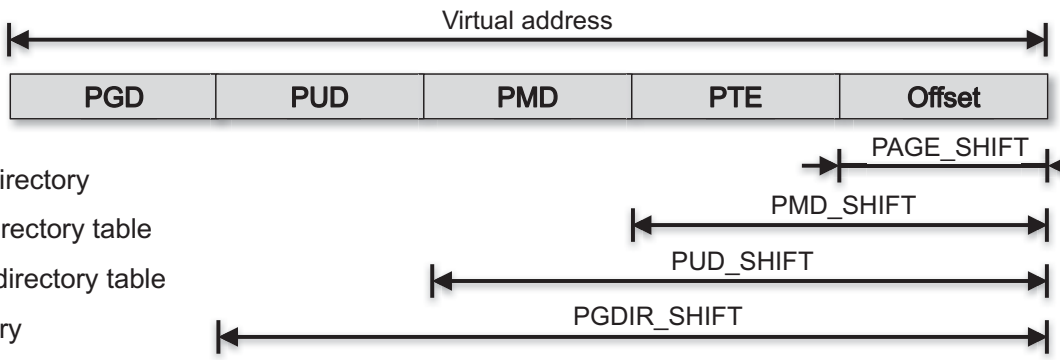
- Paging is used with an MMU.
- Page size
  - Kernel: 4MB, 4KB (no PAE) or 4KB, 2MB and 1GB (PAE or 64bit)
  - User: 4KB
  - Hugh page CONFIG option (2/4 MB pages) for users
- In x86, segmentation (see. GDT, LDT) is possible but not used actually.
- Level of page mapping in CPU architecture
  - 32 bit arch.: ***2 or 3-level*** to reduce the # of page tables (set of *pte's*)
    - Each user's address space is limited to 3GB, however, if *MMU* uses a more wider address bus, physical memory can be larger (up to 64GB).
  - 64 bit arch: ***4 level mapping***
  - Memory access overhead caused by multi-level mapping => ***TLB, MMU cache***

Mapping Levels in MMU

- 2–level mapping (x86_32)
  - 10bit (pgd offset) + 10bit(pte offset) + 12 bit(offset)
- 3-level mapping (x86_32 with PAE)
  - 2 bit(pgd offset) + 9 bit(pmd offset) + 9bit (pte offset) + 12 bit (offset in a frame)
- 4-level mapping (x86_64 bit)
  - 9bit (pgd offset) + 9bit (pud offset) + 9bit(pmd offset) + 9bit (pte offset) + 12 bit (offset) = 48 bit

4-level Page Table in Linux

- Linux page table is architecture-independent
  - 4-level page table structure
  - Fix-up codes exist in arch header files to make it compatible with 2 or 3-level mappings
  - Example
    - 4-level Linux page table → 2-level page table in x86_32
    - 4-level Linux page table → 3-level page table in x86_32 with PAE



PGD: Page global directory

PUD: Page upper directory table

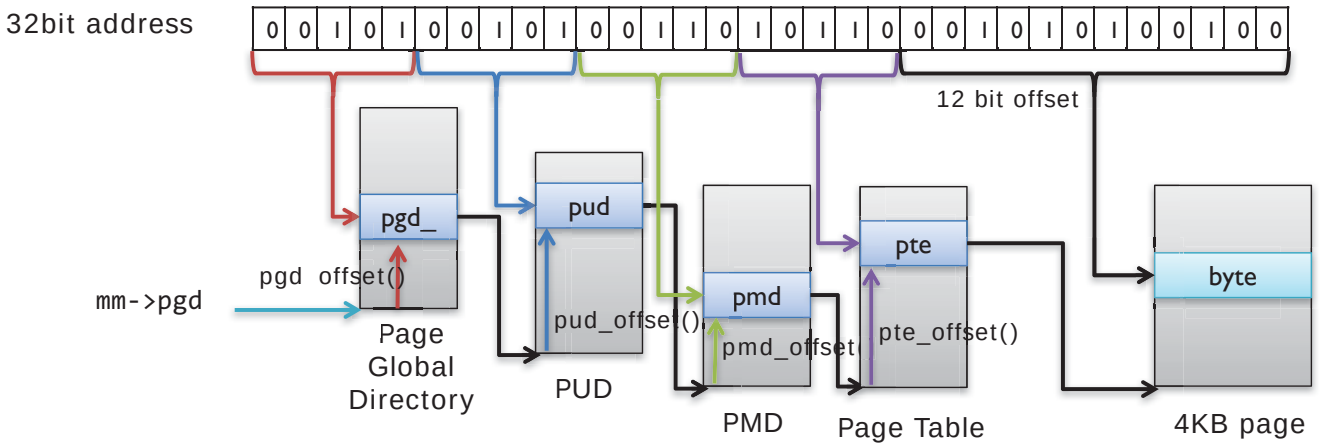
PMD: Page middle directory table

PTE: page table entry

4-level Page Table in Linux

- Page table entry lookup code

```
pgd_t* pgd;
pud_t* pud;
pmd_t* pmd;
pte_t* pte;
pgd = pgd_offset(mm, address);
pud = pud_offset(pgd, address);
pmd = pmd_offset(pud, address);
pte = pte_offset(pmd, address);
/* *pte is value of page table entry */
```

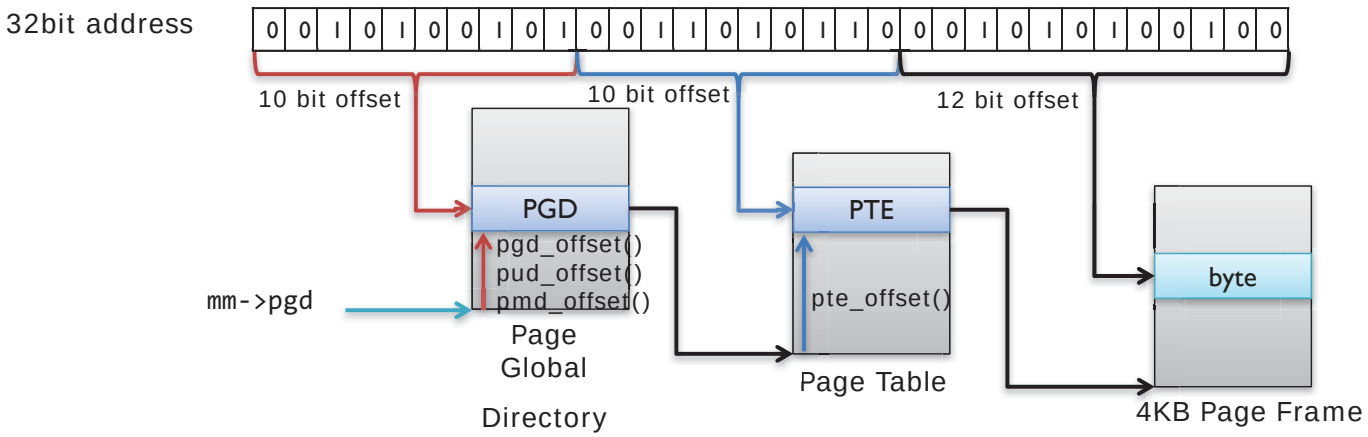


4-level Page Table in Linux

- Page table entry lookup code
  - Fixup codes

```
#define pud_offset(pgd, address) (pgd)
#define pmd_offset(pud, address) (pud)

/* in include/asm-generic/pgtable-nopud.h */
/* in include/asm-generic/pgtable-nopmd.h */
```

```
pgd_t* pgd;
pud_t* pud;
pmd_t* pmd;
pte_t* pte;
pgd = pgd_offset(mm, address);
pud = pud_offset(pgd, address);
pmd = pmd_offset(pud, address);
pte = pte_offset(pmd, address);
/* *pte is value of page table entry */
```



Mapping (3-level)

- mm_struct in a task_struct has a points to the pgd of a process.
  - pgd is one physical page frame, array of pgd_t for processes.
  - Mapping is enabled by loading: **mm_struct->pgd => cr3(x86)**
    - Context switch changes cr3 to PGD of next process
  - **High entries in PGD shares those of kernel PGD** (swapper_pg_dir)
    - Kernel mappings are consistent across processes
  - Each entry in PGD, PUD, PMD, PTE stores physical address
    - Obtaining physical address of a page is done by several macro family (explained later)
      - page_to_pfn(), __pa(address), etc.

Basic Types in 4-level x86 Mapping

```
typedef unsigned long  pteval_t;
typedef unsigned long  pmdval_t;
typedef unsigned long  pudval_t;
typedef unsigned long  pgdval_t;
typedef unsigned long  pgprotval_t;
typedef  struct { pteval_t pte; } pte_t;
```

pte Layout (32 ~ 64bit)

```
#define _PAGE_BIT_PRESENT      0    /* is present */
#define _PAGE_BIT_RW           1    /* writeable */
#define _PAGE_BIT_USER         2    /* userspace addressable */
#define _PAGE_BIT_PWT          3    /* page write through */
#define _PAGE_BIT_PCD          4    /* page cache disabled */
#define _PAGE_BIT_ACCESSED     5    /* ref bit: was accessed (raised by MMU) */
#define _PAGE_BIT_DIRTY        6    /* dirty bit: was written to (raised by MMU) */
#define _PAGE_BIT_PSE          7    /* 4 MB (or 2MB) page */
#define _PAGE_BIT_PAT          7    /* on 4KB pages */
#define _PAGE_BIT_GLOBAL       8    /* Global TLB entry PPro+ */
#define _PAGE_BIT_SOFTW1       9    /* available for programmer */
#define _PAGE_BIT_SOFTW2       10   /* " */
#define _PAGE_BIT_SOFTW3       11   /* " */
```


pte Layout (32 ~ 64bit)

```
#define _PAGE_BIT_PAT_LARGE     12    /* On 2MB or 1GB pages */
#define _PAGE_BIT_SPECIAL       _PAGE_BIT_SOFTW1
#define _PAGE_BIT_CPA_TEST      _PAGE_BIT_SOFTW1
#define _PAGE_BIT_SPLITTING     _PAGE_BIT_SOFTW2 /* only valid on a PSE pmd */
#define _PAGE_BIT_HIDDEN        _PAGE_BIT_SOFTW3 /* hidden by kmemcheck */
#define _PAGE_BIT_SOFT_DIRTY    _PAGE_BIT_SOFTW3 /* software dirty tracking for kernel
                                                    //  debugging */
#define _PAGE_BIT_NX            63    /* No execute: only valid after cpuid check */
```

Getting a *pte*

```
pgd_t *pgd;
pud_t *pud;
pmd_t *pmd;
pte_t *ptep, pte;
pgd = pgd_offset (mm, address);   // get by the virtual address
pud = pgd_offset (pgd, address);   // get by the virtual address
pmd = pmd_offset (pud, address); // get by the virtual address
ptep = pte_offset _map(pmd, address); // get by the virtual address
 /* last-level page table can be in highmem. So, kernel needs to map the last-level page
table page first, and then finds pte using pte_offset() */
ptep = pte_offset_map_lock(mm, pmd, address, &ptlp) ;
/* When to modify a pte, a page table lock (spinlock) should also be held */
```


Using & Setting a pte

Function	Description
mk_pte (page, pgprot)	takes a ***struct page*** and **protection bits** and combines them together to form the ***pte_t*** for insertion to a the page table.
pte_page (pte)	returns the ***struct page***, which corresponds to the PTE entry.
pmd_page (pmd)	returns the ***struct page*** containing the set of PTEs.
set_pte_at (mm, addr, ptep, pte)	set ***pte*** at a virtual address for MMU mapping
do_set_pte ()	explained later
pte_clear (), pmd_clear (), pgd_clear ()	clear corresponding entry to 0
ptep_get_and_clear (mm, addr, ptep)	clears an entry and returns the original value. This is important when some modification needs to be made to either the ***pte*** protection or the ***struct page*** itself.
ptep_clear_flush (vma, addr, ptep)	clear a ptep and flush a corresponding TLB entry
pte_none ()*, md_none (), pgd_none ()*	return 1 if the corresponding entry does not have any contents.
pte_present (), *pmd_present (), pgd_present ()*	return 1 if the corresponding page table entries have the ***PRESENT*** bit set. **If set, this means the physical frame is in memory.**

Using & Setting a pte: Dirty & Accessed Flag

- There are only two bits that are important in Linux, the dirty bit and the accessed bit mask in the pte's flag.
  - They are important to determine the working-set & are used in swapping.
  - Note: the ***struct page*** also has ***reference*** and ***dirty*** flag. (explained later)

Function	Description
pte_dirty()	Check Dirty bit
pte_young()	Check ACCESSED bit
pte_mkdirty()	SET DIRTY
pte_mkyoung()	Set ACCESSED
pte_mkclean()	Clear DIRTY
pte_mkold()	Clear ACCESSED

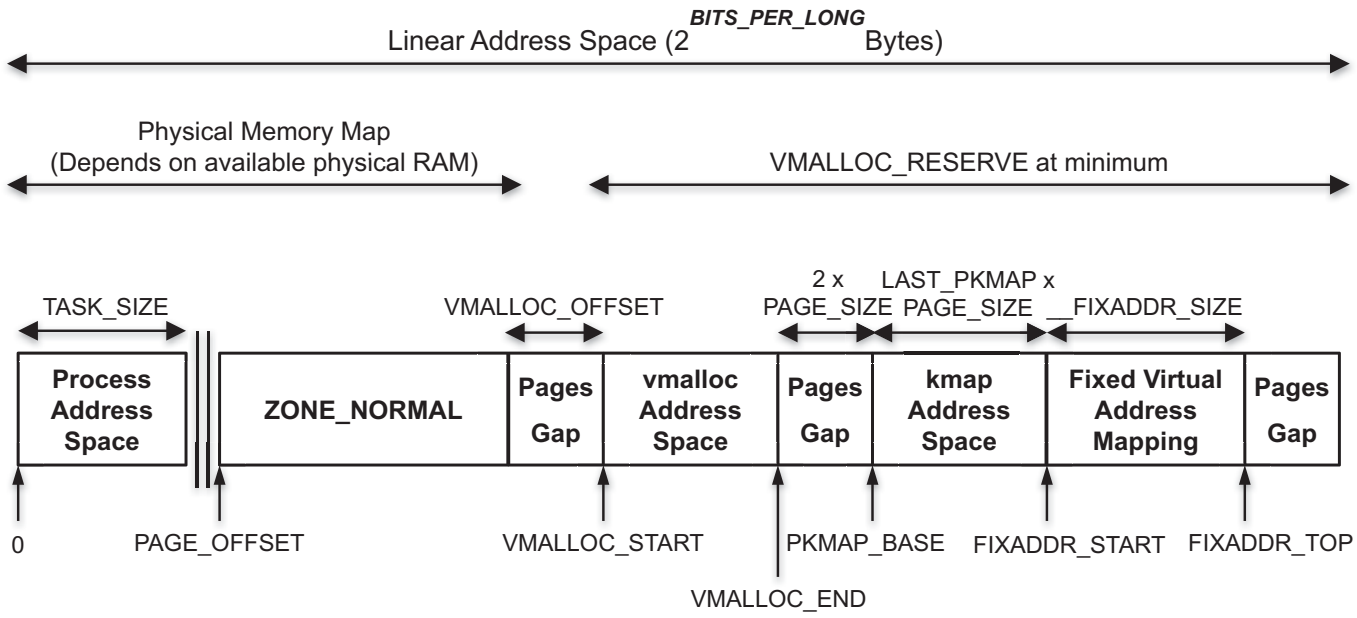
Allocating and Freeing Page Tables

- The allocation and freeing of physical pages
  - *__pgd_alloc(), __pud_alloc(), __pmd_alloc(), __pte_alloc(),*
  - *__pgd_free(), __pud_free(), __pmd_free(), __pte_free().*
  - *pte_alloc_one() uses* ***GFP_KERNEL | GFP_HIGHMEM (CONFIG_HIGHPTE)***
  - *pmd_alloc_one() uses* ***GFP_KERNEL***
  - *pgd_alloc(), pud_alloc(), pmd_alloc(), pte_alloc()*
    - *Wrapper function of __pxx_alloc() and __pxx_offset()*
  - ***GFP_xxxx*** specifies the ***"page allocation option"***
    - Emergency level of the allocation,
    - Preferred zones,
    - Can do I/O for swap out at reclaim?,
    - Retry possible?, etc.

Kernel Page Tables (x86)

- Kernel page size = 4KB, 4MB (32bit) or 4KB, 2MB, 1GB (64bit)
  - If Page Size Extension (*PSE*) bit is supported by the H/W.
- The static PGD (***swapper_pg_dir***) is the **pgd** for the kernel.
- Bootstrap (before paging is initialized)
  - 1st phase: 1~9MB is mapped to the kernel (*pg0, pg1*: each 4MB page)
    - First 1MB (for H/W, eg, IRQ vector, etc.), the following 8MB : kernel core.
  - 2nd phase:
    - ***kernel_physical_mapping_init()***
      - init page tables for ZONE_DMA + ZONE_NORMAL (1GB, 2/4MB or 4KB page)
    - ***pagetable_init()***
      - init page tables for kmap()
    - ***kmap_init()*** **(for *kmap_atomic()*)**
  - Paging is turned on in kernel mode.
    - ***swapper_pg_dir → cr3***

Kernel Address Space (32bit)

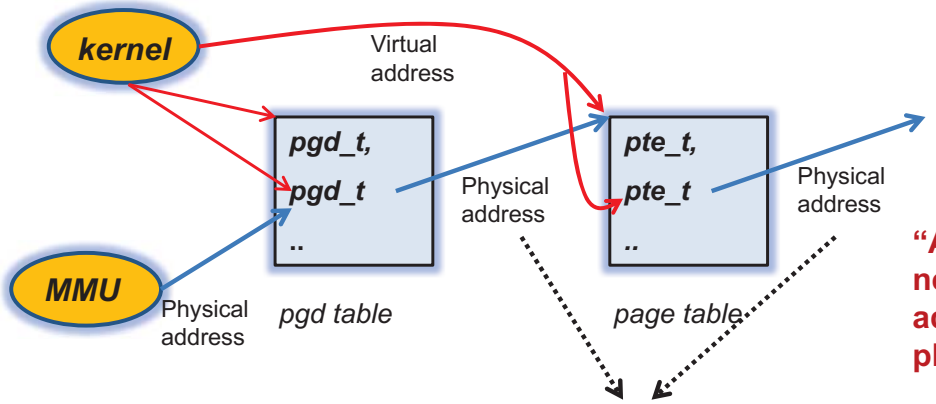


Kernel Address Space (64bit)

	Begin	End	Size	Usage
	0000000000000000	00007fffffffffff	47 bits bits	User space
	unused hole			
	ffff800000000000	**ffff87ffffffffff**	43 bits	Guard hole
PAGE_OFFSET	ffff880000000000	ffffc7ffffffffff	64 TB	**Direct mapping**
	ffffc80000000000	ffffc8ffffffffff	40 bits	Hole
	ffffc90000000000	ffffe8ffffffffff	45 bits	**vmalloc/ioremap**
	ffffe90000000000	ffffe9ffffffffff	40 bits	hole
	ffffea0000000000	ffffeaffffffffff	1 TB	virtual memory map
	unused hole			
	ffffec0000000000	fffffc0000000000	16 TB	kasan shadow memory
	unused			
	ffffff0000000000	ffffff7fffffffff	39	%esp fixup stacks
	unused			
	ffffffef00000000	ffffffff00000000	64 GB	EFI region mapping space
	ffffffff80000000	ffffffffa0000000	512 MB	**kernel text**
	ffffffffa0000000	ffffffffff5fffff	1526 MB	**module mapping space**
	ffffffffff600000	ffffffffffdfffff	8MB	vsyscalls
	ffffffffffe00000	ffffffffffffffff	2MB	unused hole

page table / pte accesses by the kernel & MMU

- When the kernel accesses (& sets) ***user page tables*** (in kernel area), it uses virtual addresses because the kernel also runs on VM !
- But the contents in ***page (directory) tables*** must physical addresses for MMU.

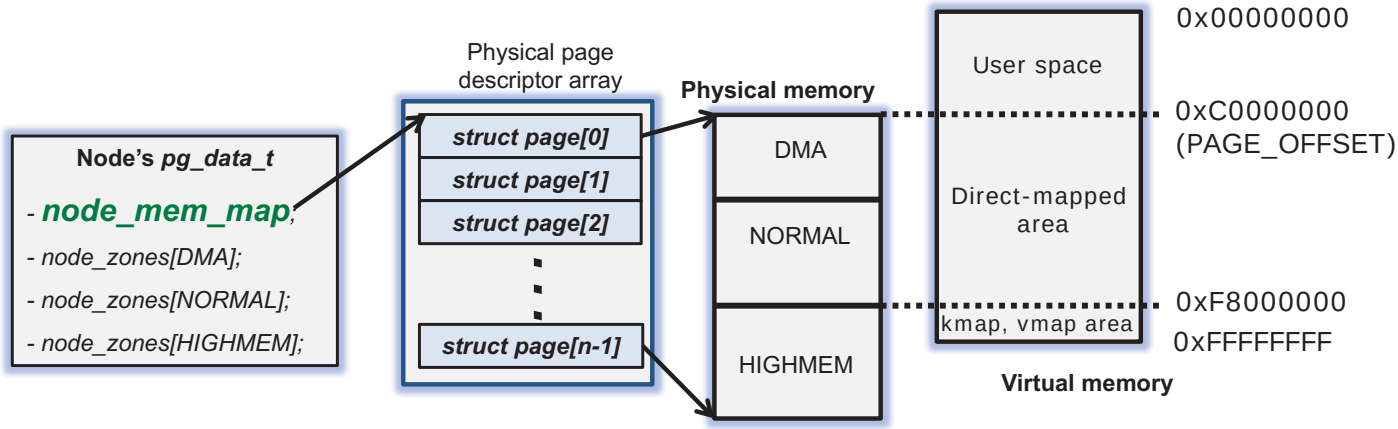


"All page-table accesses need not only virtual addresses but also physical addresses."

- Kernel must put the physical address of the page table into pgd for the *MMU*.

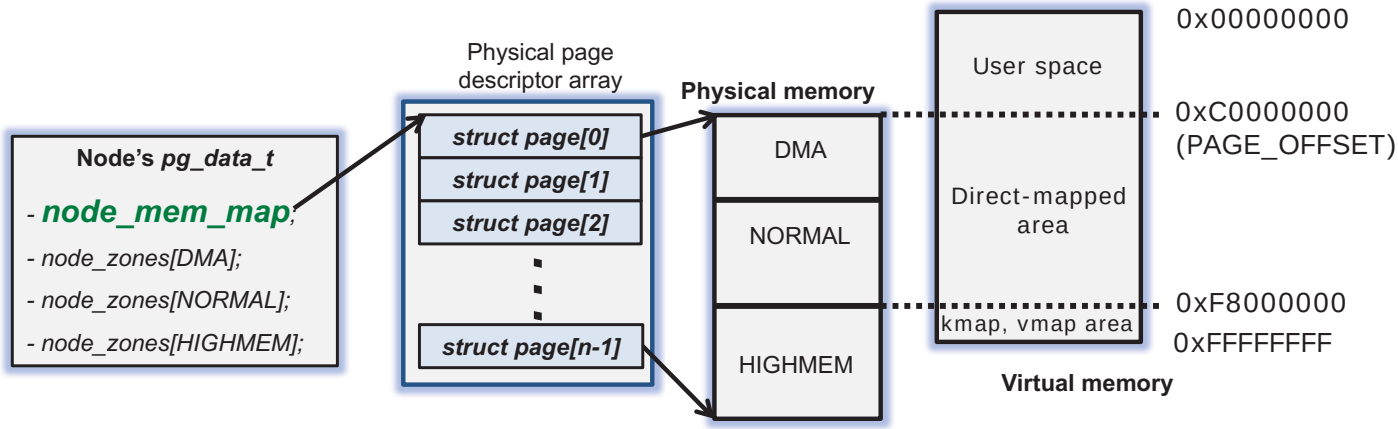
struct page → PFN → physical address

Translation between Page, PFN, vaddr, and paddr



- #define page_to_pfn(page) (page – mem_map)
- #define pfn_to_page(pfn) (mem_map + pfn)
- #define __pa(vaddr)(vaddr – PAGE_OFFSET)
- #define __va(paddr)(paddr + PAGE_OFFSET)

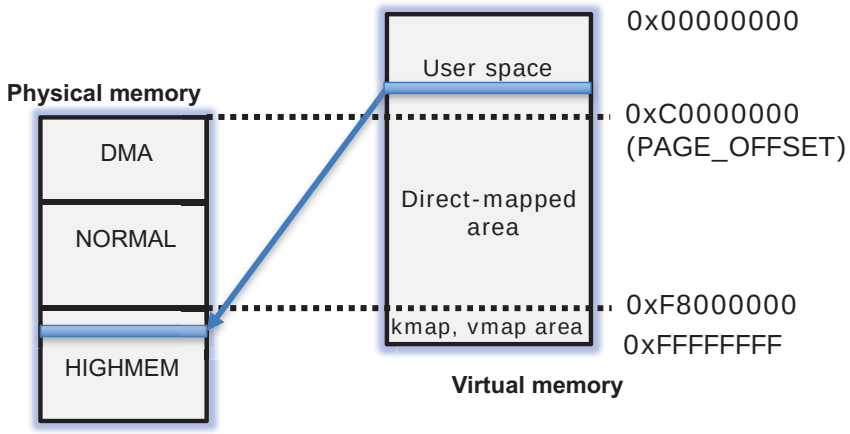
Translation between Page, PFN, vaddr, and paddr



- #define virt_to_page(kaddr) pfn_to_page(__pa(kaddr) >> PAGE_SHIFT)
- #define lowmem_page_address(page) \
  __va(page_to_pfn(page) << PAGE_SHIFT)
- void* page_address(page)
  - If lowmem page, return lowmem_page_address()
  - Otherwise, lookup kmap area and finds out PTEs having page_to_pfn(page)

High-mem/User Space Access by Kernel

- HIGHMEM page frames cannot be directly accessed in kernel
- It should be mapped in the kernel area (128MB area) in virtual memory first

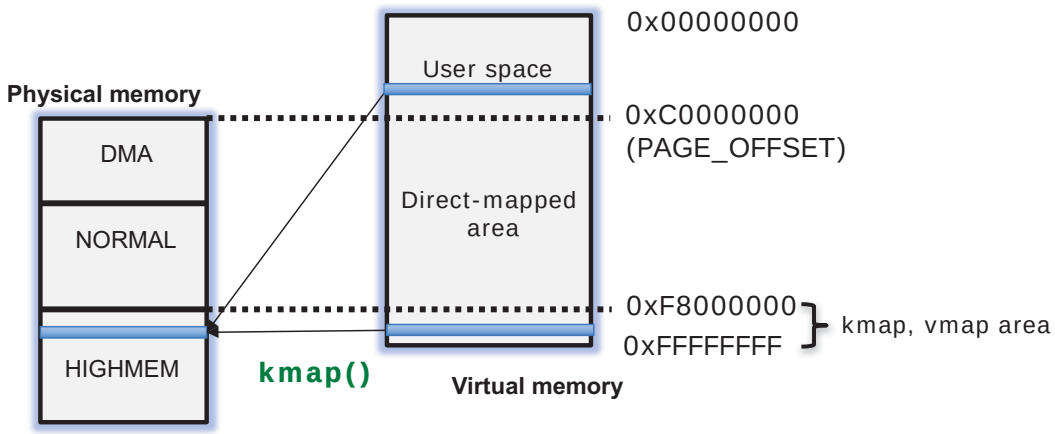


High-mem/User Space Access by Kernel

- Where HIGHMEM pages used
  - Page caches (file caches)
  - User process's memory pages
- Reason
  - Kernel hardly accesses those pages
    - When to copy page cache from/to user process through read/write system call
    - When to copy user pages for copy-on-write
- Solution
  - Temporarily maps user pages in the 128MB addressing area

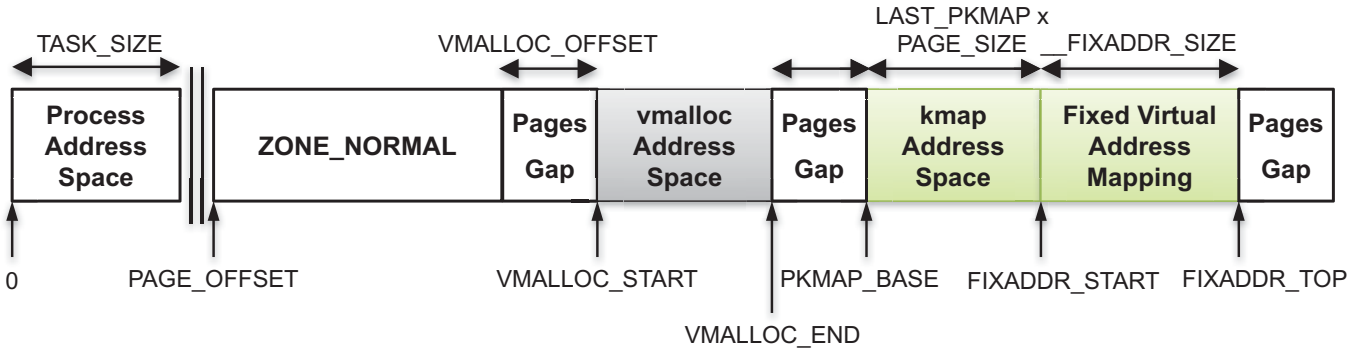
High-mem/User Space Access by Kernel

- By mapping HIGHMEM page frames on-demand, kernel codes can manipulate data in the page frame



High-mem/User Space Access by Kernel

- kmap(page) maps page in kmap area and returns virtual address
  - If the page is a lower page below ***HIGHMEM*** then return
    - This physical page can be mapped by a fixed kernel ***pte***.
  - Else, call ***kmap_high()*** -> call ***map_new_virtual()***
    - Scan to get free ***pte*** from the small page table for kmap area (use ***last_pkmap_nr***) & set mapping
    - Return the virtual address.



User's page table/space Access by the kernel

1. Kernel address mapping & conversion up to ZONE_NORMAL
   - Kernel data structures such as ***user page tables, task_struct*** in (up to) *ZONE_NORMAL* can be mapped by ***the normal & fixed kernel page tables. (by MMU)***
   - The kernel can get the physical addresses of them by ***kaddr - PAGE_OFFSET.*** (because up to ***ZONE_NORMAL***, they are linearly mapped.)

User's page table/space Access by the kernel

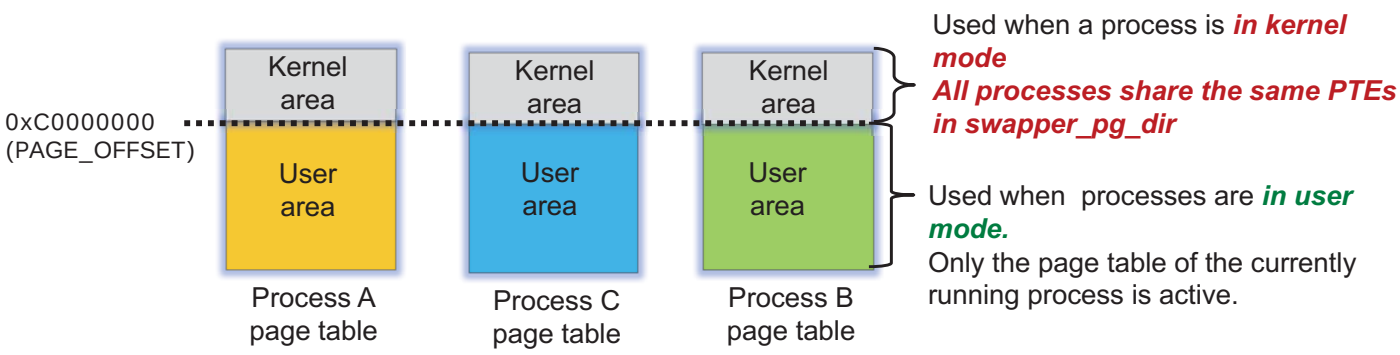
2. Kernel address mapping & conversion in ZONE_HIGHMEM
   - Kernel data structures such as ***user page tables*** in *ZONE_HIGHMEM* can be mapped by ***the temporary (& small) kernel page table (kmap area).*** That is a temporary virtual address is used to save space.
   - **void* page_address**(page)
     - If lowmem page, return lowmem_page_address()
     - **Otherwise, lookup kmap area and finds out PTEs having page_to_pfn(page)**
   - The kernel gets the physical addresses of them by the ***frame #*** in the small page table for ***kmap*** area. And then, use (pa/4K) as the index of ***struct page*** in ***mem_map*** array.

Translation Lookaside Buffer (TLB)

- Translation Lookaside Buffer (TLB)
  - A small associative memory that caches virtual to physical page table resolutions., machine dependent.
  - Locality of reference is important!
  - When the page tables have been updated, such as after a page fault has completed, the processor may need to update the TLB for that virtual address mapping.

Translation Lookaside Buffer (TLB)

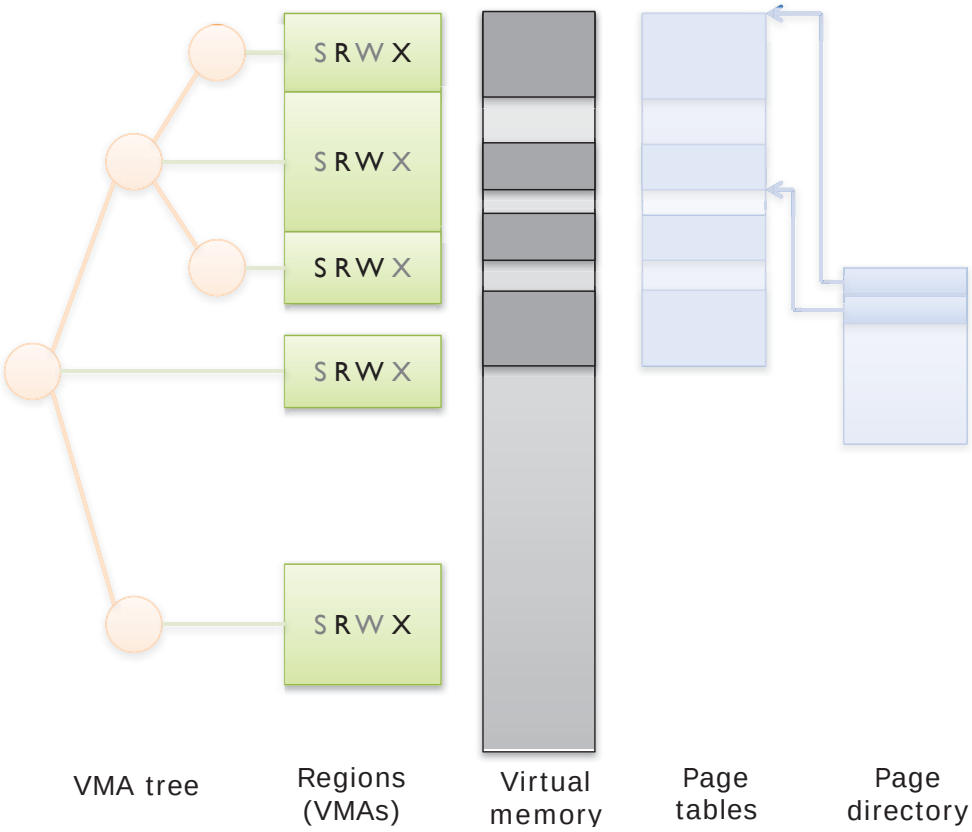
- TLB loading and TLB flushing : expensive
  - When context switching, Linux will avoid loading new page tables using Lazy TLB Flushing,
  - The call to switch_mm(), which results in a TLB flush, is avoided by borrowing the mm_struct used by the previous task and placing it in task_struct→active_mm. This technique has made large improvements to context switch times.



Process Virtual Address Space Management

Process Virtual Address Space Overview

- mmap
- page fault
- munmap



Source: A. T. Clements, et al., Scalable Address Spaces Using RCU Balanced Trees, ASPLOS 12

Process Virtual Address Space Overview

- mmap

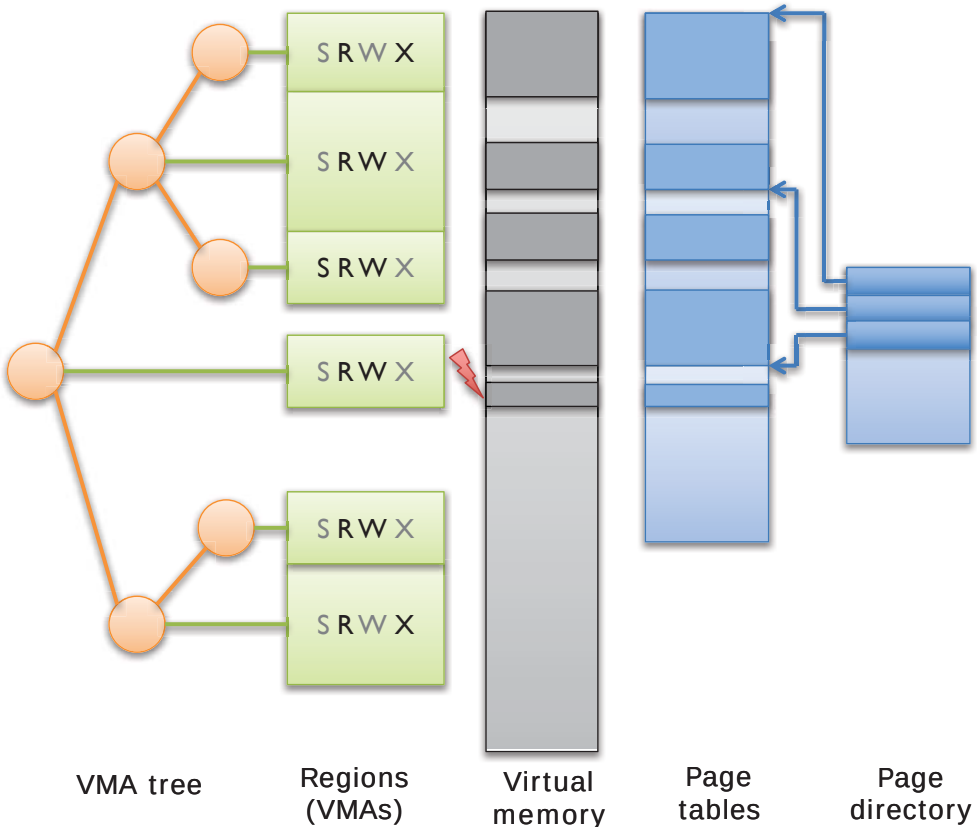
  create VMAs

- page fault

  create and fill page table on demand

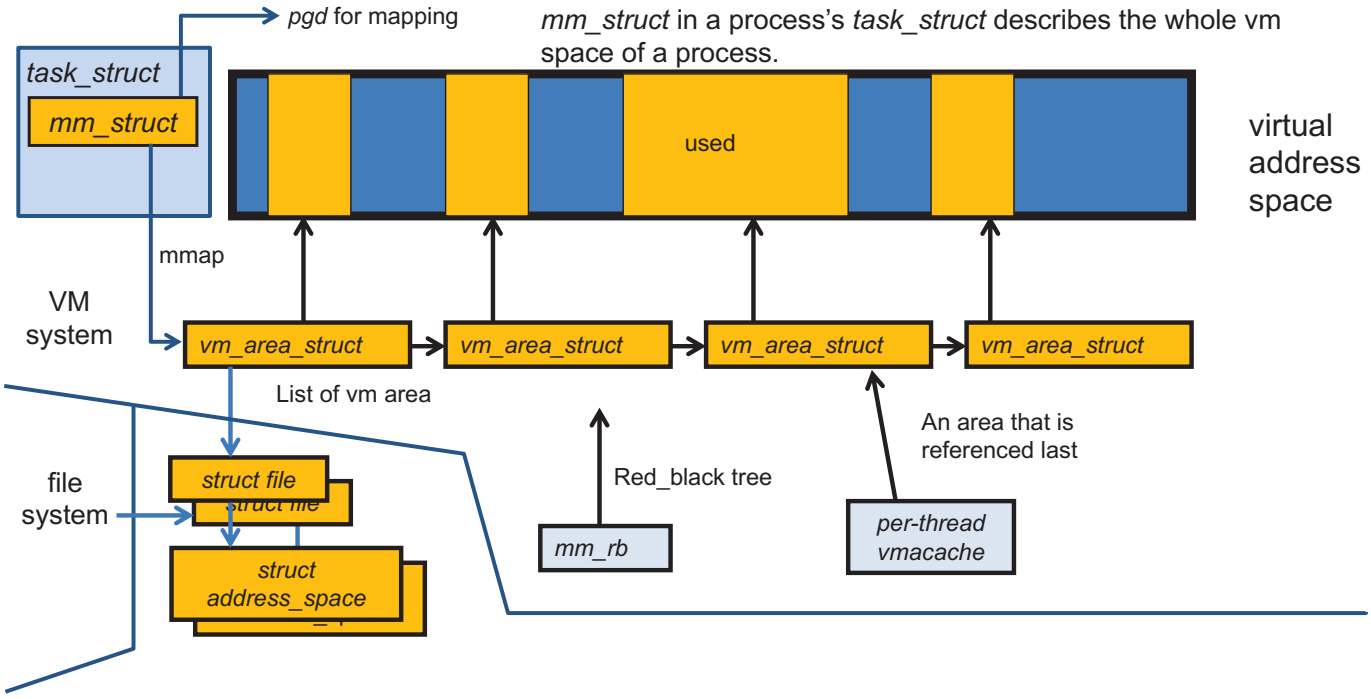
- munmap

  remove VMAs, clear page tables free pages



Source: A. T. Clements, et al., Scalable Address Spaces Using RCU Balanced Trees, ASPLOS 12

Process Address Space (User)



- The ***address_space*** of a file describes the whole vm space of a file like a ***vm_area_struct.***

struct task_struct: “sched.h”

```
struct task_struct {
        volatile long state;   /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        ::
/*  scheduler things */
        ::
        struct mm_struct *mm, *active_mm;
        ::
        struct files_struct *files; /* open file information, not for VM*/
                /* pointer to  struct file  * fd_array[NR_OPEN_DEFAULT]; */
        ::
/* signal things */
        ::
}
```


Process Address Space: *mm_struct:* 4.x include/linux/mm_types.h

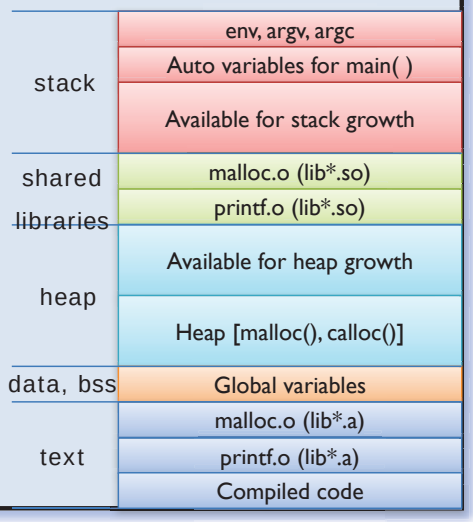
```
struct mm_struct {
    struct vm_area_struct *mmap;          // list of VMAs
    struct rb_root mm_rb;            // rb-balanced tree root (VMA tree)
    u32 vmacache_seqnum;                 // per-thread vmacache sync.
    unsigned long (*get_unmapped_area) (struct file *filp,
                    unsigned long addr, unsigned long len,
                    unsigned long pgoff, unsigned long flags);
    unsigned long mmap_base;             /* base of mmap area */
    unsigned long mmap_legacy_base; /* base of mmap area in bottom-up
                                        allocations */
            // used when 64 bit machine emulate the old 32 bit arch.
    unsigned long task_size;             /* size of task vm space */
    unsigned long highest_vm_end;        /* highest vma end address */
    ::
```

Process Address Space: *mm_struct:* 4.x include/linux/mm_types.h

```
pgd_t * pgd;          // page directory frame address
atomic_t mm_users;                    /* How many users with user space? */
atomic_t mm_count;    /* How many references to "struct mm_struct" */
atomic_long_t nr_ptes;                /* PTE page table pages */
atomic_long_t nr_pmds;                /* PMD page table pages */
int  map_count;                       /* number of VMAs */
spinlock_t page_table_lock;           /* Protects page tables and some counters */
struct rw_semaphore mmap_sem;  // lock for VMA list access
struct list_head mmlist;        /* List of maybe swapped mm's.  These are globally */
      /*  strung together off init_mm.mmlist, and are protected by mmlist_lock */
unsigned long hiwater_rss;   /* High-watermark of RSS(Resident Set Size) usage */
                  /* accounting */
unsigned long hiwater_vm;      /* High-watermark virtual memory usage */
```


Process Address Space: *mm_struct:* 4.x include/linux/mm_types.h

```
unsigned long total_vm;        /* Total pages mapped by all VMA regions in the process */
unsigned long locked_vm;       /* number of resident pages locked in memory */
unsigned long pinned_vm;       /* Refcount permanently increased */
unsigned long shared_vm;       /* Shared pages (files) */
unsigned long exec_vm;         /* VM_EXEC & ~VM_WRITE */
unsigned long stack_vm;        /* VM_GROWSUP/DOWN */
unsigned long def_flags; // VM_LOCKED or not,
                  // if 1, all future mappings are locked by default
unsigned long start_code, end_code, start_data, end_data;
unsigned long start_brk, brk, start_stack;
unsigned long arg_start, arg_end, env_start, env_end;
                  // start and end addr. of argv[], env[]
::
```



Process Address Space: *mm_struct:* 4.x include/linux/mm_types.h

```
/* Special counters, in some configurations protected by the
 * page_table_lock, in other configurations by being atomic. */

struct mm_rss_stat rss_stat;  // number of memory resident pages

struct linux_binfmt *binfmt;  // address of load_binary(), core_dump(), etc. (eg. elf_binary)

cpumask_var_t cpu_vm_mask_var; // A bitmask representing all possible CPUs in an SMP
                               // system. The mask is used by an InterProcessor Interrupt
                               // (IPI) to determine if a processor should execute a particular
                               // function or not. This is important during TLB flush for each CPU.
/* Architecture-specific MM context */
mm_context_t context;   // x86: segment information ldt

unsigned long flags; /* Must use atomic bitops to access the bits */

struct core_state *core_state; /* coredumping support */
```


Process Address Space: *mm_struct:* 4.x include/linux/mm_types.h

```
    :: AIO thigs;
        :: cgroup things;

    /* store ref to file /proc/<pid>/exe symlink points to */
    struct file *exec_file;

    ::  // huge page things;
    pgtable_t  pmd_huge_pte; /* protected by page_table_lock */
    ::  // NUMA things

    struct uprobes_state uprobes_state;

    ::
};
```

mm_struct access functions: 4.x

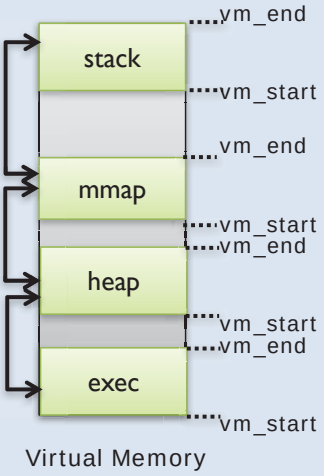
Function	Description
mm_init()	Initializes an *mm_struct* by setting starting values for each field, allocating a *PGD*, initializing spinlocks, etc.
allocate_mm()	Allocates an *mm_struct* from the ***slab*** allocator
mm_alloc()	Allocates an *mm_struct* using *allocate_mm()* and calls *mm_init()* to initialize it. This is used only for **execve()**.
exit_mmap()	Walks through an *mm_struct* and unmaps all VMAs associated with it
copy_mm()	Makes an exact copy of the current tasks *mm_struct* needs for a new task. This is only used during ***fork().***
free_mm()	Free the *mm_struct* and return it to the ***slab*** allocator
get_task_mm()	Take a process's mm and increase reference count (mm_users)
mmput()	Return the mm, and decrease reference count (mm_users) If ref. counter (mm_users) == 0, release vm_areas and call mmdrop(). This is called during **exit()**
mmdrop()	If ref. counter(mm_count) == 0, call free_mm()

Regions

- Region (or mmaped region)
  - A virtually contiguous space that never overlaps and represents a set of addresses with the same protection and purpose.
  - **Text, data, bss, heap, stack, read-only shared library, memory-mapped file, shared-memory, etc.**
  - Can be viewed at /proc/PID/maps
  - Region information: **vm_area_struct**
    - If a region is backed by a file (text, data), it has the pointer to the **struct file.**
    - If anonymous, it has a pointer to **anon_vma** instead of **struct file.**

vm_area_struct

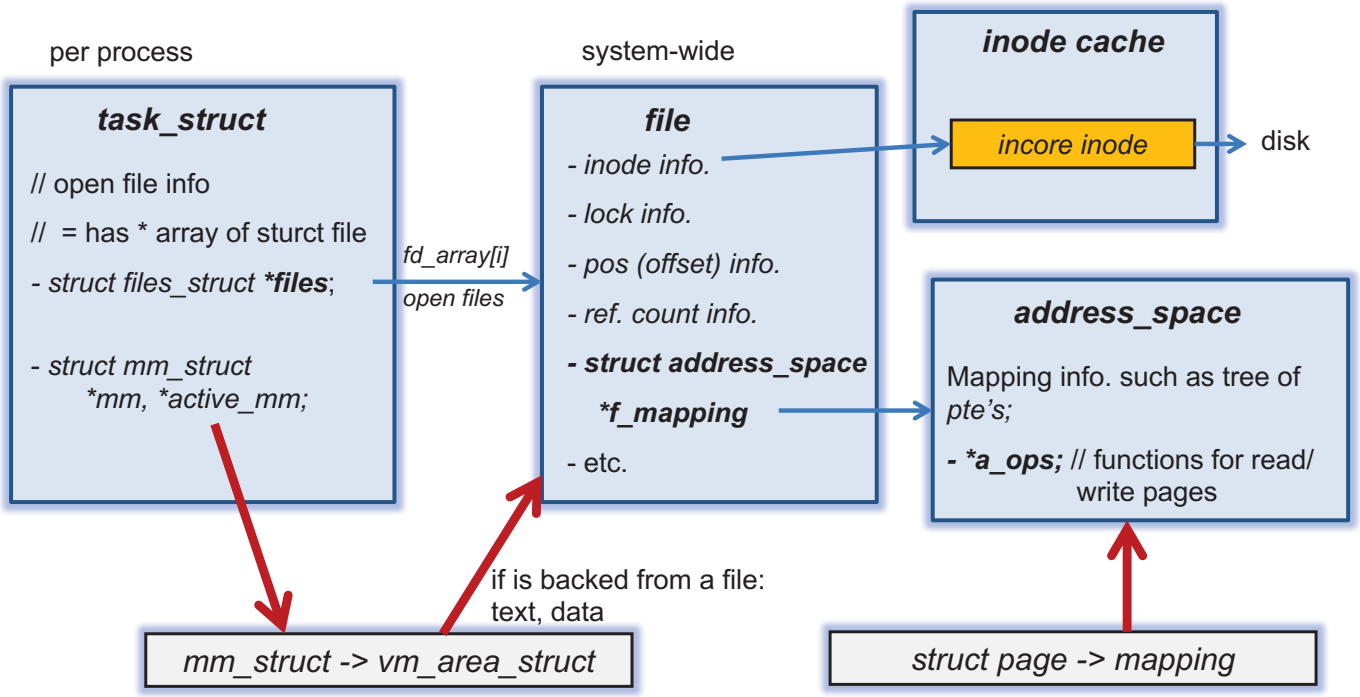
```
struct vm_area_struct {
        unsigned long vm_start;        /* Our start address within vm_mm. */
        unsigned long vm_end;      /* The first byte after our end address within vm_mm. */

        /* linked list of VM areas per task, sorted by address */
        struct vm_area_struct *vm_next, *vm_prev;  /* links */
        struct rb_node vm_rb     // left & right of red_black tree

         /*
         * Largest free memory gap in bytes to the left of this VMA.
         * This helps get_unmapped_area find a free area of the right size.
         */
        unsigned long rb_subtree_gap;
```



vm_area_struct

```
        struct mm_struct *vm_mm;        /* The address space we belong to. */
        pgprot_t vm_page_prot;  // some common flags for pte's protection field such as RW,
        unsigned long vm_flags;        /* Flags, see mm.h. */
        ::
        struct list_head anon_vma_chain; /* heap, stack vma chain */
        struct anon_vma *anon_vma;      /* some information for anon COW handling */
/* Function pointers to deal with this struct. */
        const struct vm_operations_struct *vm_ops;
             // open, close, fault, mmap functions for demand paging, etc.
        /* Information about our backing store: */
        unsigned long vm_pgoff;        /* Offset (within vm_file) in PAGE_SIZE  units, */
        struct file * vm_file;         /* File we map to (can be NULL). */
                                        /* linked to the file's struct address_space (inode) */
          ::
};
```

struct file & address_space for VM & file system



vm_flags of a region

Basic Protection Flags	
VM_READ	Pages may be read.
VM_WRITE	Pages may be written.
VM_EXEC	Pages may be executed.
VM_SHARED	Pages may be shared.
VM_DONTCOPY	VMA will not be copied on fork.
VM_DONTEXPAND	Prevents a region from being resized.
VM_GROWSDOWN	VMA can be expanded downwardly

mmap related flags
VM_STACK_FLAGS, VM_DATA_FLAGS, etc..

Locking Flags	
VM_LOCKED	If set, the pages will not be swapped out. It is set by mlock().
VM_IO	the area is an mmaped region for I/O to a device. It will also prevent the region from being core dumped.
VM_RESERVED	Do not swap out this region.

madvise() flags	
VM_SEQ_READ	A hint that pages will be accessed sequentially.
VM_RAND_READ	A hint stating that read-ahead in the region is useless.

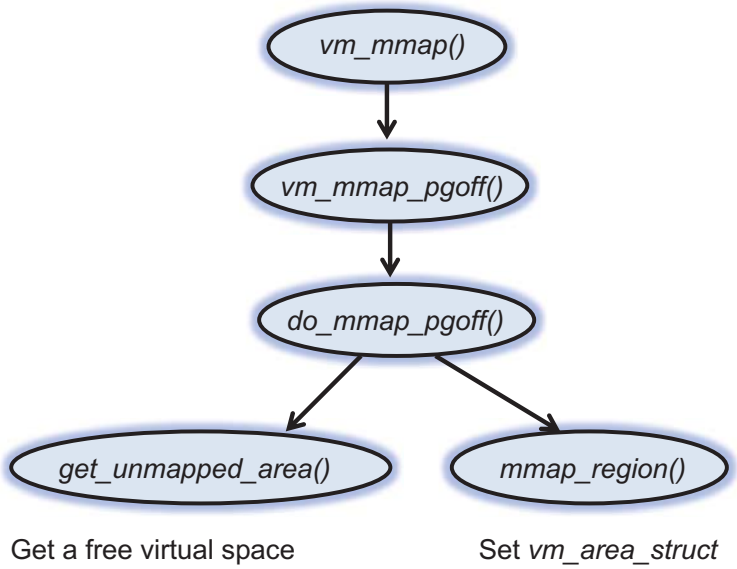
Creating a Memory Region (1/2)

1. do_fork() for process creation
   - copy_mm()
     - Generic function for real fork() or clone()
     - If CLONE_VM is set
       - Shared parent's mm_struct (increase mm_users counter)
     - Else
       - Allocate a new mm_struct
       - Copy mm_struct ; // including vmas
       - Alloc new pgd; null contents -> page fault -> use sharing(text) or COW
2. clone() for thread creation
   - Share the old mm_struct, but use a different stack pointer
   - Call copy_mm() with CLONE_VM set

Creating a Memory Region (2/2)

3. exec() for program loading into an existing process
   - Drop the old mm_struct,
   - Allocate a new mm_struct,
   - For each region, initialize "vm_area_struct"s, but no page tables.
     - Call vm_mmap() for text & data,
     - Call setup_arg_pages() for stack,
     - Call set_brk() for bss & heap

Creating a Memory Region

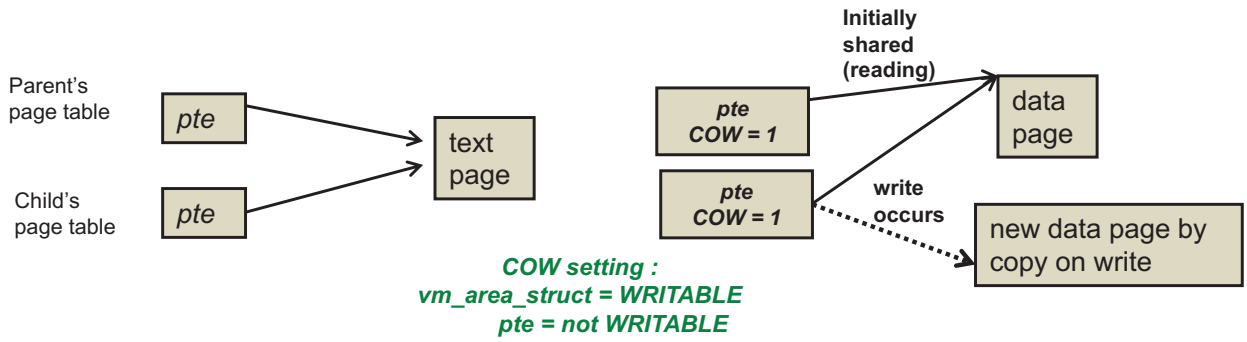


Merging Contiguous Regions, Remapping

- Merging
  - By calling vma_merge()
  - Usually, used when extending an anon region (heap or stack)
  - Allocate a new vma and merge it with the existing one
- Remapping
  - Provided for grow or shrink an existing memory mapping
  - If a region is to be moved, do mremap() first calls get_unmapped_area() to find a region large enough to contain the new resized mapping and then calls move_vma().

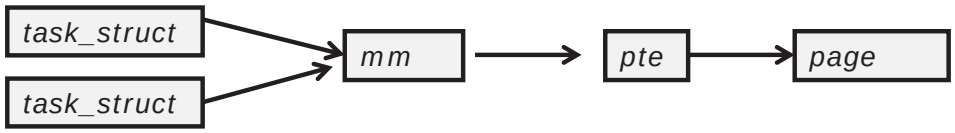
Memory Sharing

- Fork
  - Parent and child have their own mm_struct, page tables, pte's.
  - But shares the whole physical pages initially (when forked).
  - Text : shared forever
  - Data, Heap, Stack : COW (Copy on Write) is used !
    - Initially shared, but if a writing is occurred, the new page must be created by copying.
    - For this, the two pte's have COW(Copy on Write) settings. When a write occurs and COW is set, copy the page frame by allocating a new frame.



Memory Sharing

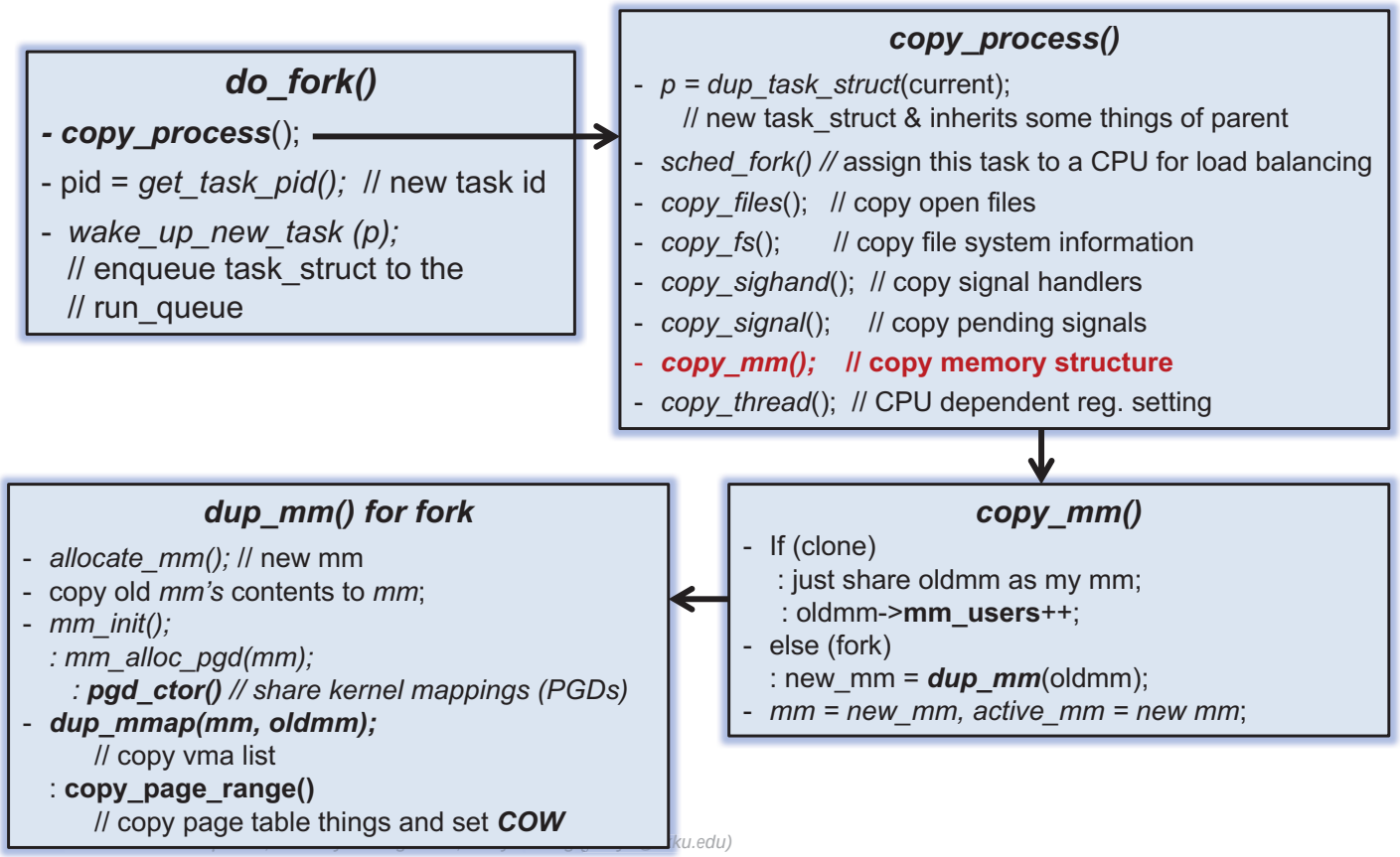
- Threads (by clone())
  - Share all regions by sharing the existing mm_struct.
  - All shared but just use per-thread stack pointers.
- So threads are very light weighted. (fast creation, no more kernel resources)



fork() / clone() syscalls

- Create a new process (process or thread)
  - fork : create a child process & its new address space that has the same memory layout with the parent.
    - Old mm is copied to the new mm. (same addr. space, sharing)
    - Text is shared.
    - Initially, all other regions are shared. However, if one of the parent and child write-accesses a page in this regions, the page will be copied for two processes. This called COW (Copy On Write).
  - clone : thread creation (pthread lib. calls clone() for users)
    - struct mm itself is shared for sharing of all regions. (share same page tables)
    - The new thread just uses a different stack pointer.
- fork(), clone() will call do_fork() → copy_process() with different flag values (cloning or not).

do_fork()



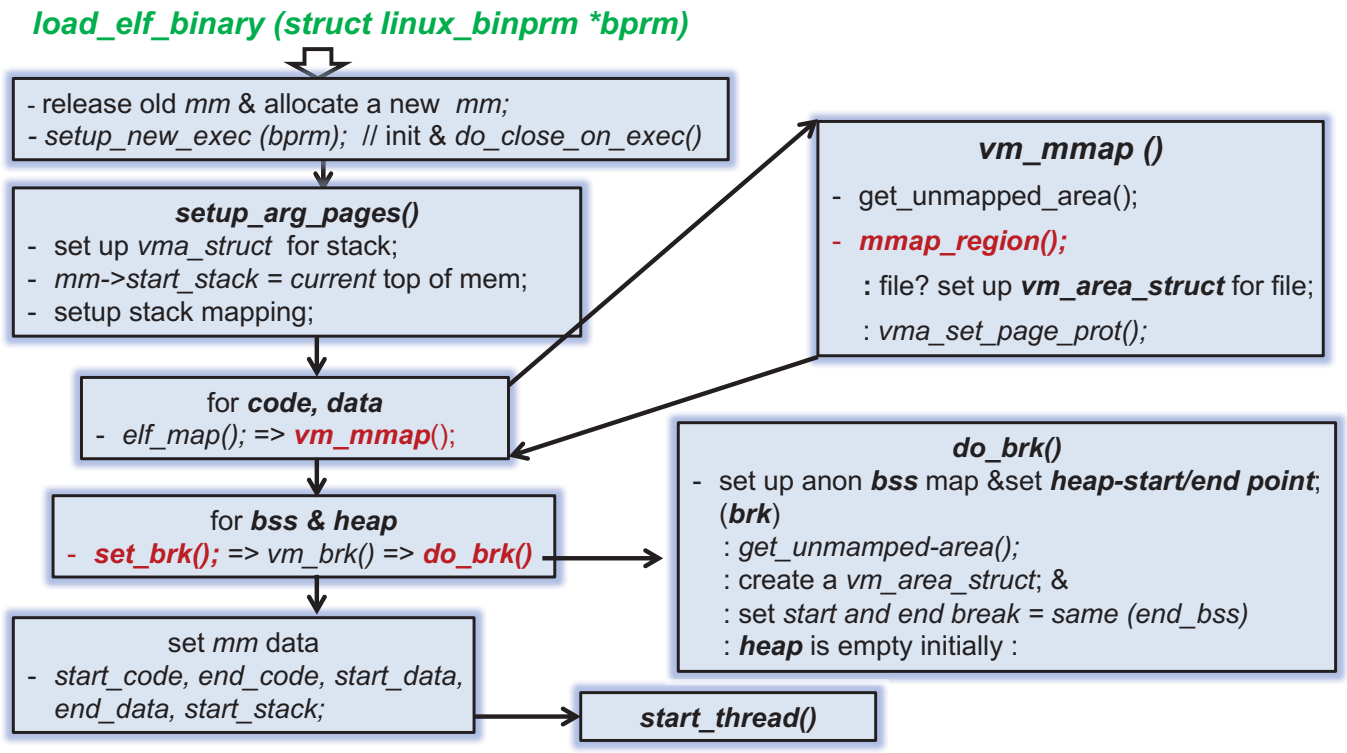
active_mm & mm

- task struct's **active_mm** and **mm**
  - mm is a virtual address space of a process
  - active_mm is an effective mm in current context (active_mm->PGD is the value of CR3)
  - Kernel threads share mm of previous task
    - kthreadd (mm is NULL) forks to create a new kthread
    - In context switch, if next is kthread, prev task's active_mm is used
- Lazy TLB handling (a process without an mm pointer)
  - Context switch to kernel thread
    - No need to flush TLB because kthread works on kernel address (above PAGE_OFFSET).
    - So mm-switching is deferred. So TLB is valid!
  - active_mm is always remembered across context switch
    - When flushing a specific entry in TLB, if the entry is not current active_mm's one, no TLB entry flush occurs

do_execve() for exec() syscall: 4.x

- Load a new program into an existing process.
- Prepare a new mm & vma_struct's for regions & demand paging
  1. open the file;
  2. sched_exec();// in an SMP, do load-balancing
  3. prepare_binprm(bprm); // binary format info. per each binary format (a.out, elf)
  4. bprm_mm_init(bprm); // alloc mm, initialize mm, alloc pgd, and initialize stack vm_area (4KB)
  5. exec_binprm() → search_binary_handler(); // call load_elf_binary() or load_aout_binary()
- Finally, has a mm struct and new vm_area_struct's for (code, data, bss, stack). (heap's vm_area_struct, later when necessary)
- But no pte's at all. (will be set by demand paging later at page faults)

load_elf_binary()



Kernel Dynamic Memory Allocation & Physical Memory Management

Dynamic Kernel Memory Allocation (1/3)

1. Physically contiguous allocation
   - For DMA or many important kernel data structures such as user page tables.

- Spaces for physical page allocation (3 ways)

1. ***per-CPU pageset. (hot & cold cache):*** A **single-page** request => a ***hot page*** is allocated.
   1. ***Hot page: recently freed (free()) page by the kernel. It's highly probable in H/W cache.***
   2. ***Cold page: freed page by swapout. It may not be in H/W cache.***

Dynamic Kernel Memory Allocation (2/3)

2. ***free_area (buddy area) per ZONE: 2^{order} (order = 0,1,2,3,…):*** A multi-page request.
   - Kernel-heap for contiguous dynamic allocation.
   - Keep lists of free memory segments per size.
   - If a free segment is larger than the sixe of the request, the memory space can be split.
   - When returning the memory, merge the returned spaces into a larger one, if possible.

3. ***slab cache per CPU:*** A kernel-object request such as ***task_struct***
   - *Slab cache*: per object-type kernel cache.
   - If *slab* is empty, allocated from the buddy area.
   - When the kernel returns the object, it goes to ***slab cache*** for later reuse.
   - **kmalloc**() is a special slab cache with fixed sizes

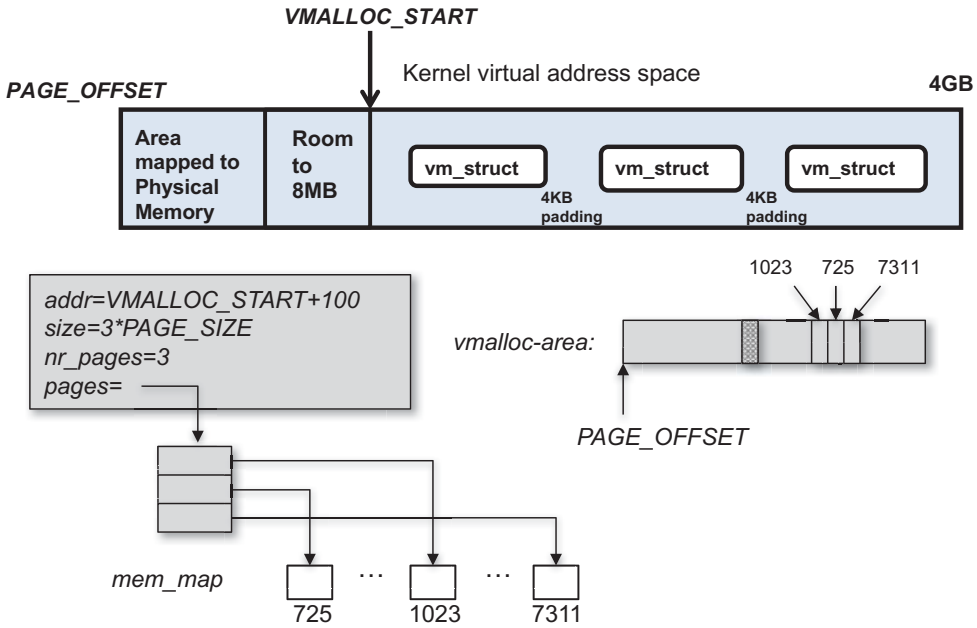
Dynamic Kernel Memory Allocation (3/3)

2. Discontiguous allocation

- vmalloc(): virtually contiguous but not physically
- Synonym of malloc() in the kernel.
- For temporary kernel data.
- The virtual regions are reserved in kernel's virtual address space.
- But in physical memory, pages can be discontiguous.

Discontiguous Allocation

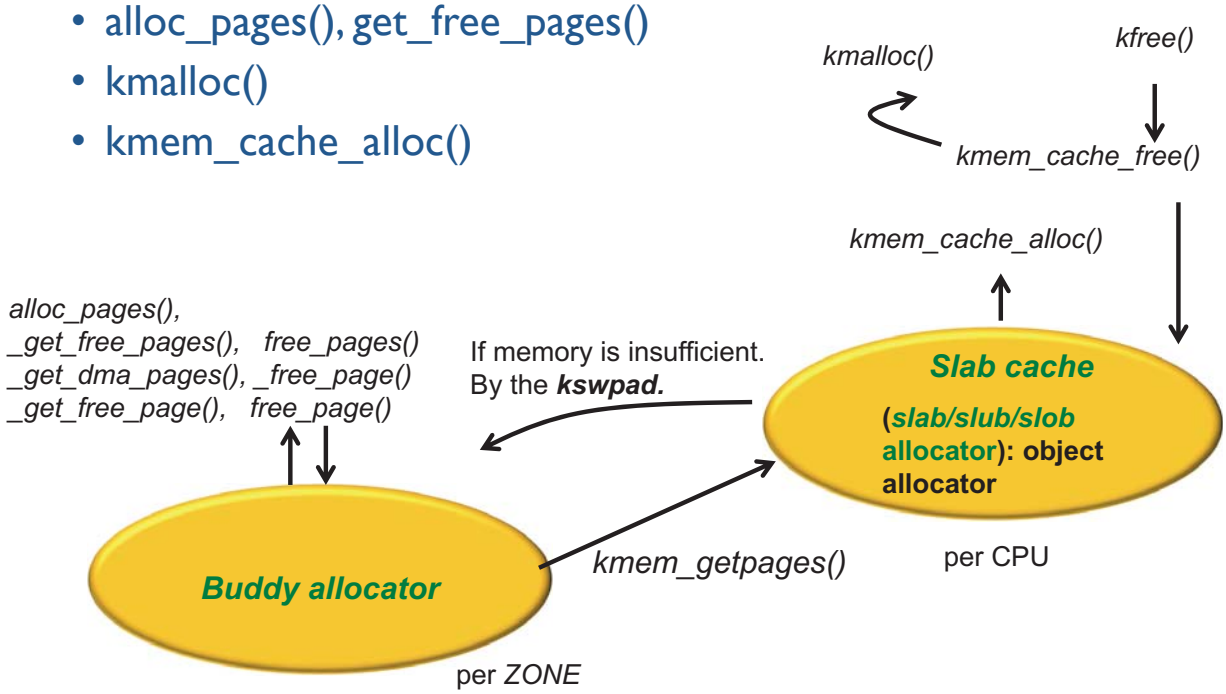
- Discontiguous allocation : virtually contiguous but physically not.
  - vmalloc (size), vfree(size) :when contiguous allocation has been failed, (large chunk of memory)



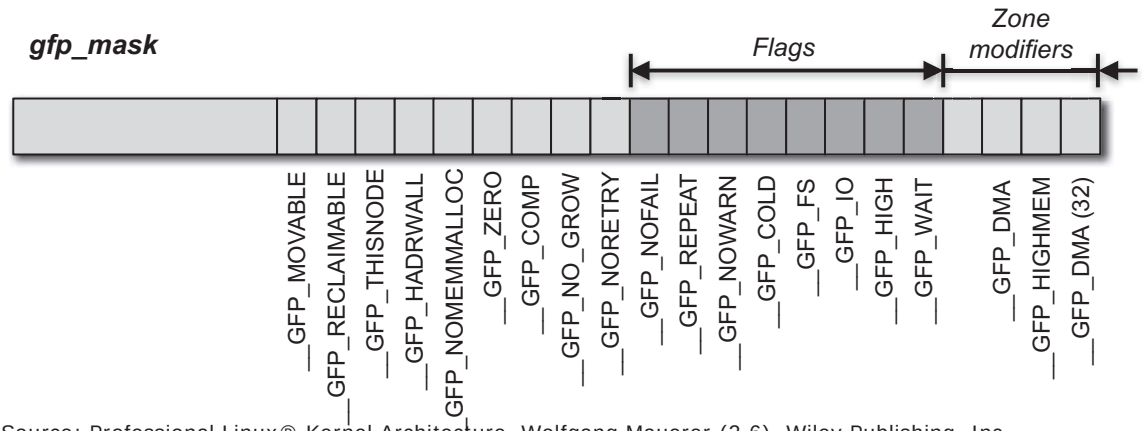
Source: Professional Linux® Kernel Architecture, Wolfgang Mauerer (2.6), Wiley Publishing, Inc.

Contiguous Page Allocation

– Major functions
  - alloc_pages(), get_free_pages()
  - kmalloc()
  - kmem_cache_alloc()



gfp_mask (zone selection & operation type)



Source: Professional Linux® Kernel Architecture, Wolfgang Mauerer (2.6), Wiley Publishing, Inc.

Zone Modifier	Zones scanned for alloc (fall-back order)
Empty	ZONE_NORMAL -> ZONE_DMA
__GFP_DMA	ZONE_DMA
__GFP_DMA & __GFP_HIGHMEM	ZONE_DMA (if DMA is set, this means -> only DMA)
__GFP_HIGHMEM	(ZONE_MOVABLE) -> ZONE_HIGHMEM -> ZONE_NORMAL -> ZONE_DMA

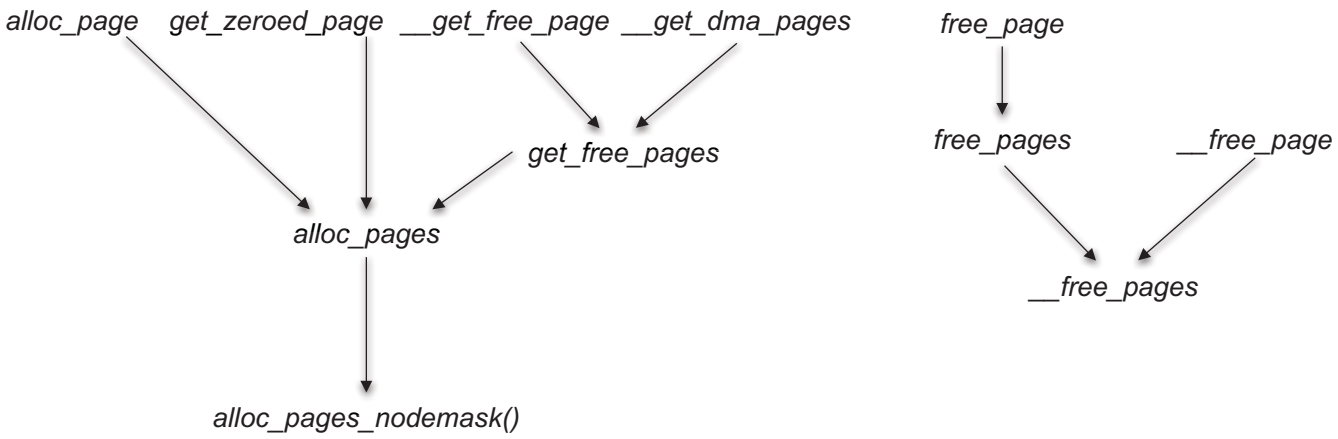
gfp_mask flags

mask flags	Description
__GFP_WAIT	the memory request may be interrupted.
__GFP_HIGH	if the request is very important
__GFP_IO	the kernel can perform I/O operations during an attempt to find fresh memory. (ex. During alloc., can do page reclaim)
__GFP_FS	allows the kernel to perform VFS operations
__GFP_COLD	allocation of a "***cold*** " page that is in the CPU cache is required.
__GFP_NOWARN	suppresses a kernel failure warning if allocation fails.
__GFP_REPEAT	automatically retries a failed allocation but stops after a few attempts.
__GFP_NOFAIL	retries the failed allocation until it succeeds.
__GFP_ZERO	returns a page filled with zero bytes if allocation succeeds.
__GFP_HARDWALL	It limits memory allocation to the nodes associated with the CPUs assigned to a process, NUMA things
__GFP_THISNODE	fallback to other, nodes is not permitted, NUMA things
__GFP_RECLAIMABLE & __GFP_MOVABLE	page mobility mechanism. marks that the allocated memory will be reclaimable or movable, respectively. This influences from which sublist of the freelist the page or pages will be taken.

gfp_mask flag/modifier grouping

Alloc flags	Description
GFP_ATOMIC	*(__GFP_HIGH)*, alloc must be done atomically
GFP_NOIO	*_GFP_WAIT* and during alloc, no disk swap
GFP_NOFS	*_GFP_WAIT | _GFP_IO* and during alloc, no file I/O
GFP_KERNEL	*(__GFP_WAIT | __GFP_IO | __GFP_FS): NORMAL, DMA Zone*
GFP_USER	*(__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_HARDWALL)*
GFP_HIGHUSER	*(__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_HARDWALL | \ __GFP_HIGHMEM)*
GFP_HIGHUSER_ MOVABLE	*(__GFP_WAIT | __GFP_IO | __GFP_FS | \ __GFP_HARDWALL | __GFP_HIGHMEM | \ __GFP_MOVABLE)*

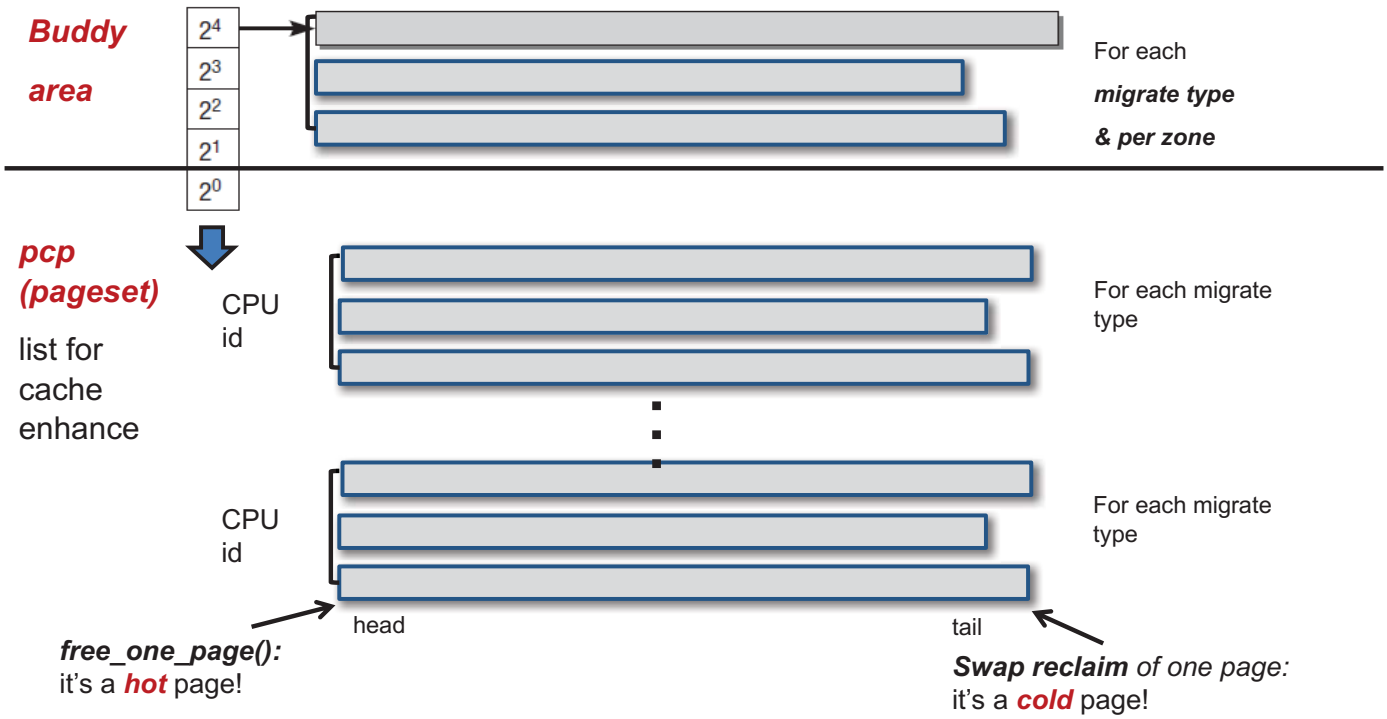
Buddy APIs: func(gfp_mask, order)



Source: Professional Linux® Kernel Architecture, Wolfgang Mauerer (2.6), Wiley Publishing, Inc.

- page_cache_alloc() & page_cache_alloc_cold() are also convenience functions to yield cache-warm and cache-cold pages, respectively, by setting the __GFP_COLD modifier accordingly.

Buddy Area (Free-area) and pcp(per CPU pagelist)



Buddy area & allocator

- Lists of free frames per size (2^{order}).
  - Ex) list of 4KB frames, list of 8KB, 16KB,,, frames, etc.
- Used for kernel memory allocation (contiguous/non-contiguous)
  - Ex) allocations of important kernel data structure such as DMA area and user page tables uses contiguous alloc.
- When returning the frames, adjacent frames are merged into a larger-size frames.

Zone alloc_flag Options in Lower APIs

	Alloc flags	***Description***
Watermark, used in ***zone_water mark_ok()***	*ALLOC_NO_WATERMARKS*	don't check watermarks at all
	ALLOC_WMARK_MIN	use *pages_min* watermark
	ALLOC_WMARK_LOW	use *pages_low* watermark
	ALLOC_WMARK_HIGH	use *pages_high* watermark: **default**
Rules	*ALLOC_HARDER*	try to alloc harder: **use generous rules** (1/4 discount)
	ALLOC_HIGH	*__GFP_HIGH* : urgent alloc: more generous in high-mem (1/2 discount)
NUMA	*ALLOC_CPUSET*	check for correct cpu-set

Fast & important alloc.

- Finally, when an allocable zone (**zone_watermark_ok()==true**) is found,
  - buffered_rmqueue() for the zone is called.

Physical Page Allocation

gfp_mask selects the highest possible zone

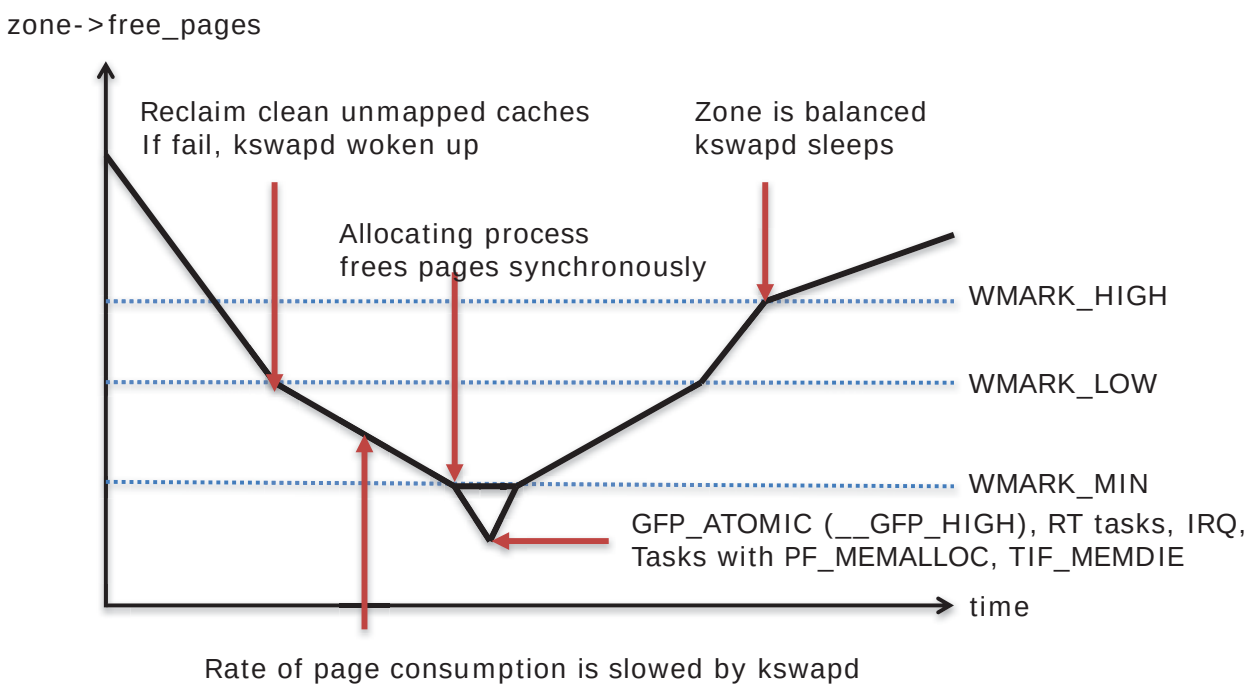
HIGHMEM, NORMAL, DMA

Alloc. ***fall-back*** order, but use ***gfp_mask.***

alloc_pages(order, gfp_mask,,)

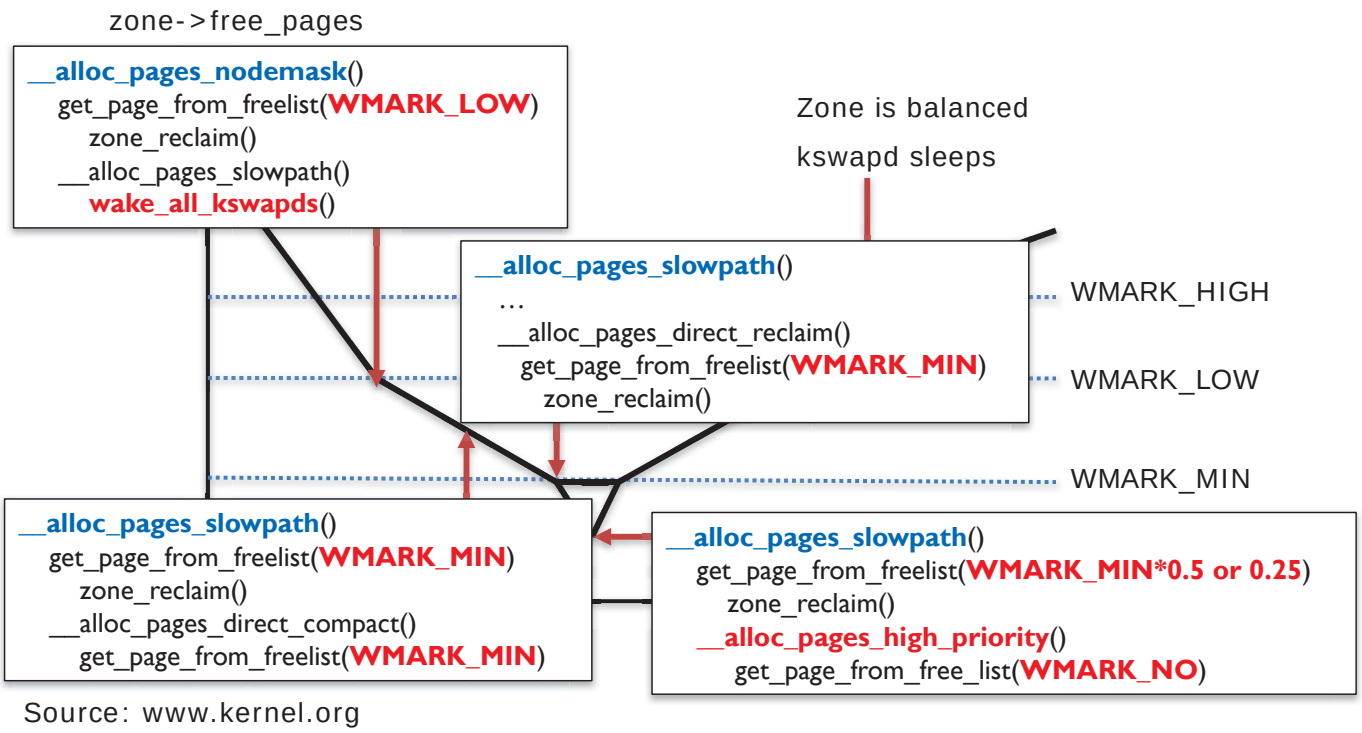
// ***gfp_mask*** masks has the wanted-zones.
// It also has flags on ***fail-sleep, do-I/O-or-not, WATERMARK*** type.
- call ***__alloc_pages_nodemask()*** ***// for a node***
  - ***get_page_from_freelist()*** ***// for all zones by the fallback order***
    : if (free pages < WMARK_LOW) call sync ***zone_reclaim() with I/O;***
    : retry alloc. in this zone. If fail, try the next zone;
  - if (failed in all zones), call ***__alloc_pages_slowpath()***
    : wake up ***async. kswapd*** with writeback I/O;
    : try again with ***get_page_from_freelist****(); with WMARK_MIN*
    : if (failed again in all zones),
      __alloc_pages_high_priority() // do not use ***WMARK***
      // & with a decreased priority => this means an increased ***nr_to_scan***
    : if (failed again and order > 0) ***__alloc_pages_direct_compact();***
      // after zone compaction, try ***get_page_from_freelist();***
    : if (failed again) ***__alloc_pages_direct_reclaim();***
      retry ***get_page_from_freelist()*** *if some progress done*;
    : if (failed again and retry is not possible) ***__alloc_pages_may_oom();***
    return *page* or *no_page*;

Zone Watermarks: Page Reclaim & Zone Balancing



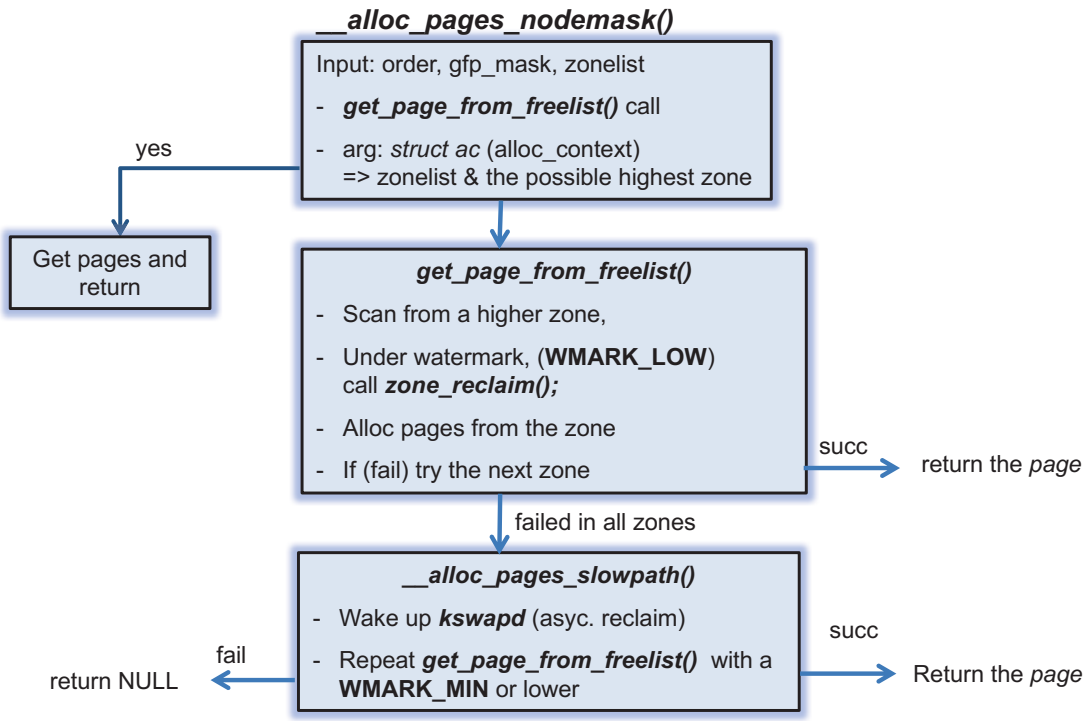
Source: www.kernel.org

Zone Watermarks: Page Reclaim & Zone Balancing

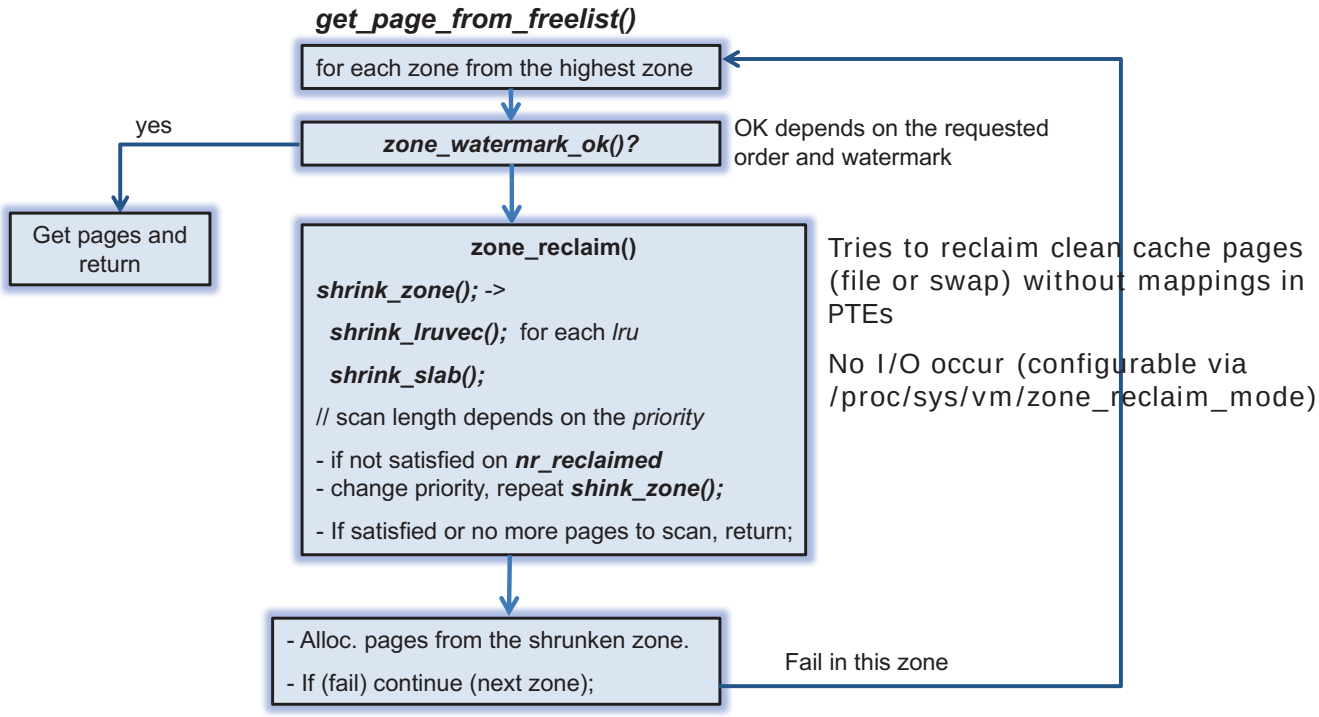


Source: www.kernel.org

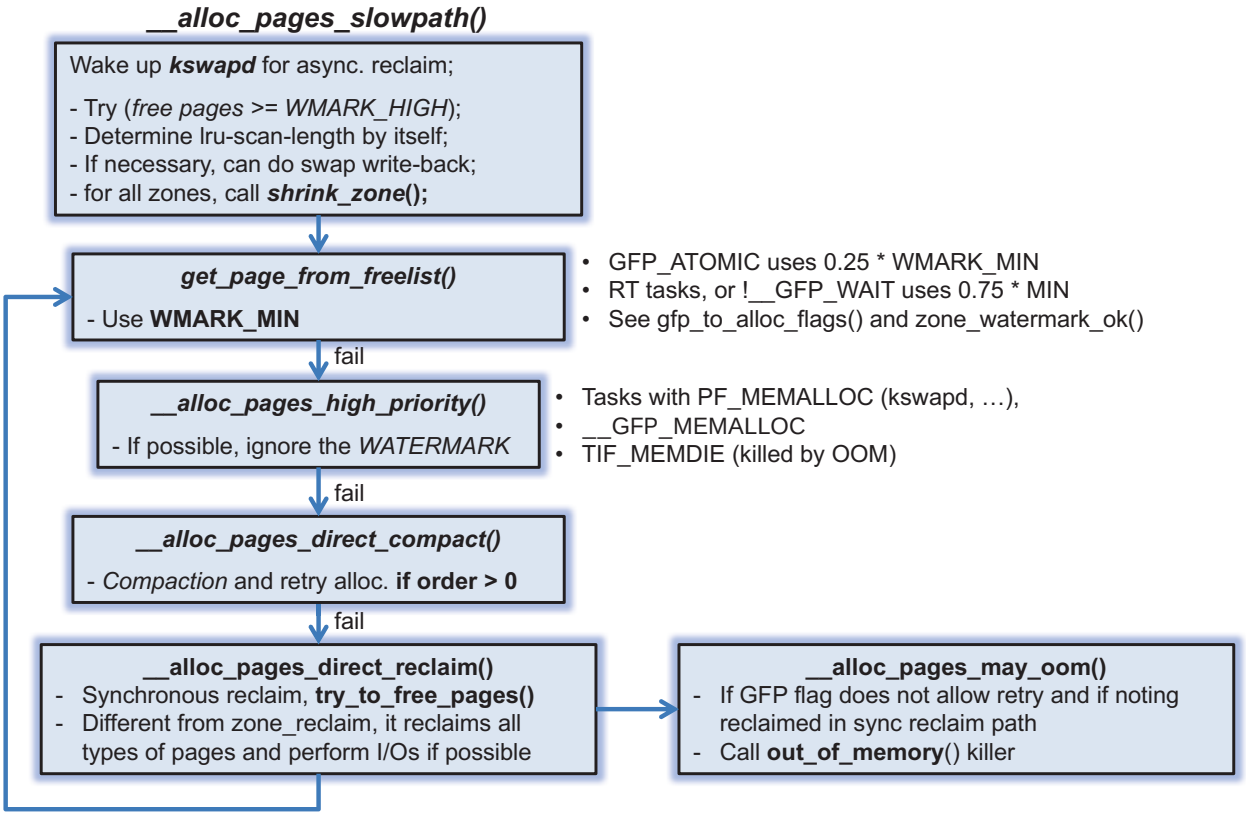
alloc_pages_nodemask()



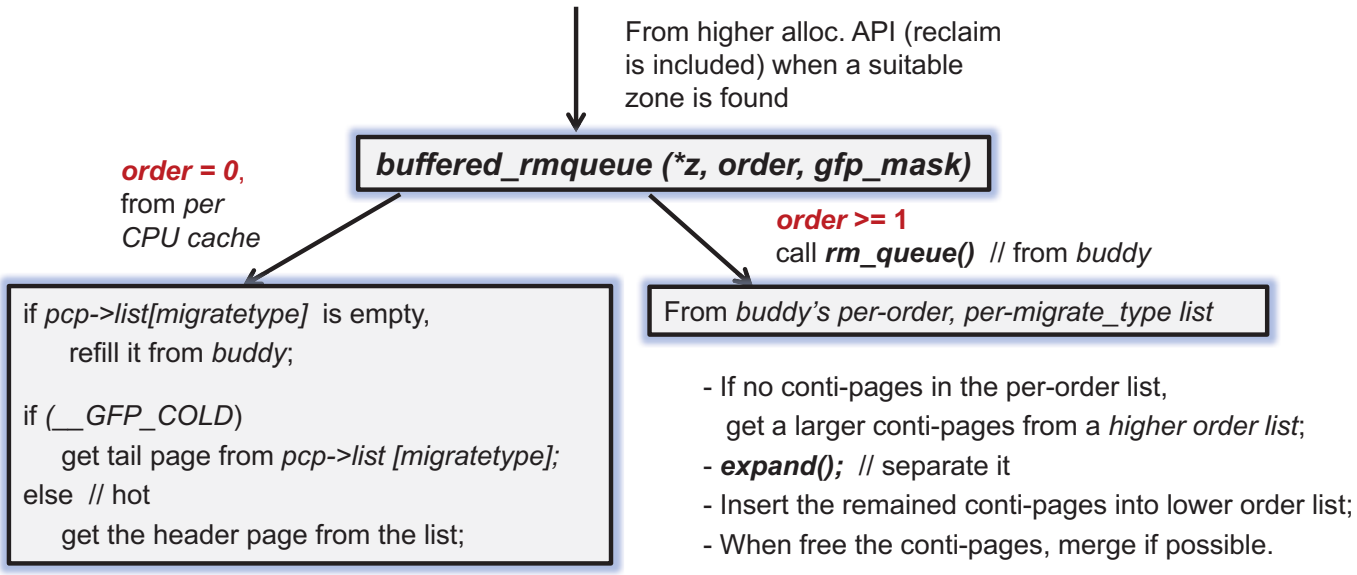
get_page_from_freelist()



__alloc_pages_slowpath()



Lowest level page alloc. (when an allocable zone has been found)



Avoiding Fragmentation

- Grouping pages by mobility
  - Buddy allocator suffers from physical memory fragmentation.
- Page grouping (1st tech.) (called pageblock)

page alloc. request category	Description
Unmovable	Mainly used by kernel core (fixed position), a non-movable page cannot be located in the middle of a block of movable pages
Reclaimable	Cannot be moved directly. However, after reclaim (*kswapd*, sync-reclaim), it can be used for constructing a larger block.
Movable	Always can be moved, other user spaces. When moves, the *pte* is updated.

- High zone partitioning (ZONE_MOVABLE) (2nd tech.: option)
  - A portion of the ZONE_HIGH is assigned to ZONE_MOVABLE (virtual zone).
  - A movable alloc. will be done on ZONE_MOVABLE.

Free Area List of a Zone per Migrate Types

- A zone has a buddy (free_area) list per each migrate type.

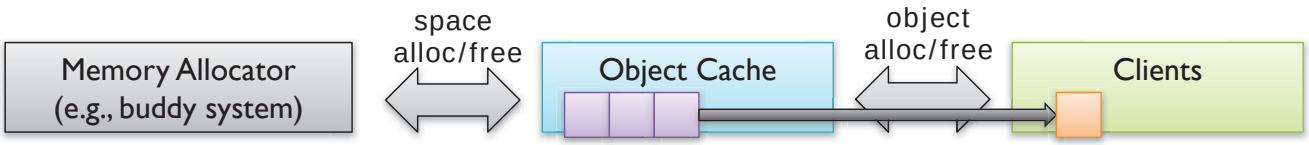
Migrate type	Description	Fallback order when fails
MIGRATE_ UNMOVABLE	*Buddy*'s unmovable free pages	*UNM -> REC->MOV->RES*
MIGRATE_ RECLAIMABLE	*Buddy*'s reclaimable free pages	*REC -> UNM -> MOV -> RES*
MIGRATE_ MOVABLE	*Buddy*'s movable free pages	*MOV -> REC -> UNM -> RES*
MIGRATE_ RESERVE	Initialized for urgent page allocation. Used when alloc. from other mig-types is impossible.	
MIGRATE_ ISOLATE	Space in which pages are isolated for page migration (compaction or NUMA balancing)	

Page Block

- A page block : a set of contiguous free pages.
- Two kernel global variables to arrange free page blocks for each migrate type of a zone.
  - ***pageblock_order*** (2^n): denotes an allocation order that is considered to be "large,". (default: 10)
  - ***pageblock_nr_pages*** (2^n): denotes the corresponding number of pages for this allocation order.
- In page migration for compaction,
  - Try to reserve at least 1 page-block per each migrate type free area.
  - Zone has some management data structures for available page blocks.

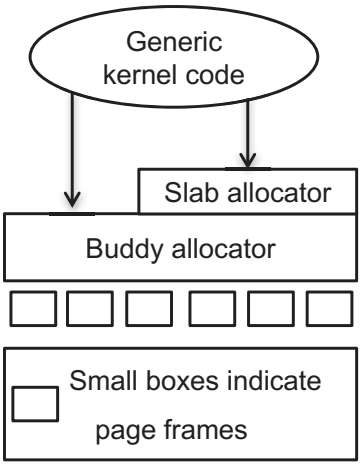
Slab Cache

- If space is needed for a string with 10 characters, using a 4KB page is wasteful. The obvious solution is to split the memory in a page into smaller units that can then hold large numbers of small objects.
- A slab cache is a cache of a type of kernel object. (ex.)
  1. Alloc. a task_struct; // from slab cache of "task_struct" first, it none, fill the slab from buddy area and get it.
  2. Freeing a task_struct; // when the process exits.
     - This returned task_struct to the slab cache of task_struct for further allocation requests to task_struct's.
     - From the next allocation request, use the slab cache for speed up. (caching)
- Slab coloring: different objects → different cache lines



Slab Cache

- APIs
  - Specific slab cache creation/destroy
    - kmem_cache_create()
    - kmem_cache_destroy()
  - Allocating, freeing objects
    - kmem_cache_alloc()
    - kmem_cache_free()
  - General slab caches
    - kmalloc(), kmalloc_node()
  - kmalloc()
    - Predefined set of slab caches
    - kmalloc-8, kmalloc-16, kmalloc-32, …
  - /proc/slabinfo

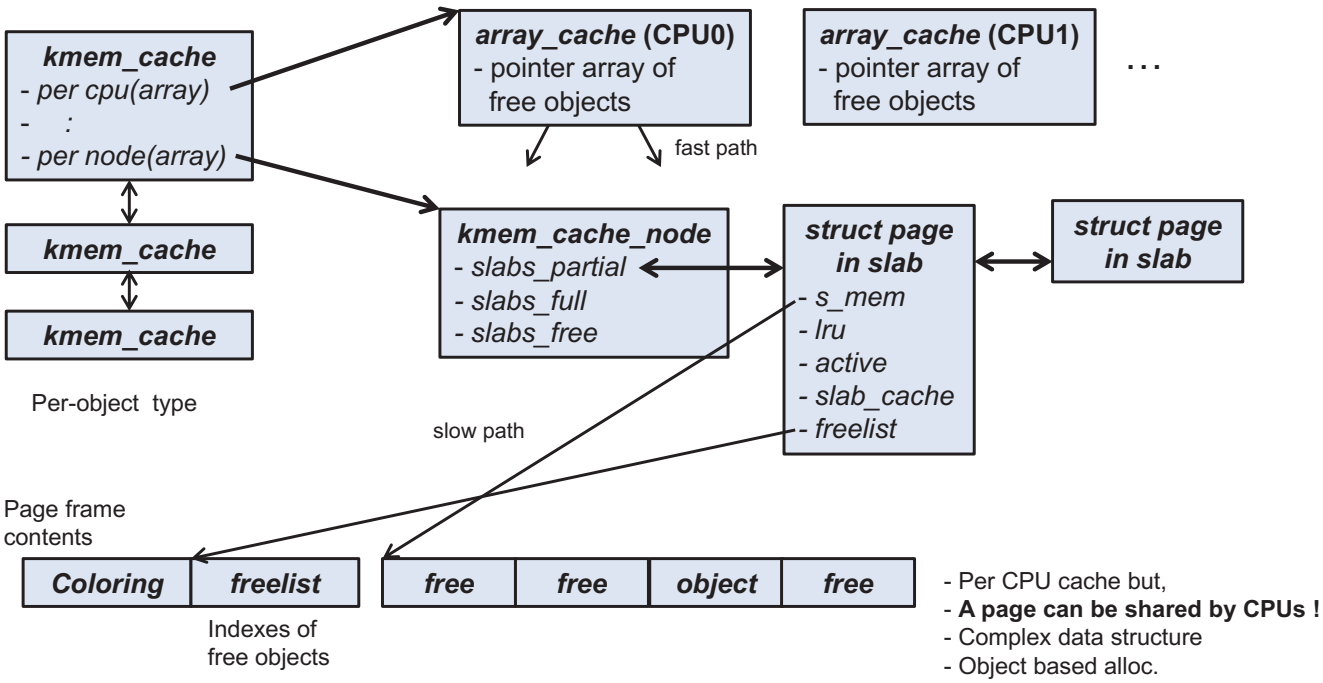


Source: Professional Linux® Kernel Architecture, Wolfgang Mauerer (2.6), Wiley Publishing, Inc.

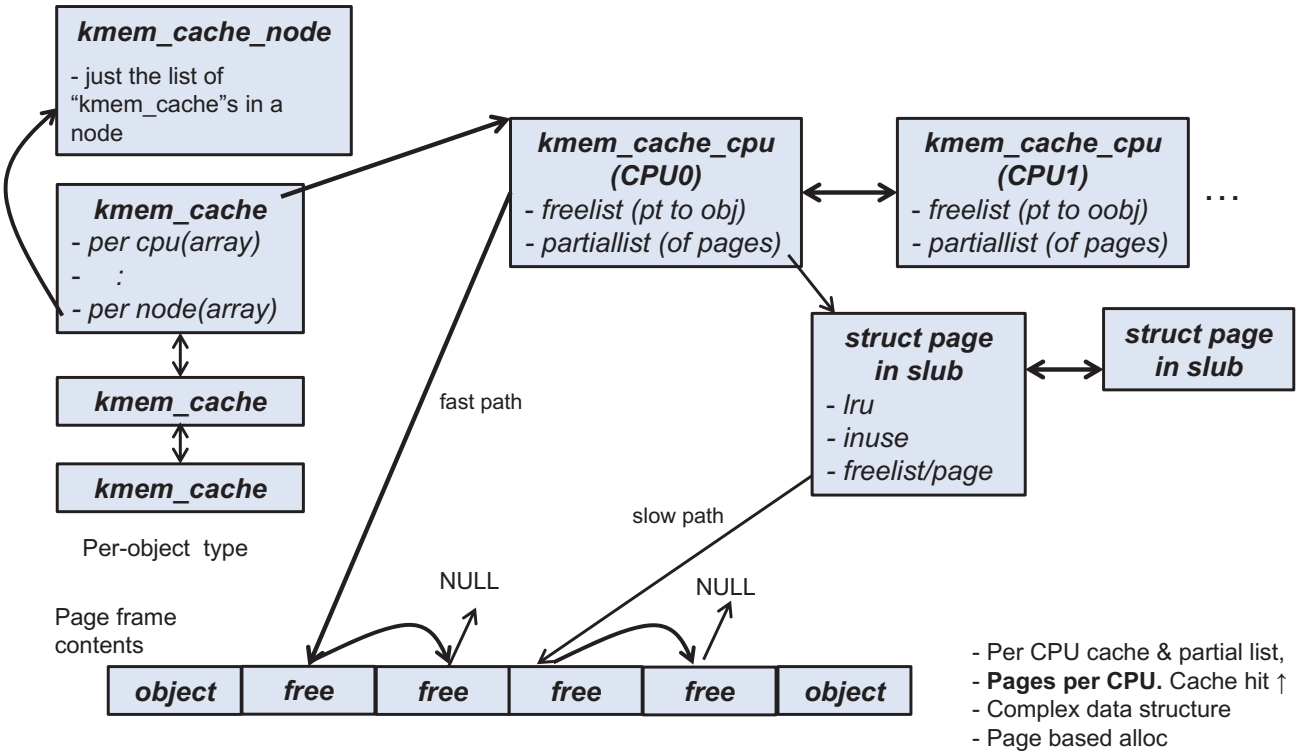
Alternative Allocator for Slab Cache

- The three object allocators providing same API
  - The **SLAB** allocator: traditional allocator
  - The **SLUB** allocator: default, for large systems
    - Considering the per-CPU cache in SMP.
    - Better performance of slub in contrast to slab on large machines.
  - The **SLOB** allocator: for embedded system
    - is especially optimized for low code size. It is centered around a simple linked lists of blocks (thus its name). To allocate memory, a likewise simple **first-fit algorithm** is used (no slab coloring).

Slab Allocator



Slub Allocator (Default)



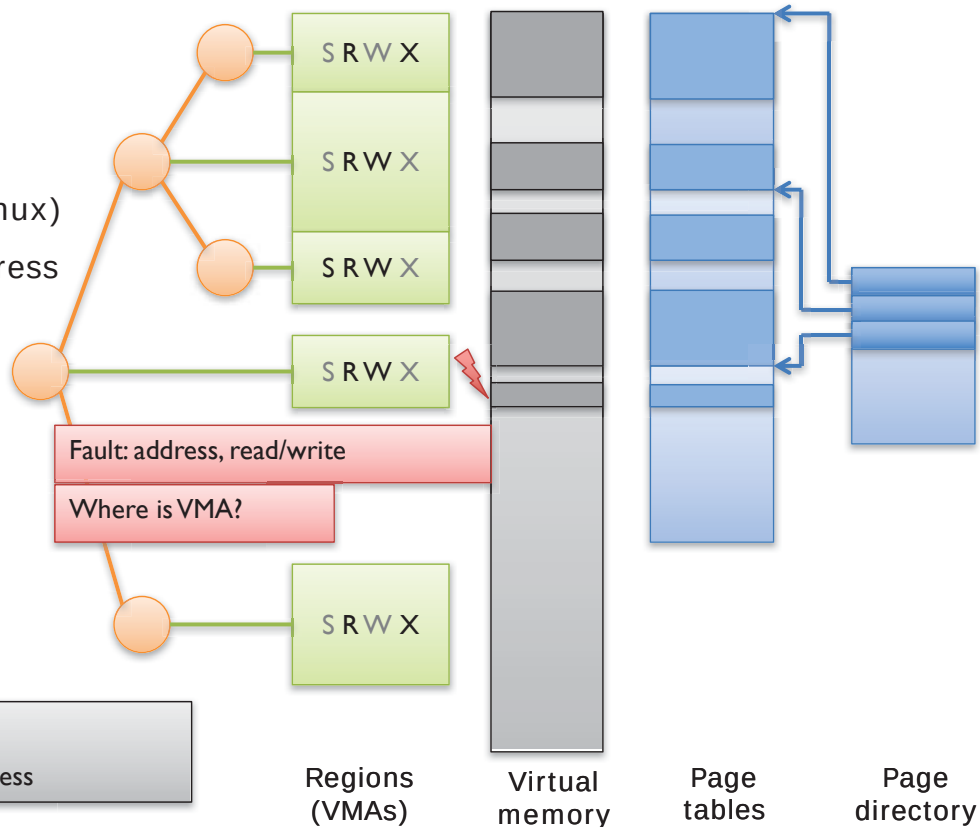
SLOB allocator

- SLOB allocator
  - Global kmem_cache for all types of slab caches.
  - A page can have multiple type of objects.
  - Access: kmem_cache => page => page's freelist
  - First-fit based.
  - Small code size.
  - Suitable to embedded systems.

Demand Paging

Demand Paging Overview

- VMA search tree (red-black tree in Linux)
  - Indexed by address



If no VMA found,

SEGFAULT signal to a process

Source: A. T. Clements, et al., Scalable Address Spaces Using RCU Balanced Trees, ASPLOS 12

Demand Paging (1/3)

- Fetch a page only when the page fault on it occurred as a result of a reference !
  - To save the memory space.
  - To reduce the total loading time.
- Fault-page loading
  1. File-backed page.
     - ***Try page cache first.*** If not found do file I/O.
  2. Swap-out page.
     - ***Try swap cache first.*** If not found do swap I/O.
  3. Anonymous page (swap-backed but not loaded yet. ***bss, heap, stack***)
     - Use a ***zero page*** & CoW handling.

Demand Paging (2/3)

- Use read-ahead to reduce the further page faults.
  - Major fault (accompanying I/Os)
    - swapin_nr_pages() for swapping (max 2^3 pages).
    - Max 32 pages in filemap_fault()
  - Minor fault
    - fault_around_bytes (default is 64KB)
- When a page is prepared by demand paging,
  1. The ***struct page of a page*** that is newly fetched from of a file is inserted into the ***page cache***.
  2. The ***struct page of a page*** that is newly fetched from of a swap dev. is inserted into the ***swap cache.***
  3. The ***struct page*** of an anonymous zero page will have ***anon_vma*** structure.

Demand Paging (3/3)

- Cooperation with the swapping system.
  1. The ***page*** that is newly fetched from of a file will be inserted into the ***ACTIVE_FILE*** or ***INACTIVE_FILE LRU***. (mainly to ***INACTIVE_FILE***).
     - Why mainly to ***INACTIVE_FILE?.*** → 2Q (not accessed pages evict quickly)
  2. The ***page*** that is newly fetched from of a swap dev or newly allocated will be inserted into ***ACTIVE_ANON LRU***. (this fault is reference to the page)
  3. Readahead (or fault_around) pages are inserted into ***INACTIVE_ANON LRU***.

Important pte Flags/Fields (revisited)

pte field/flag	***Description***	
pte_present()	True if the target frame is in memory. A not-present *pte* causes a page fault. The page may be got from page cache or swap cache.	
pte_none()	The *pte* has nothing. When fault by not-present, this page is an anon page (bss, heap, stack) or a file page (text, data).	
swap-entry / frame	***frame #***	Physical page address if a present page
	swap-entry	Swap info. : not *present* but not *none*
PROT	*WRITE / READ / EXECUTE* Clean data/anon pages are set to *NO-WRITE* to raise protection faults for *COW* handling. (data => copy, anon =>zero page split)	
ACCESSED	Set by *MMU* when a page is accessed.	
DIRTY	Set by *MMU* when a page is write-accessed.	

- ***ACCESSED***: when determine a reclaimable page, it is tested and cleared by reverse mapping
- ***DIRTY***: when swap-out write-back, this flag is copied to page and cleared. (to decide to do write-back or not)

struct pages in Kernel Caches

- Page cache
  - struct page of a file-backed page belongs to the page cache (a radix-tree of pages) for VM.
  - This page can have one or more pte's pointing to this page, (used, may shared)
  - or no pte at all. (ex. Currently not used but kept in cache for reuse.)
  - If a process triggers a page fault (pte_none(), not pte_present()),
    - The kernel tries the page cache first to reduce disk I/O.
    - If (not found in the page cache), fetch the page into the page cache by a disk I/O.
  - A page without pte mapping can be in the page cache. (process exited)

struct pages in Kernel Caches

- Page cache is also used for VFS (file-system I/O files with 4K physical blocks)
  - Fetch: by read (do_generic_file_read()): not by page fault.
  - Page cache write-back of "delayed-write" pages.
    - Dirty pages are written-back to backing storage in background
      - Older versions: pdflush (kernel daemon) do synch. with disk (periodically)
      - In 4.x: flush-x kworker do sync. (refer to balance_dirty_pages())
      - If a process generates too many dirty pages, the process is paused (by calling io_schedule_timeout())
    - or "fsync" to specific files
    - or by Shell's "sync" command.

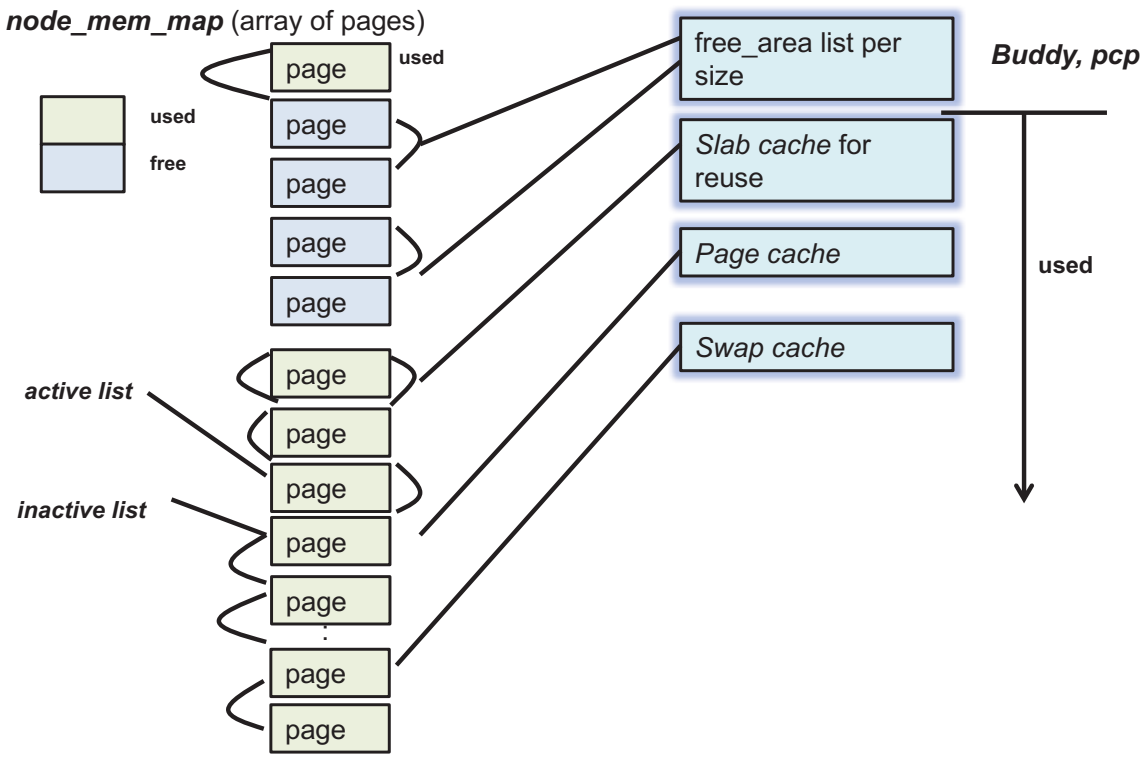
struct pages in Kernel Caches

- Buffer cache for the file system
  - In case that the I/O block size is smaller than the normal page size, the old **buffer cache** is used by VFS.
  - Caching with **buffer headers**.
  - Actual data are contained in a page that is assigned to VFS buffer.
  - So a page can have several I/O blocks.
  - This pages are also inserted into the page cache.

struct pages in Kernel Caches

- Swap cache
  - A swapped-in page from a swap device belongs to the swap cache.
  - If another process triggers a page fault on this page, the page in swap cache can mapped without I/O.
  - When the wanted page is found in swap cache,
    - If write access & a single mapped-pte (not shared), this is a dirty data, heap or stack.
    - In this case, swap caching is meaningless, so remove it from swap cache.
- A page neither in the page cache nor in the swap cache (examples)
  - Dirty data pages before swap out.
  - Clean anon pages (actually a shared zero page).

Pages & Caches

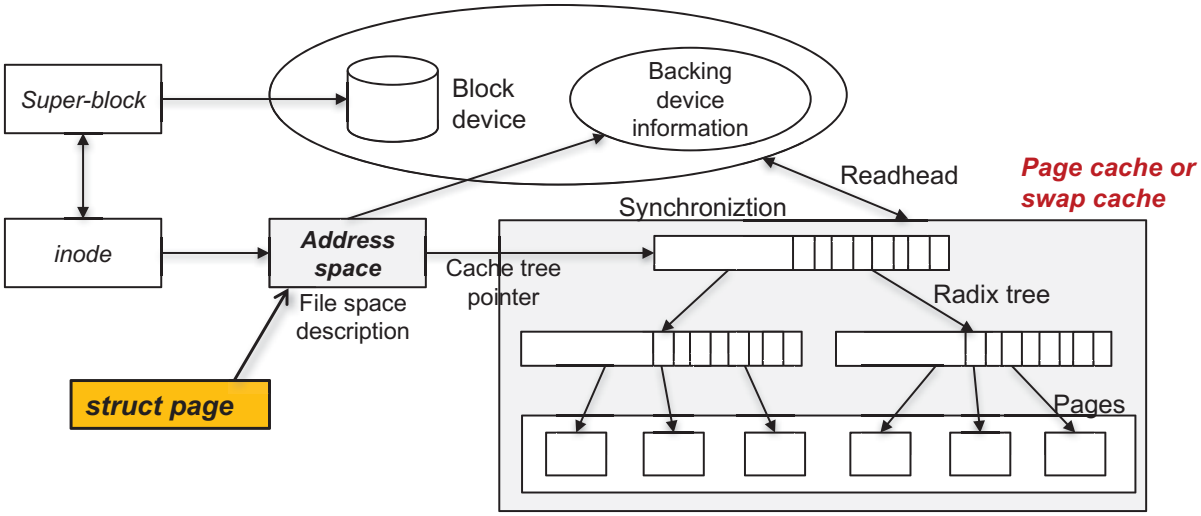


struct page (revisited)

struct page	Description		
u long flags	Atomic flags, some possibly updated asynchronously lock, **referenced, dirty, file-backed / swap-backed, lru, swapcache, active**, etc.		
struct address_space *mapping	File-backed	***address_space*** (for page cache) & **vma_interval_tree** in address_space for reverse mapping)	forma file (text, clean data)
	Swap-backed	***anon_vma*** for reverse mapping page_mapping() = ***swap address_space***	from a swap dev.
	Swap-backed, but not swapped out yet = anonymous pages	***anon_vma*** for reverse mapping	BSS, heap, stack, dirty data
	Free	NULL	
pgoff_t index	Our offset within mapping (eg. if file-backed, the offset in the file)		
atomic_t _map_count	number of ptes (that share this page), start at -1. If it is in swap_cache or in page-cache, 0		
atomic_t _count	Reference counting		
struct list_head lru	Have ***next & *prev**, used for ***INACTIVE_ANON, ACTIVE_ANON, INACTIVE_FILE, ACTIVE_FILE, UNEVICTABLE*** lru lists.		
u long private	Stores private data. Swap-cache page has **swap_entry** (block number in swap device)		
others	Information for managing Buddy/Slab-cache, mem-cgroup, kernel debugging		

Caches & Address Space of a File/Swap

- A struct page can have a mapping as follows:
  - File address space (page cache): text, clean data
  - Swap address space (swap cache): any page swapped in/out
  - Anon map (anon_vma chain): dirty anon, dirty data before swap out
  - Nothing: zero anon



Source: Professional Linux® Kernel Architecture, Wolfgang Mauerer (2.6), Wiley Publishing, Inc.

struct address_space (file, swap)

- ***struct page (file-backed or swap-backed) has a pointer to struct address_space.***

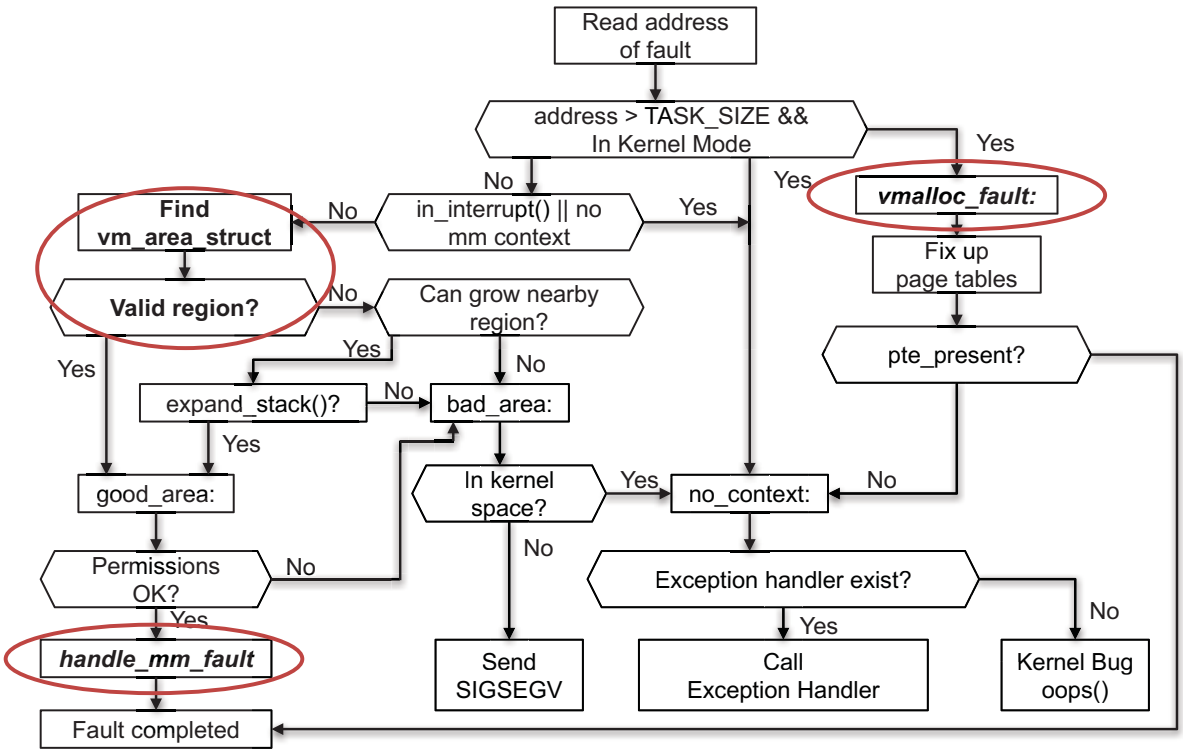
```
struct address_space {
    struct inode            *host;          /* owner: inode, block_device */
    struct radix_tree_root  page_tree;      /* radix tree of all pages */ page or swap cache
    ::
    struct rb_root  i_mmap;         /* tree of private and shared mappings to pte's*/
    ::
    unsigned long           nrpages;        /* number of total pages */
    unsigned long           nrshadows;/* number of shadow entries (later)*/
    ::
    const struct address_space_operations *a_ops; /* methods for I/O*/
    ::
}
```

struct anon_vma for anon pages

- ***struct page of an anonymous page has a pointer to struct anon_vma.***

```
struct anon_vma {
    struct anon_vma *root;        /* Root of this anon_vma tree of a task */
    ::
    struct anon_vma *parent;      /* Parent of this anon_vma */
    struct rb_root rb_root;  /* Interval tree of private "related" vmas, */
                                 // for reverse mapping
};
```


do_page_fault()

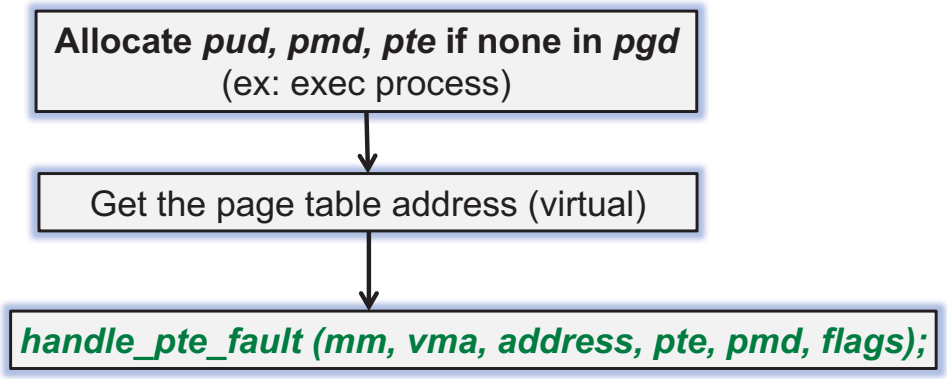


Page Fault Handling

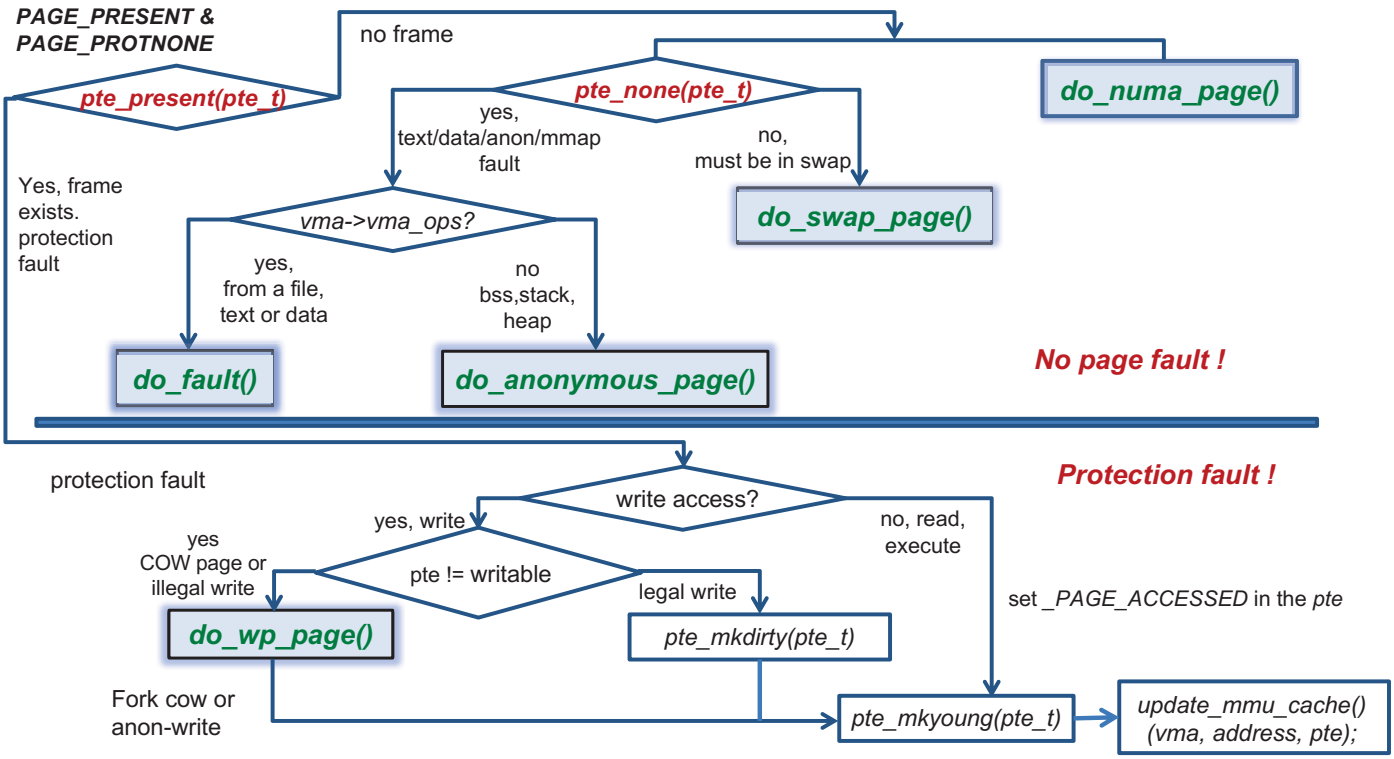
- Two types of page faults → handle_mm_fault()
  - Major fault: occurs when a page has to be read from disk, (swap, file)
  - Minor fault: disk I/O is unnecessary
    - Anonymous: Region valid, but no page frame: just allocate a page frame (bss, heap, stack: zero page)
    - Anonymous: Region extension, extend the region and allocate a page
    - File, swap-backed: Found in page cache/swap cache
    - Write on a read-only page
      - If the page is a COW page, make a cop of it, mark it writable, and assign it to the process.
      - else, SIGSEGV
    - Kernel's page fault: only in vmalloc area (the first access to the allocated page)
    - Error: exception handler
      - in kernel mode, access user space: kernel's bug
      - An user access to an invalid area: SIGSEGV

handle_mm_fault(): 4.x

- __handle_mm_fault();



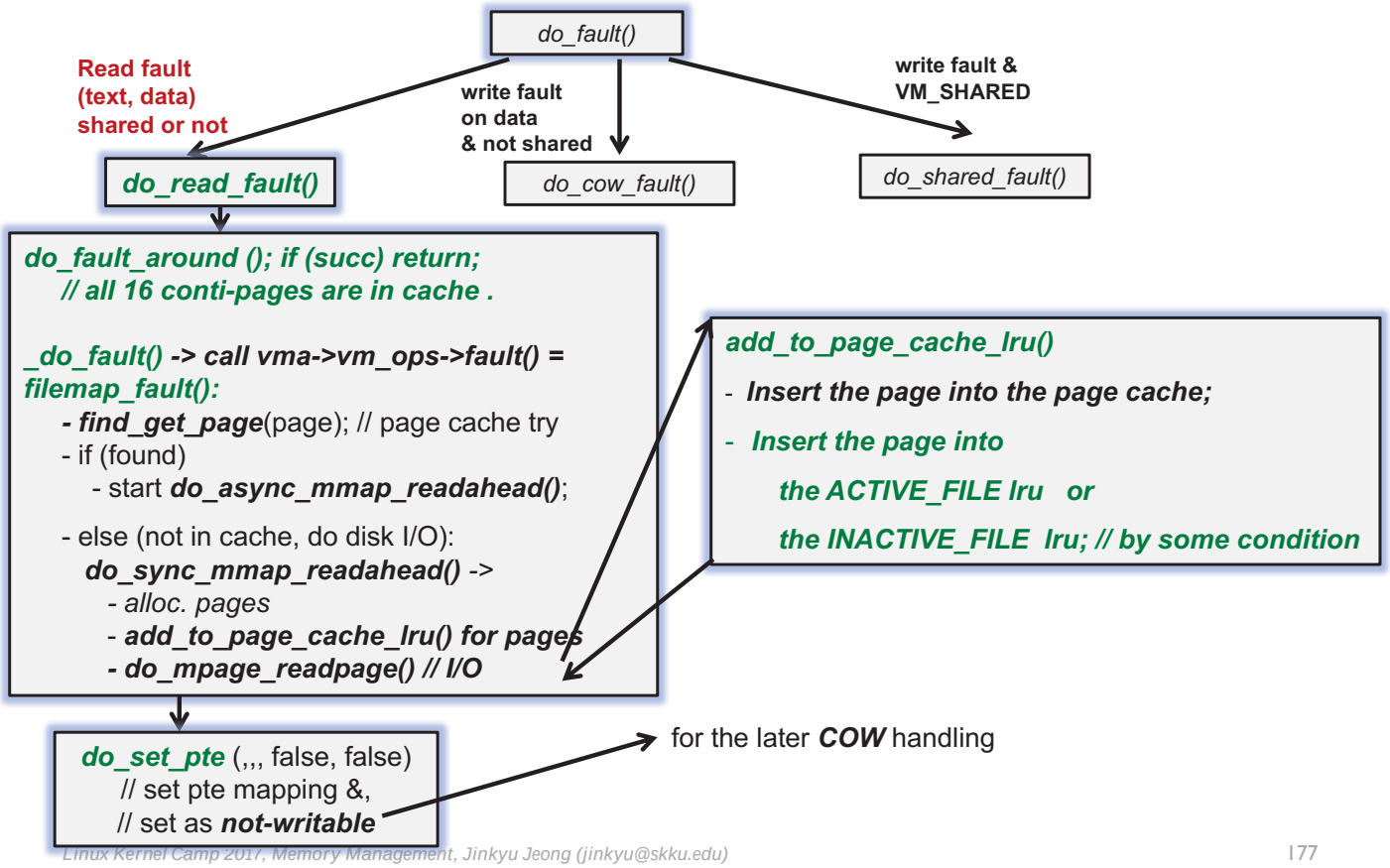
handle_pte_fault(): 4.x



do_fault(): 4.x, memory.c

```
static int do_fault (struct mm_struct *mm, struct vm_area_struct *vma,
        unsigned long address, pte_t *page_table, pmd_t *pmd, unsigned int flags, pte_t orig_pte)
{
    pgoff_t pgoff = (((address & PAGE_MASK)
      - vma->vm_start) >> PAGE_SHIFT) + vma->vm_pgoff;  // to find pte.
    pte_unmap(page_table);  // When this pte/page-table is in high-mem, this pte was accessed by the
                // kernel by using kmap area mapping temporarily.
                // Now, no more access is necessary, so unmap it.
                // when page table access is necessary, kernel calls pte_offset_map_lock()

    if (!(flags & FAULT_FLAG_WRITE))  // read fault
        return do_read_fault (mm, vma, address, pmd, pgoff, flags, orig_pte);
    if (!(vma->vm_flags & VM_SHARED))  // write fault and not shared memory (private mapping)
        return do_cow_fault( mm, vma, address, pmd, pgoff, flags, orig_pte);
                // early COW handling: (explained later)
    return do_shared_fault (mm, vma, address, pmd, pgoff, flags, orig_pte);
                // fault on VM_SHARED area
}
```

do_fault() for file-mapped no-page



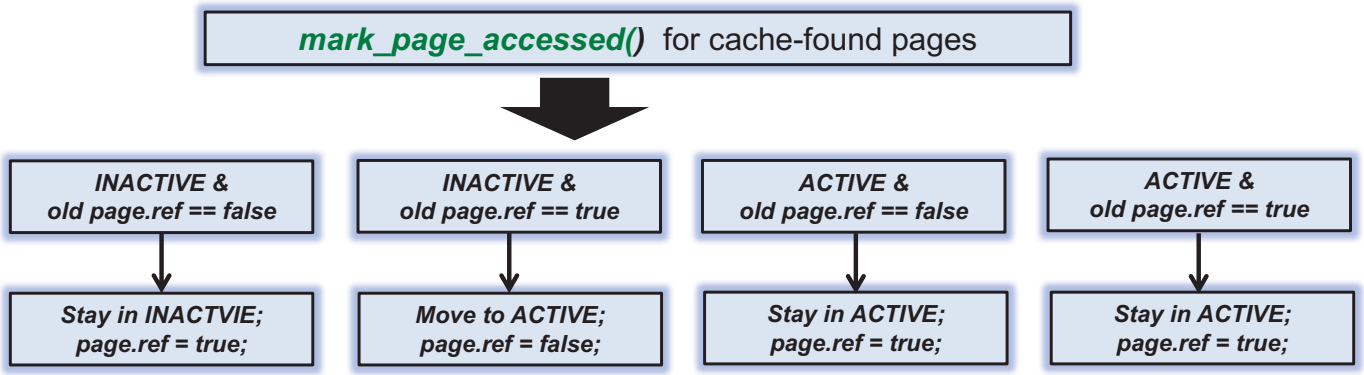
do_fault_around()

- in *do_read_fault()*, // for text & data
  - ***do_fault_around()*** is called first before calling ***vma->vm_ops->fault()***.
  - To map neighbor pages (64KB) of the fault pages (including the fault page), if they are in the page cache.
  - Set pte's;
  - Return;
  - For reducing page faults.

add_to_page_cache_lru()

```
int add_to_page_cache_lru (struct page *page, struct address_space *mapping,
                           pgoff_t offset, gfp_t gfp_mask)
{
    void *shadow = NULL;     int ret;
    __set_page_locked(page);
    ret = __add_to_page_cache_locked(page, mapping, offset, gfp_mask, &shadow); // to page cache

    if (shadow && workingset_refault(shadow)) {  // refault distance <= workingset size
     SetPageActive(page); insert into active list (pagevec actually)
     workingset_activation(page);  // page_zone(page)-> inactive_age++
                                   // num of pages goes to active list into inactive list
    } else
     ClearPageActive(page);

    lru_cache_add(page); insert to ACTIVE_FILE or INACTIVE_FILE lru

    return ret;
}
```


mark_page_accessed()

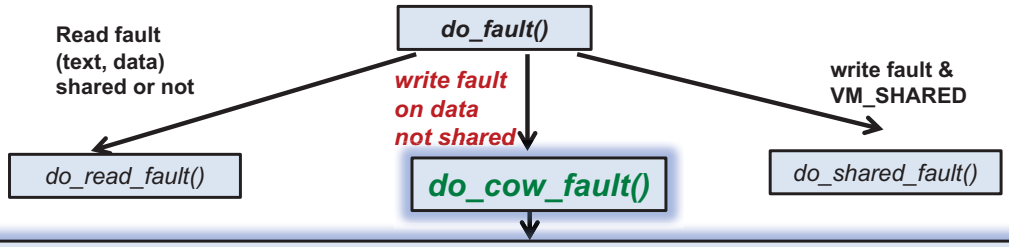


- ***page.ref*** flag is used only for text and clean data pages at page reclaim. (explained in swapping)
- ***page.ref*** will be cleared at page reclaim again.
- ***page.ref*** of a file-fetched page is cleared, because it's a new comer.
- ***page.ref* is totally different from the *ACCESSED* flag of a pte !**

Text & Data Read Fault Summary

- Text & data read fault
  - Case: In the *page cache* and *up_to_date*
    - data page: not yet anon (when write occurs, it becomes an *anon page*)
    - start ***async-prepaging (async-readahead)***
    - set *pte* (no dirty, no anon, not writeable) : for later *COW* handling for data page
  - Case: A new-fetch from a file
    - do ***sync-page-readahead*** (with *prepaging*), & put into the *page cache*;
    - By regarding the ***refault-distance***, put into active or inactive list
    - Clear ***page.ref***; (new comer to the active or *inactive-list*)
    - set *pte* (**not writable**, no dirty, no anon) : for the later *COW* handling for the data page

do_cow_fault() for file-mapped no page



1. *new_page* = ***alloc_page_vma()*** // alloc. a free page for **early *COW* handling;**
2. ***__do_fault*** *(,,, &fault_page);* // fault processing on the original page (fault page)

   // the same routine used in the read fault handling (fault-page -> page cache,
   // read_ahread…)
3. ***copy_user_highpage*** *(new_page, fault_page, address, vma);* // data copy

   // copy the fault page into the *new_page* whether it is shared or not (to utilize page cache)

   // the *new_page* is an *anon* page now. // *page.ref* is not necessary!
4. ***do_set_pte****(,,new_page,* **true, true**,) // pte = **writable & dirty** and anon-map from now on
6. ***lru_cache_add_active_or_unevictable*** *(new_page, vma);*

   // insert the *new_page* into the ***ACTIVE_ANON lru;***

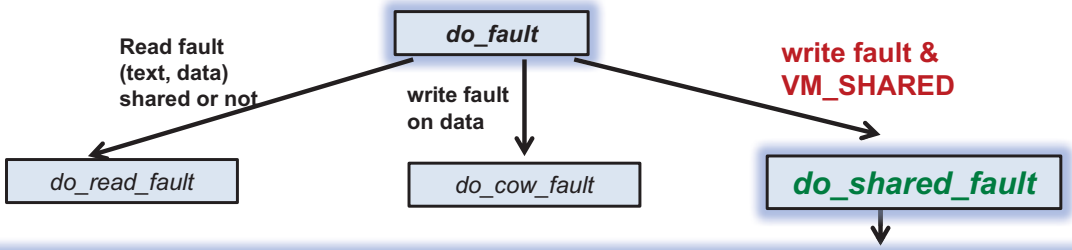
do_set_pte() for file-backed & anon pages

```
// do_set_pte - setup new PTE entry for given page and add reverse page mapping & anon_map.
// @write: true, if new entry is writable, @anon: true, if it's anonymous page
 void do_set_pte (struct vm_area_struct *vma, unsigned long address,
                  struct page *page, pte_t *pte, bool write, bool anon)
{   pte_t entry;
    ::
    entry = mk_pte (page, vma->vm_page_prot);
    if (write)  entry = maybe_mkwrite (pte_mkdirty(entry), vma);  // pte <= dirty & writable
    if (anon) {   // anon
        ::
        page_add_new_anon_rmap (page, vma, address);
                                  // set swap-backed, make anon_vma links & reverse mapping
    } else {    // file-backed page
        ::
        page_add_file_rmap (page);  // set file-backed & reverse mapping
    }
    set_pte_at(vma->vm_mm, address, pte, entry);
    update_mmu_cache (vma, address, pte);
}
```


Data Page Write Fault Summary

- Data-page write fault (first (write) fault on a data page)
  - Whether this page is shared or not, two copies of the data page are created for reuse of the cached original page in later faults.
  - original copy => *fault_page*
    - Clean data page in the *page cache* ("*active/inactive* & *page.ref*" are handled in the same manner of the read-fault)
  - actual copy for the process => *new_page*
    - Becomes an anon page.
    - Goes to active list.
    - *page.ref* is not necessary any more. (this is an *anon* now)
    - *set_pte*: pte mapping set, make dirty,
      set ***swap-backed***, make anon-map (including reverse map) for page->mapping
    - Neither in page cache nor in the *swap cache* yet.
    - The new page -> ***ACTIVE_ANON*** lru.

do_shared_fault() for file-mapped no page



1. ***__do_fault*** (,,, &fault_page); // get page from file or pagecache, mem_mapped file case
   page alloc, insert into page cache;
   insert the page into active or inactive lru list;
2. ***do_set_pte*** *(,,fault_page,* ***writable/dirty, no_anon****)* // set pte and inc page->_map_count
   // set reverse mapping
3. ***set_page_dirty*** *(fault_page);* // page dirty
4. ***balance_dirty_pages_ratelimited*** *(mapping);*

   // This page becomes a dirty page. If the cache has too many *dirty* pages,

   // do file system write-backs. (*mem-map* file)

do_anonymous_page() for bss/heap/stack

do_anonymous_page *(struct mm_struct *mm, struct vm_area_struct *vma, unsigned long address, pte_t *page_table, pmd_t *pmd, unsigned int flags)*

1. ***If (read access && shared zero page is possible), // share global zero-page and set COW***

   entry = ***pte_mkspecial*** *(pfn_pte(my_zero_pfn(address), vma->vm_page_prot));*

   // entry is a pte, **set as not writable for COW**

   set_pte_at *(mm, address, page_table, entry);*

   update_mmu_cache (vma, address, page_table); // x86, nothing
2. ***Write access or (read access but zero-page sharing is not possible) // map to a new zero page***

   - ***anon_vma_prepare*** *(vma);* // if no anon-vma, create & attach
   - *page =* ***alloc_zeroed_user_highpage_movable*** *(vma, address);* // alloc a new page writable;
   - *__SetPageUptodate(page);*
   - *entry =* ***mk_pte*** *(page, vma->vm_page_prot);* & - ***pte_mkwrite(pte_mkdirty(entry));*** **// writable & dirty**
   - ***page_add_new_anon_rmap*** *(page, vma, address);* // anon map and rmap for page
   - ***lru_cache_add_active_or_unevictable*** *(page, vma);* // to ***ACTIVE_ANON lru***
   - ***set_pte_at*** *(mm, address, page_table, entry);* & ***update_mmu_cache*** *(vma, address, page_table);*

Anonymous Page Fault Summary

- Anonymous page fault (first fault on a bss / stack / heap page)
  - Read access
    - map to a process's *shared zero-page.*
    - Set *pte* as special : not writable, set *COW.*
  - Write access or (read & zero page sharing is forbidden)
    - Alloc. a new zeroed page, make the *page/pte* dirty,
    - Make the *pte* map,
    - Add a page *anon-map*,
    - *page->mapping* : *anon_vma,*
    - Insert into *ACTIVE_ANON lru*

do_swap_page() for swapped-out pages

do_swap_page ()

```
get swap_entry (swap info.) from the pte;     // swap dev. & offset
page = lookup_swap_cache (entry);
else if (not found) {
    page = swapin_readahead (entry,, vma, address);
        //  async I/O and put into swap cache, do async read-ahead
        // to ACTIVE_ANON
}
// now, the page has been prepared from the swap cache or swap device.
```

read fault

```
set_pte_at();
do_page_add_anon_rmap();
  // if no anon_vma, make it
  // & set reverse mapping
swap_free (entry);
  // free the swap_entry of the
  // pte
```

write fault, & *map_count* =1

```
set PageDirty();
remove the page from the swap cache;
 // if not shared, it is dirty, so it is
 // meaningless to keep it in cache.
set_pte_at();
do_page_add_anon_rmap();
swap_free (entry);
```

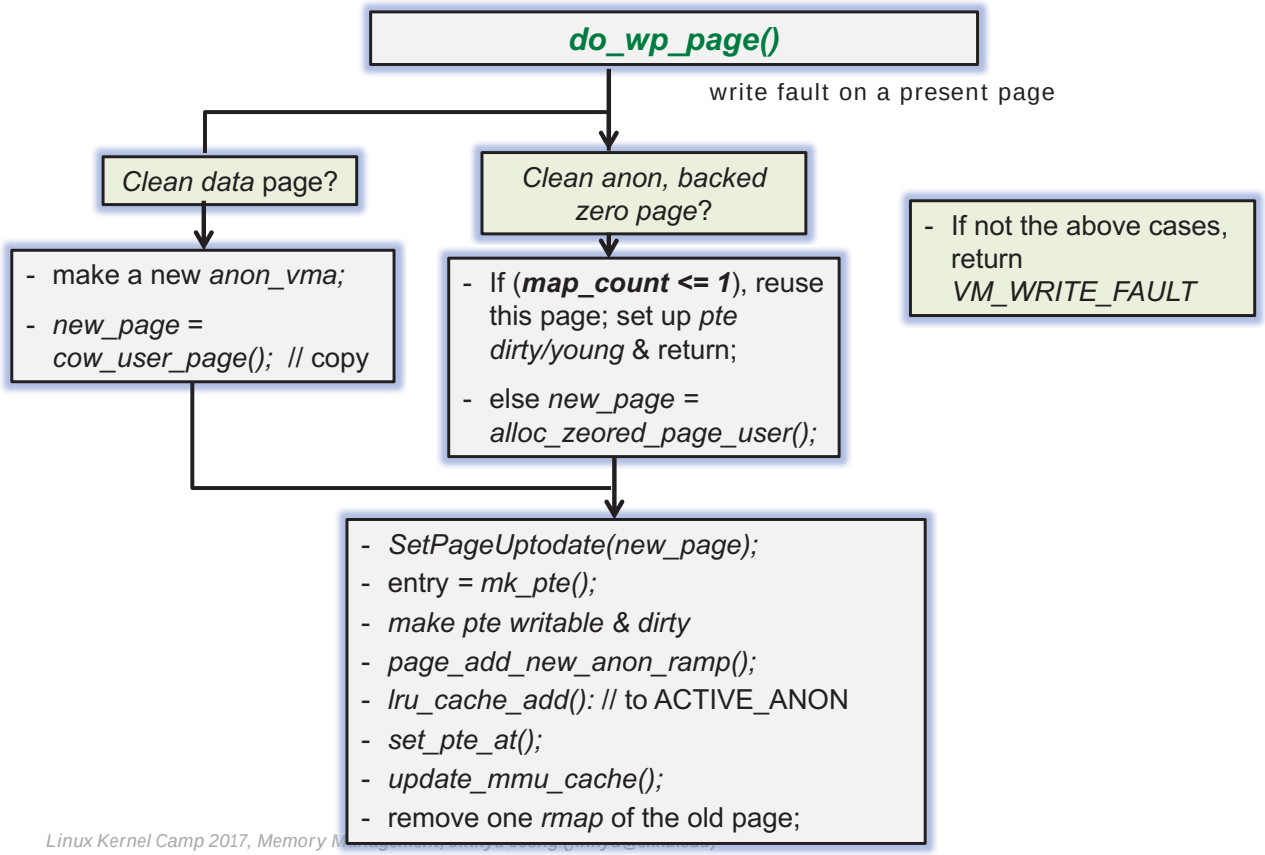
write fault, & *map_count* > 1

```
set_pte_at();
do_page_add_anon_rmap();
do COW by do_wp_page();
// new page -> ACTIVE_ANON
// this is the case that a first write
// occurred on this  swapped-out
// page in initial "fork-sharing"
swap_free (entry);
```

Swap Fault Summary

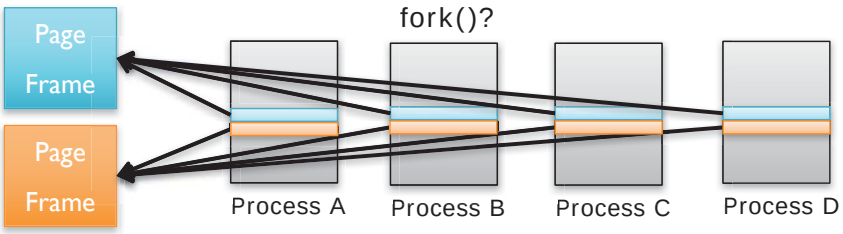
- Prepare the fault page from the swap cache or the swap device.
- If (write fault & not a shared page),
  - The page becomes dirty, so it is meaningless to keep this page in the swap cache.
  - So remove it from the swap cache and treat it as a read fault from now on.
  - Removed from swap cache, so it is now anon. add *anon_vma*.
- If write fault, (fork-shared)
  - ***do_wp_page()*** for *COW* handling
    - If (zero page) *COW*;
    - If (dirty-shared) use the fault page; // shared memory
- Read fault
  - Just keep in the swap cache

do_wp_page(): called from COW & swap-fault



Reverse Mapping

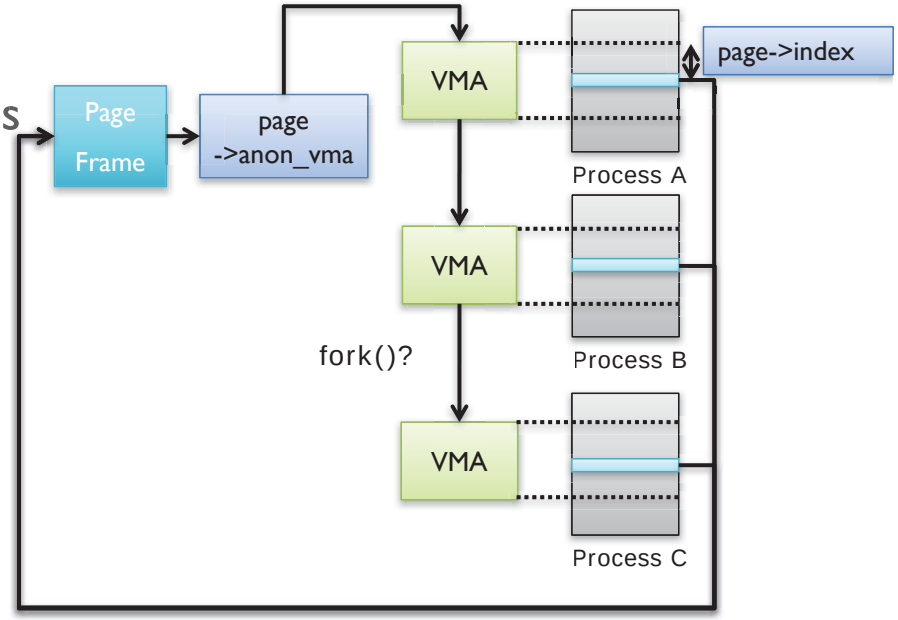
- Record mapping at every mmap()



  - Cannot record all mapping information
  - `sizeof(struct page)` should be small
  - Swapping mechanism needs to know PTEs mapping a struct page
- Reverse mapping
  - Idea: link VMAs when a page in a VMA is shared among processes

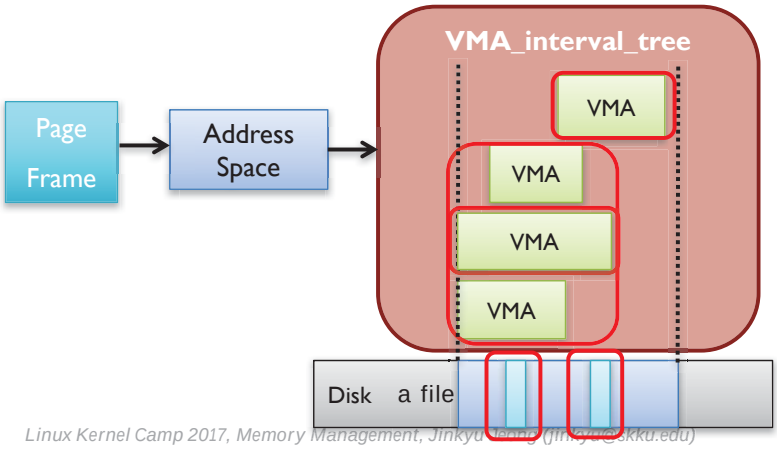
Reverse Mapping

- For anonymous pages
  - anon_vma
    - Links VMAs
  - When to find PTEs
    - page->anon_vma
    - page->index



Reverse Mapping

- For filemap pages
  - When forked, anonymous pages have the same VMA
  - But, filemap pages can be shared by not fork() but mmap()s
- VMA_interval_tree (rb_tree)
  - Represents VMAs for the same file
  - Aligned to start offset of each VMA
  - Offset in a file is a key to find VMAs

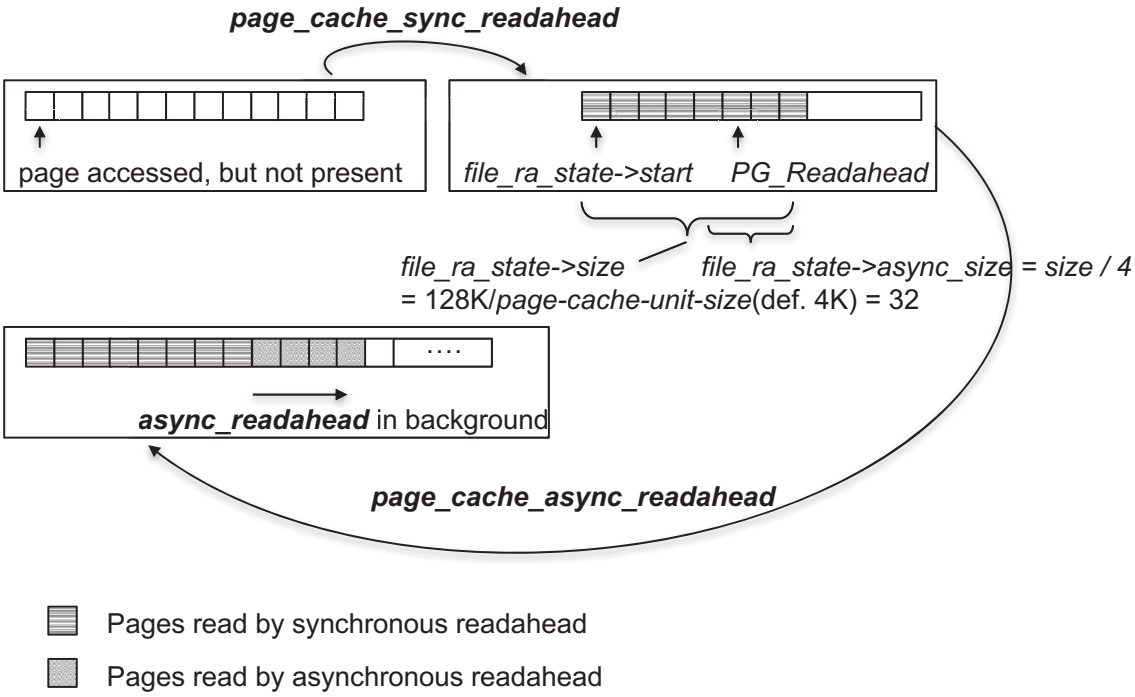



```
struct address_space {
    …
    struct rb_root   i_mmap;
/* tree of private and shared mappings */
    …
}
```


Read-ahead from File & Swap

- To reduce the number of major faults,
  - do **read_ahead**() pages into caches in sync and/or async mode,
  - when the access method is thought to be sequential. (default, can be changed by **fadvice**() syscall.)
  - **do_page_cache_sync_readahead**(), **swapin_readahead**()
  - Max nr_pages = 512; (depends on page cache size) usually 32.
  - Contiguous allocations are preferable both in disk/swap and memory.

Read-ahead from File & Swap



Source: Professional Linux® Kernel Architecture, Wolfgang Mauerer (2.6), Wiley Publishing, Inc.

File System I/O

- do_generic_file_read() // not a fault, just disk I/O
  - Almost same as ***filemap_fault().***
    in using page cache & *ACTIVE/INACTIVE* & *page.ref* handling.
  - *copy_to_user();*
- File I/O page
  - No *_map_count* if its not mmapped.
  - But has file address space as mapping
  - *ACTIVE_FILE or INACTIVE_FILE*
  - At reclaim, write to file if dirty

Swapping (Page Reclaim)

LRU Approximations of Older OS'es

1. LRU must counts all memory accesses on a page. => impossible !
2. BSD 4.3's swapping (LRU approximation)
   - Periodically (timer interrupt handler) clears all *pte's ACCESSED* flag.
   - When clear : 1 => 0 (*pte.age* = 0), 0 => 0 (*pte.age*++)
   - When reclaim, the page with the eldest *pte.age* is the victim page.
3. SysV: Second chance alg.(Clock alg.)
   - At page scan, clear *ACCESSED* bit: (only when insufficient free memory)
   - At clear : 1 => 0 (skip), 0 => 0 (reclaim)
   - *ACCESSED* bit clear scan: faster when the free memory is insufficient, else slower. (varying intervals)

Considerations (1)

- Considerations
  1. Scan-targets of page-reclaim is the set of struct pages in LRU lists, not a set of pte's.
  2. A **struct page** can have multiple pte's mapped.
  3. When the ACCESSED flags of pte's must be cleared?
  4. When do swap-writeback, how copy pte's DIRTY to page's dirty?
  5. When free memory is enough, nothing happens in caches.

Considerations (2)

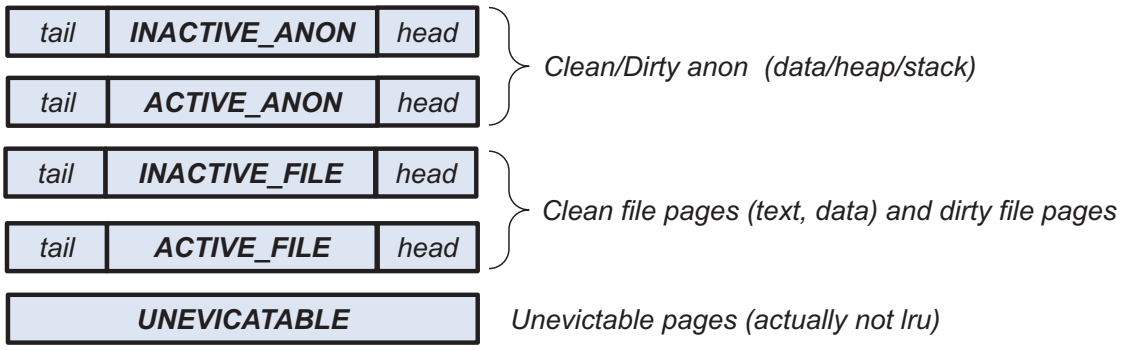
- Important attributes of a page in swapping
  - page_referenced() : the number of pte's with ACCESSED(1) among the mapped pte's of this page. (most important !) Used at the time of "reclaim or active-to-inactive demotion".
  - page.ref
    - set if the page is accessed though mark_page_accessed() (e.g., sys_read(), sys_write(), …)
    - or associated ptes are set while executing page_referenced()
  - file-backed or swap-backed:
    - If dirty, writeback is necessary at the time of reclaim.

Linux 4.x LRU/Swapping

- A working-set algorithm implemented in a Double-LRU scheme.
  - **INACTIVE_FILE, INACTIVE_ANON** : set of victim-candidate pages.
  - **ACTIVE_FILE, ACTIVE_ANON** : hot pages in the working set.
  - Reclaim scan works only on **INACTIVE lru's**.
    - Second-chance algorithm for each list
  - Before reclaim scan, **shrink_active_list**() is done to balance active and inactive lists
  - 2Q LRU to evict single-use data quickly

LRU Lists per Zone

- **INACTIVE** list : candidate-pages for reclaim, but classified into the working-set.
- **ACTIVE** list : hot pages in the working set.
- Promotion: INACTIVE → ACTIVE, demotion: ACTIVE → INACTIVE
- Why called LRU?: use the head at insertion, scan from the tail at page reclaim.



- active_pagevec[14], and lru_pagevec[14] for deferred decision (to active or inactive)
  - per CPU page buffer for recently fetched page (14 pages): just a cooking buffer to reduce reclaim scan & deferred decision.
  - When full or at reclaim, all pages are inserted to the active or inactive list (deferred decision)

Collaborations with the page fault handler

- Page fault handler decides "ACTIVE or INACTIVE" at page fault.
  - **Anon pages**:
    - at the first fault → active
    - fetched from swap device or swap cache → inactive
  - **File pages** (text & data): active or inactive by the "**Refault distance**" of a page. (explained later)
    - If a fault page is regarded as workingset → active
    - If a page is new or out of workingset → inactive

Refault distance & zone->inactive_age for file-backed pages

- Refault distance & zone->inactive_age : just for file-backed pages.
  - ***inactive_age***: the cumulated sum of
    - The number of memory-evicted pages form *INACTIVE_FILE*, plus,
    - The number of promoted pages (*INACTIVE_FILE => ACTIVE_FILE*)
    - i.e., the number of total pages that have been removed from the *INACTIVE_FILE* lru from booting.
    - **An approximated value of the number of page access**
  - ***E = inactive_age(t_e)*** when a page is evicted from memory.
  - ***R = inactive_age(t_r)*** when the page is fetched to memory again later.
  - Then, "***R – E*** " is the total number of pages removed from ***INACTIVE_FILE*** during the page was outside of memory.

Refault distance & zone->inactive_age for file-backed pages

- ***(R -E) + nr_inactive(t_r) : Refault-distance***
  - Is the total number of pages that were in INACTIVE_FILE during the *p* is outside of memory.
  - This is the minimum number of pages referenced during this period.
  - This is called ***refault-distance*** of a page when it comes back to memory.
- ***Workingset-size(t_r) = nr_active(t_r) + nr_inactive(t_r)***
- If ***((R -E) + nr_inactive(t_r) <= nr_active(t_r) + nr_inactive(t_r))***
- then // ***refault-distance <= working-set size***,
  - so this page should go to the ***ACITVE_FILE.***
- else,
  - goes to the ***INACTIVE_FILE.***
- So, testing " ***(R -E) <= nr_active (t_r)*** " is enough !

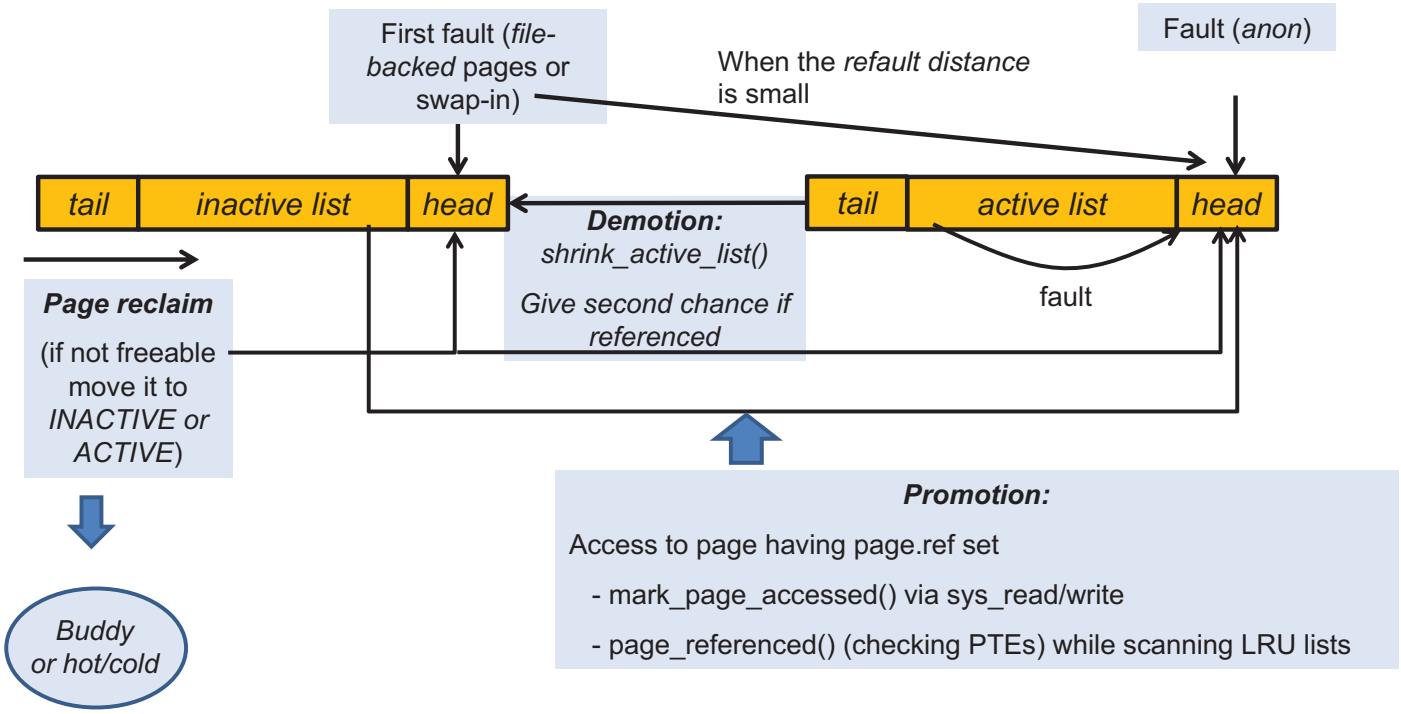
Implementation by *Shadow-Entry*

- Shadow-entry: a snapshot of the inactive_age:
  - Return value of the function ***workingset_eviction****(mapping, page);*
    - Called when an *INACTIVE_FILE* page is evicted or when it is promoted to *ACTIVE_FILE.*
    - ***zone->inactive_age ++;***
  - When a page is evicted & removed from the page cache, the ***shadow-entry*** replaces the ***struct page*** in the *page cache.* ***(E)***
  - At a later fault on the file-backed page, the ***shadow-entry (E)*** in the page cache & the current ***inactive_age (R)*** are taken to calculate the ***refault distance.***
  - If ***(R -E) <= nr_active(t) : to ACTIVE_FILE***
  - else to ***INACTIVE_FILE***

Unevictable List

- Unevictable pages: in order to reduce vm_scan at page-reclaim
  - **RAM file system** pages
  - Locked shared memory region
  - "mlock()ed" pages (eg. during mmap() time)
  - mem-cgroup control data structures
- Checking at page reclaim
  - shrink_activelist(), shrink_inactivelist(), shrink_pagelist() checks the page with page_evictable().
  - If (unevictable) move it to the unevictable list.
  - If the unevictable reason is released, move it to an LRU list again.

LRU Management



Linux 4.x LRU/Swapping Principles

- Page-reclaim scan will be done only when the numbers of free pages of zones is under the specified watermark.
  - Synchronous reclaim: ***zone_reclaim***() and ***try_to_free_pages***() when ***alloc_pages***() fails in a zone.
  - Asynchronous reclaim: ***balance_pgdat***() (kswapd) when ***alloc_pages***() failed in all zones.
- Metrics for deciding a working-set page.
  - **page.ref** flag
  - ***page_referenced***(): number of pte's with **ACCESSED** among all mapped pte's.
  - **Refault distance.**

Linux 4.x Reclaim Procedure

- Reclaim Procedure
  1. Refill inactive lrus from active lrus (**shrink_active**(), old refill_inactive()).
     - **nr_to_scan** is determined by the reclaim-priority.
  2. Reclaim pages without **referenced-ptes** from inactive lru's. (**page_referenced**()).
  3. If page is mapped in ptes, unmap it. (anon, text, data, file-mapped)
  4. If the reclaimed page is **DIRTY**, do write-back. (dirty-anon to swap, file system dirty page to file).
     - pte.dirty is copied to page.dirty at unmapping.
     - When write-back is done, the page is inserted at the tail of inactive lru list
  5. Reclaim clean pages
  6. Put back reclaimed pages.
     - Single page => hot-&-cold cache per cpu. (to enhance local CPU cache in SMP).
     - 2n page => buddy allocator.
  7. Retry (async. != sync.), case by case, but usually,
     - **Sync**.: stop if **nr_to_reclaim** satisfied. Side-effect: zone-balanced. (until max_scan).
     - **Async**.: try to make all zones to be balanced. (high watermark & balance).

Linux 4.x Reclaim Types

- ***zone_reclaim()*** (**sync. reclaim**): for zones under low watermark.
  - Quickly reclaim clean swap cache pages or page cache pages
  - Core part of the ***zone_reclaim()*** is ***shrink_zone().*** (also called by ***kswapd***)
- **try_to_free_pages()** (sync. reclaim): for zones under min watermark
  - Perform write-back I/O if necessary
  - Similar to kswapd in terms of shrinknig zones, but stops if nr_to_reclaim is satisified
- ***kswapd*** (**async. reclaim**):
  - Tries to make all zones to be *balanced*.
  - For each zone, call ***shrink_zone();***
  - A balanced zone:
    - ***nr_free >= WMARK_HIGH*** &
    - ***Well-distribution*** *of free pages per size in buddy.* (2^n).
  - If not all-balanced, retry with a modified priority. (***nr_to_scan*** ↑)

Page Reclaim in Detail

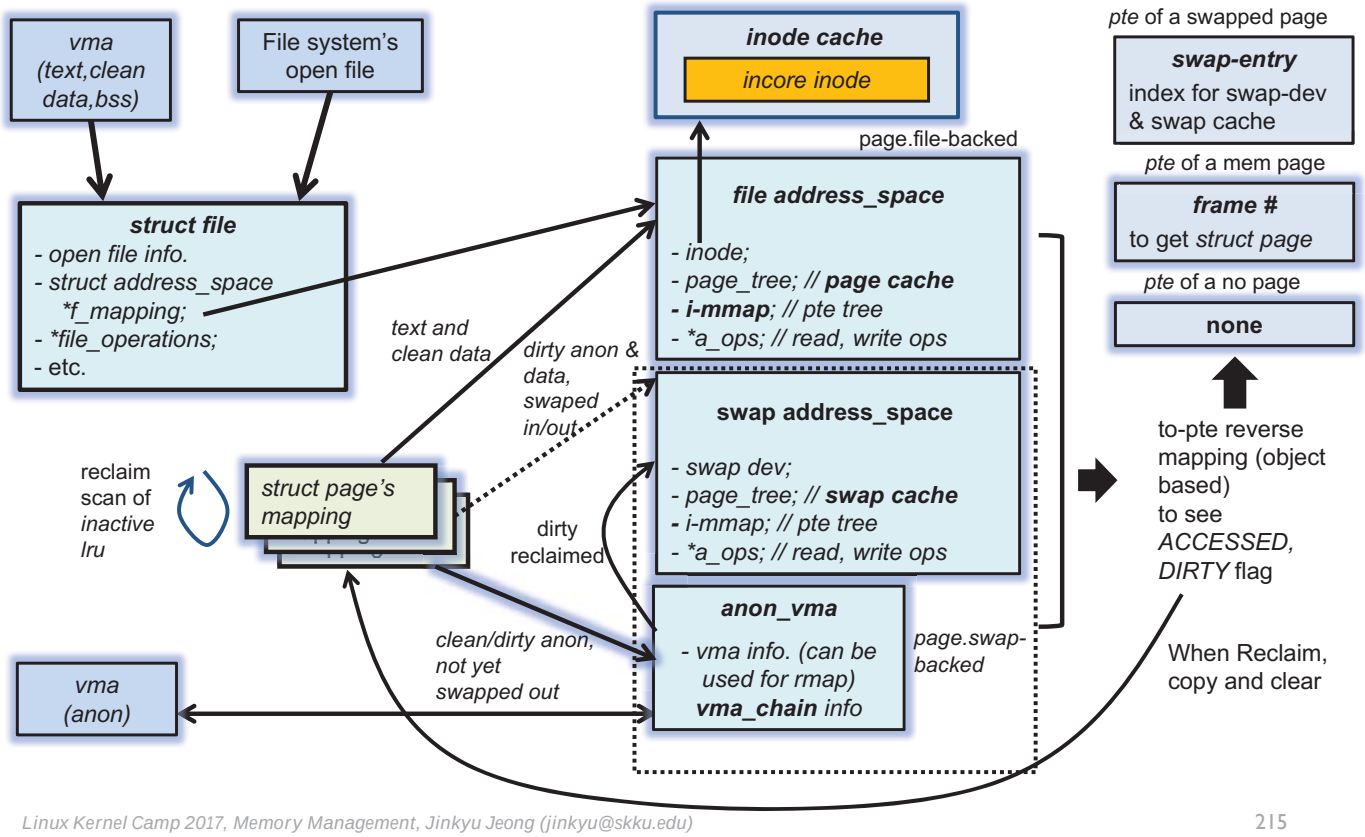
Major *Page* Attributes for Swapping

Members	Description	
_map_count	Number of *pte's* mapped into this page.	
mapping	*file address_space*	*page cache structure, it has vma rmap (vma_interval_tree)*
	swap address_space	*swapcache (not directly stored, but a trick exists to return swap address space, (i.e,.* ***page_mapping()****)*
	anon-vma	*anon reverse mapping (anon map)*
	nothing	*shared zero page* for clean *anon* pages
file/swap-backed flag	*file-backed*	*text, clean data, clean bss*
	swap-backed	anon (dirty data/bss & heap / stack), in swap device or to be in
dirty	S/W *dirty* flag. When *try_to_unmap()* for reclaim is being done, this is copied from *pte's DIRTY* flag.	
referenced (page.ref) for file-backed	***S/W reference flag***. Cleared when inserted into *INACTIVE or ACTIVE* for the first time. Set when there is another fault on the page. Used & cleared in INACTIVE scan. Do not confuse this with ***page_referenced()*** ***(= num of referenced ptes) !***	
ACTIVE (y/n)	Is in *ACTIVE* or *INACTIVE lru*?	
lru pointer	*INACTIVE_ANON, ACTIVE_ANON, INACTIVE_FILE, ACTIVE_FILE, UNEVICTABLE*	
private	An *anon* page has the ***swap_entry*** if the page is mapped in swap cache	

Page Attribute Set & Test Functions

Functions	Description
page_referenced()	- check map_count, mapping - return the number of *pte's* (***ACCESSED***) among all *pte's* mapped. - Clear ***pte's ACCESSED flag*** for the next scan !! - Most important one !!!
page_check_references()	- See ***page_referenced() & check page.ref flag*** and clear or set - Used in inactive page classification: *ACTIVE, KEEP, RECLAIM_CLEAN, RECLAIM* - i.e. used in ***shrink_page_list()*** // final memory eviction
page-mapped()	- return *_map_count >= 0,* i.e. has ptes?
page_evictable()	- *evictable* page?
mark_page_accessed()	- set page.*ref. if already set, clear it and activate the page*
SetPageActive()	- set active to go to *active lru*
ClearPageActive()	- reset active to go to *inactive lru*
ClearPageDirty()	- make page *clean*
SetPageDirty()	- set page *dirty* (not the *pte's DIRTY*)
PageDirty()	- page *dirty*?

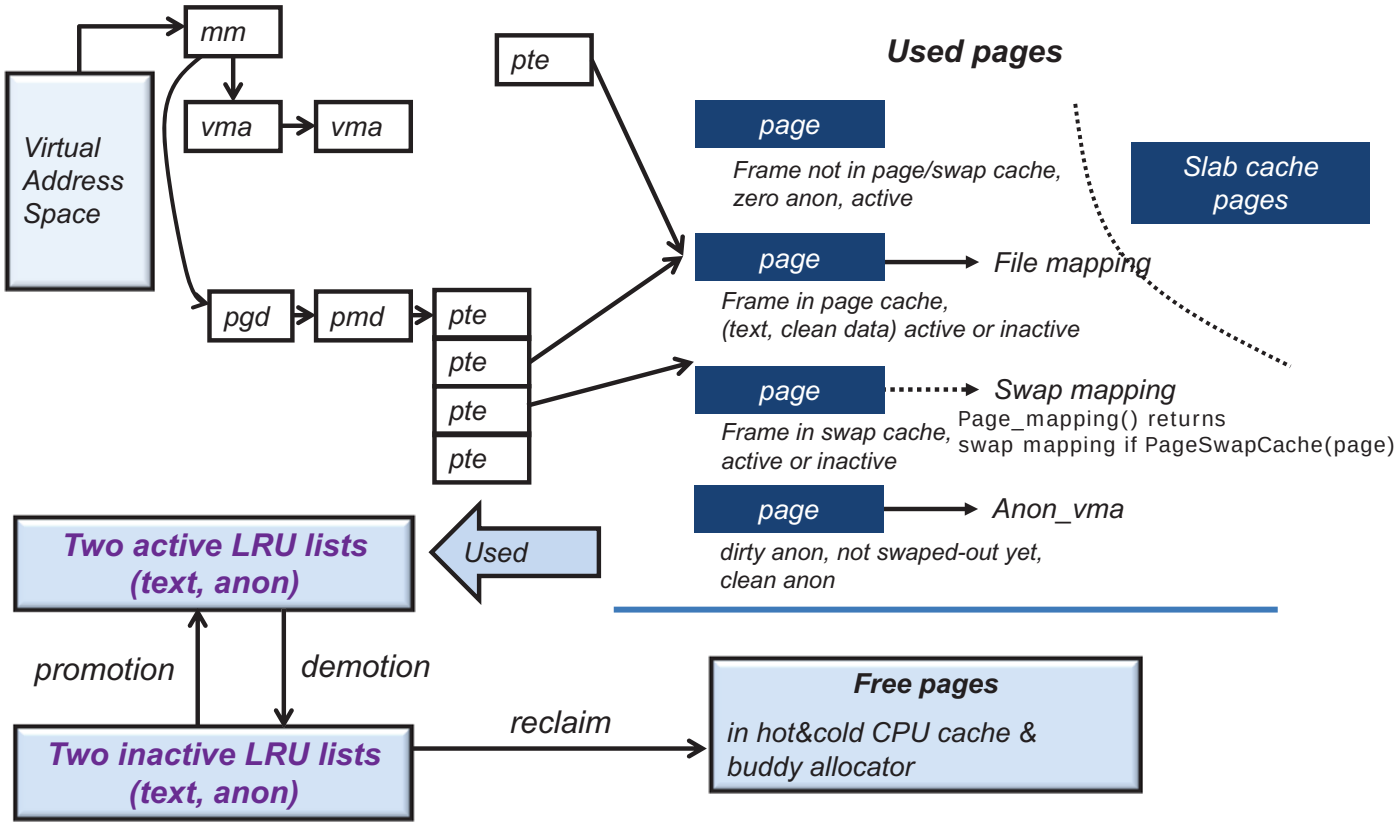
Address Space & Reverse Mapping



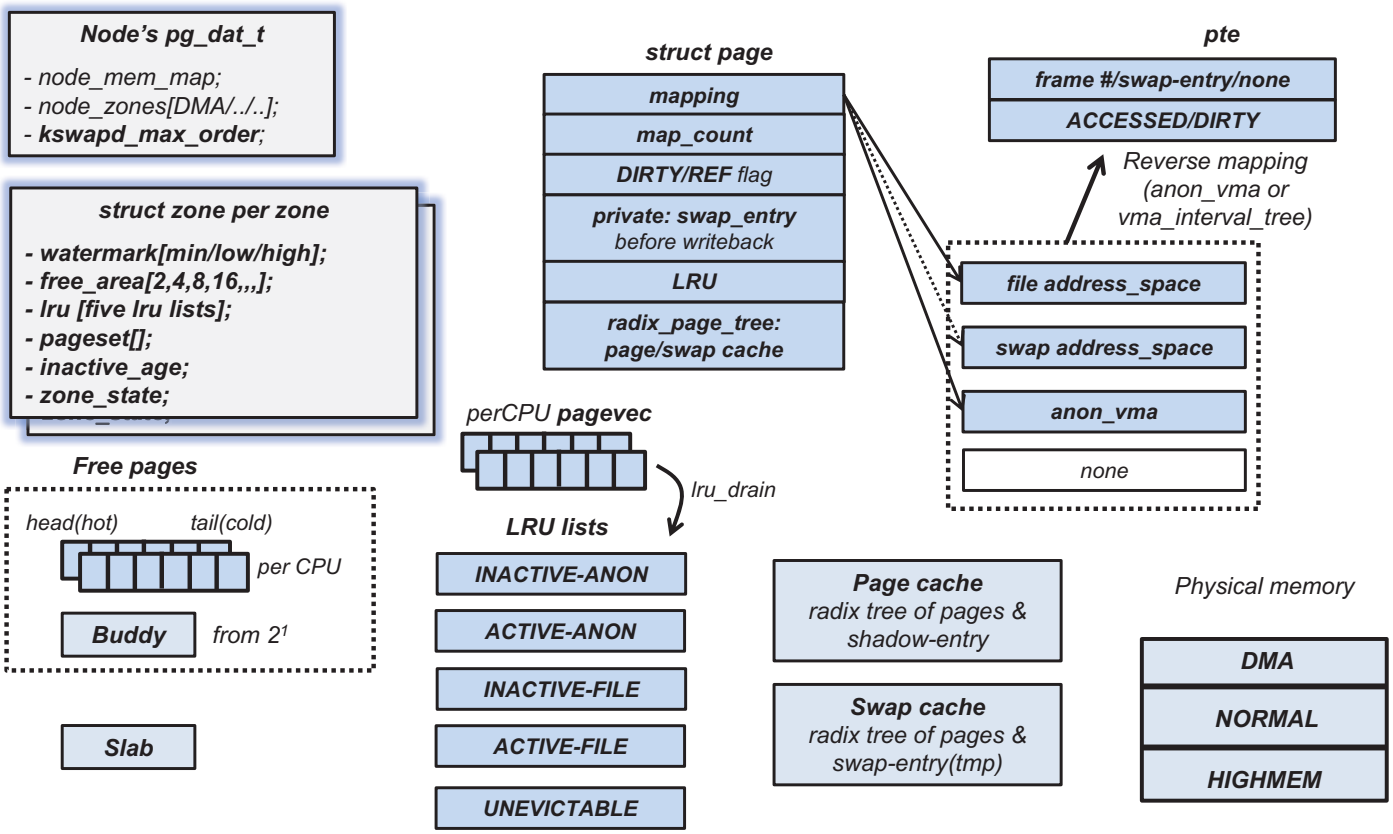
Why reverse mapping? (*page to pte's*)

1. try_to_unmap(page) :
   - When a page is decided to be evicted from memory,
   - All *pte's* mapped to this page must be dropped (cleaned).
   - ***try_to_unmap(page)*** finds all *pte's* of a page by using the pte-tree in,
     - File address space ***rmap_walking***
     - Swap address space *rmap_walking*
     - Anon-map *rmap_walking*
2. Anon page fetched from the swap device (by read) must be checked whether it is dirty by looking at the DIRTY flag of the pte at a later time of re-swapout.
3. When do page_referenced().
   - *Check & count numer of pte's with ACCESSED,*
   - *Clear ACCESSED flag of a pte*

Active & Inactive Lists & Page States



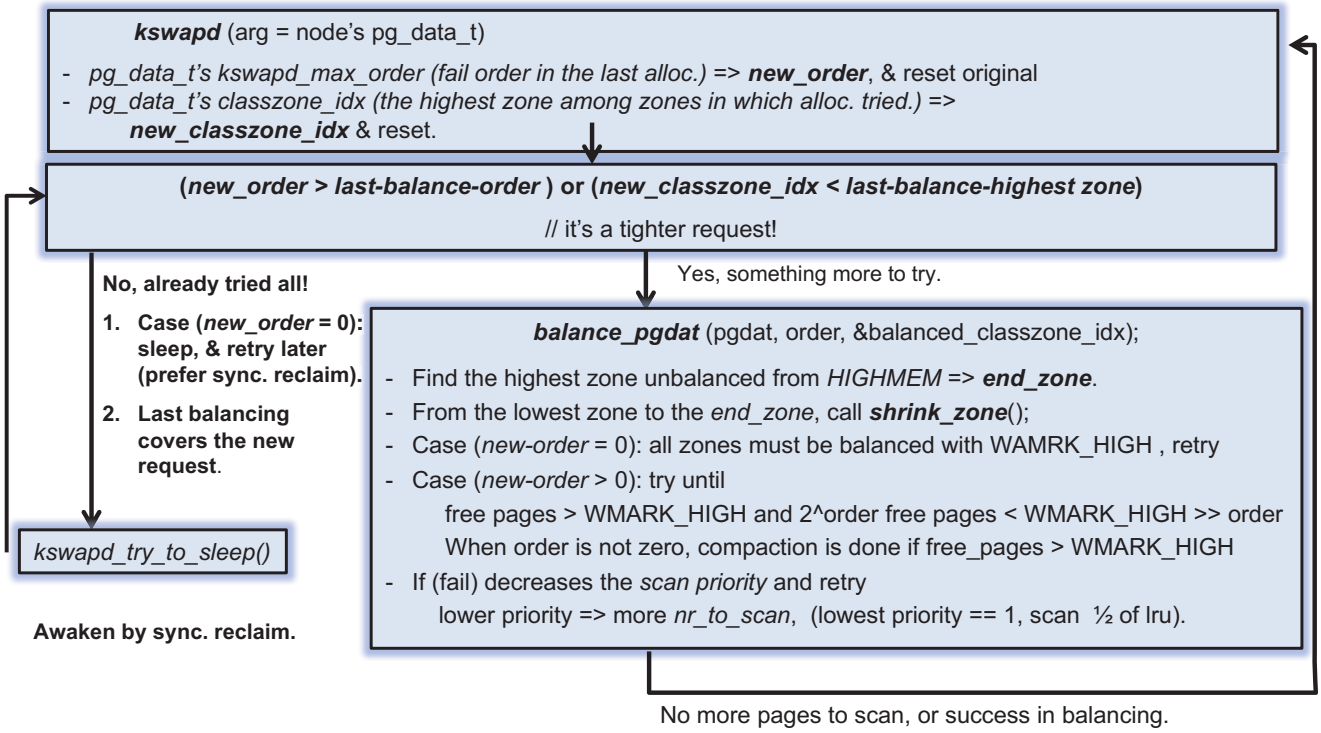
Data Structures for Swapping



Async. Swapping (page reclaim in all zones)

- ***kswapd*** kernel thread
  1. Starts running in background when free pages of every wanted zones are under ***WAMRK_LOW***
  2. From ***HIGH_MEM*** to ***DMA*** zone, (called, *fallback order*)
  3. Try to make all zones balanced.
  4. A balanced zone is a zone that satisfies,
     - Satisfies ***WMARK_HIGH*** &
     - Number of free pages per *alloc-order* (2^{order}) of a ZONE are well distributed.
  5. initial ***nr_to_scan = nr_lru_pages / $2^{priority}$)***, increased when retry by the scan priority. (priority 12 → 1)
  6. Repeat: ***reclaim INACTIVE_ANON => shrink ACTIVE_ANON => reclaim INACTIVE_FILE => shrink ACTIVE_FILE => shrink_slab()***
  7. Sleep if all zones are balanced or no more pages to scan.
  8. i.e. try to make all zones to be balanced (***well-distributed, WAMRK_HIGH***).

kswapd Kernel Daemon



balance_pgdat()

1. Determine the *end_zone* (the first unbalanced zone from the HIGH_MEM in decreasing order), skips a balanced zone (here, no watermark, just see balance.).
2. Form highest zone to end_zone, call *age_active_anon();* (demotion of active pages)
3. For every zone from *the lowest zone to end_zone*:
   - Call *kswapd_shrink_zone();*
     - If (this zone is balanced (*WAMRK_HIGH*) && no *lowmem_pressure**) return true;
     - else call *shrink_zone();* return true if (*nr_scanned* >= *nr_to_reclaim*)
       // lowmem_presssure is set if quite many *buffer-headers* are in *HIGH_MEM*
4. Wakeup all sleep processes waiting for alloc_page;
5. Case (new-order = 0): all zones must be balanced with WAMRK_HIGH , retry.
6. Case (new-order > 0): free pages > WMARK_HIGH, 2^order free pages > && WMARK_HIGH / 2^order. This is checked by *pgdat_balanced()*.
7. If (fail) decreases the scan priority and retry from the first.
   - lower priority => more *nr_to_scan,* (final priority == 1, scan ½ of *lru*).

* lowmem_pressure: True if current shrinking zone is ZONE_HIGHMEM and lowmem zones have too many buffer heads
When this is set, highmem zone continues shrinking to relieve the pressure of lowmem zones.
Note that buffer heads are allocated from lowmem zones

balance_pgdat()

balance_pgdat *(pg_data_t *pgdat, int order, int *classzone_idx)*
- Start with *DEF_PRIORITY* (12) (*nr_to_scan* : *nr_pages* / 2 ^{priority})

for all zones from the highest to lower, choose the highest unbalanced zone to start (end_zone):
- Skip *zone_reclaimable*(); // fully scanned zone in the last scan
- *age_active_anon*(zone,,); // if (*INACTIVE_ANON* length * *inactive_ratio*(see source for detail) < *ACTIVE_ANON length*)
  // shrink_active_list(); // here, ACTIVE_FILE is skipped because, *file-backed* pages mostly go to *INACTIVE*;
- If this zone is the *HIGH_MEM* & has too many *buffer headers*, *end_zone* = *HIGH_MEM*; break;
- If not *zone_balanced*(NO_*WATERMARK*, order) end_zone = this zone; // test only 0~order distribution
  // i.e. even if below *WMARK_HIGH*, skip if free pages per order are well-distributed;

for all zones from the *NORMAL* zone to the *end_zone*:
- if (*node-level–compaction-necessary* is set) but at least one zone is balanced and not fragmented, reset the flag;

If (*scan priority* < (*DEF_PRRIORITY* – 2)), *sc.may_writepage* = 1; // allow writeback during shrink_page_list()

for all zones from the NORMAL zone to the end_zone:
- ***kswapd_shrink_zone*** *(zone, end_zone, &sc, &nr_attempted); // nr_to_reclaim* (WAMARK_HIGH) is determined and scan.
- if at least one zone is balanced, *raise_priority* = false;

Wakeup all processes waiting for alloc_pages;
If (*nr_reclaimed* > 2^{order}), set order = 0; // to stop kswapd
If (node compaction is necessary) *compact_pgdat(pgdat, order);*
If (*nr_reclaimed* == 0 && raise_priority is allowed)
priority-- & retry; // *nr_to_scan* = *lru length* / $2^{priority}$

Until scan *priority* = 1
(1/2 scan) && !***pgdat_balance***()

kswapd_shrink_zone()

kswapd_shrink_zone *(*zone, classzone_idx, struct scan_control *sc, *nr_attempted)*

- sc->***nr_to_reclaim*** = *max(SWAP_CLUSTER_MAX(32), high_wmark_pages(zone));*
- If (this is *HIGH_MEM* & too many buffer headers), set ***lowmem_pressure***;
- If (***zone_balanced***(WMARK_HIGH, order,,) && ! ***lowmem_pressure***) return;
  // over low & 0~order: well-balanced
- ***shrink_zone***(zone, sc,); // shrink until ***nr_to_reclaim*** is obtained;
- If (*nr_scanned* > *nr_to_reclaim*), return true, else return false;
- So, not responsible to fill free pages up to high water mark.

zone_balanced *(WMARK, order,,)*

- Call zone_watermark_ok()
- Watermark is WMARK_HIGH + balance_gap (min(1% of zone pages, WMARK_MIN))

shrink_zone()

alloc_page() group

kswapd

sync. reclaim

async. reclaim

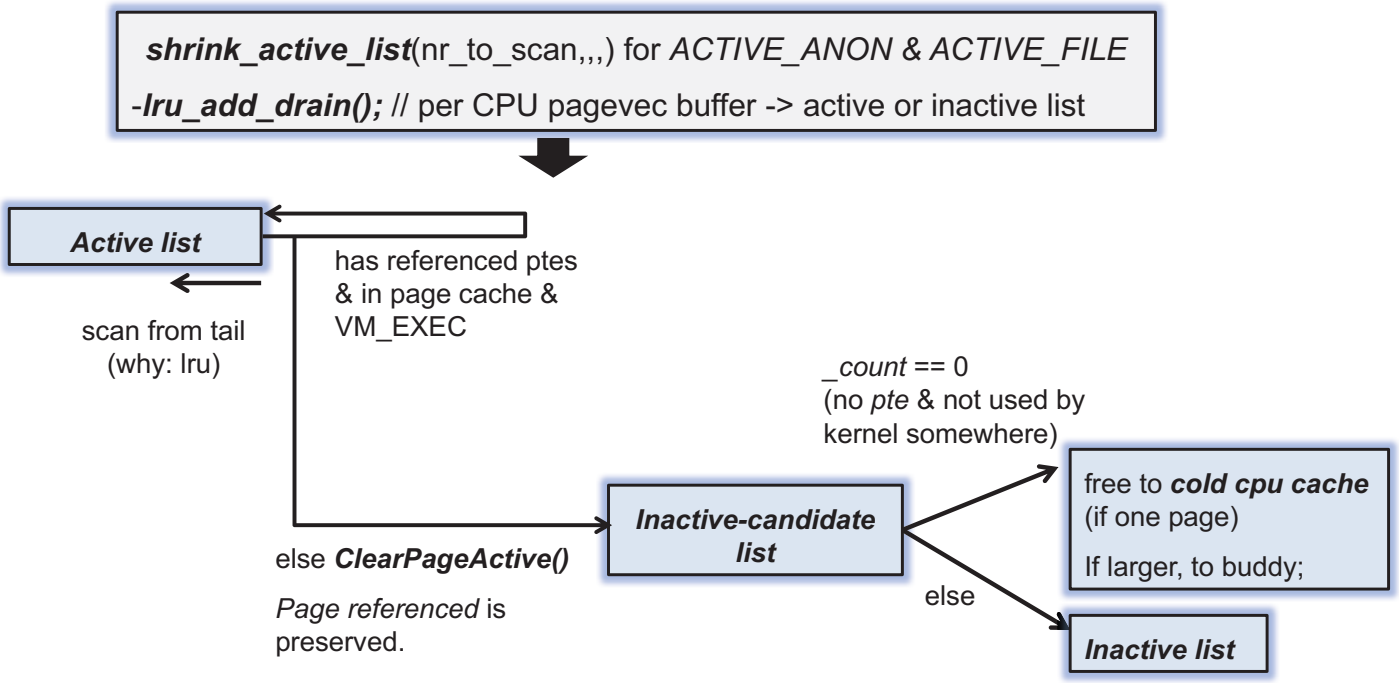
shrink_zone(struct zone *zone, struct scan_control *sc,,)

- ***shrink_lruvec*** *(,,);*
- ***shrink_slab****();* // reclaim free slab caches and other caches (inode cache)
  // once called, skip at the next turn of zone (call just once)
- *If (nr_to_scan is satisfied) but (GFP_REPEAT is true and nr_to_reclaim is not satisfied) retry*

shrink_lruvec *(,,)*

- *nr_to_scan* is determined by the scan priority & lru size;
- Repeat while (*INCTIVE_ANON or ACTIVE_ANON or INACTIVE_ANON has at least one page*)
  - For all lru lists (*INACTIVE_ANON, ACTIVE_ANON, INACTIVE_FILE, ACTIVE_FILE*)
    - call ***shrink_list***(); // inactive list => ***shrink_inactive_list***()
      // active list => ***shrink_inactive_list***()
    - If *nr_to_reclaim* is not satisfied, or *nr_to_scan* is not satisfied (because inactive length is short) continue;
- If *(INACTIVE_ANON* length is small) call ***shrink_active_list()*** again;
- Increment *nr_reclaimed* & return

shrink_active_list()



Inactive LRU

- Which pages are in the inactive lru?
  - A page with pte mapping (text, data pages by the page fault handler).
  - Pages without pte mapping or without address-space mapping.
  - File-backed pages with zero and one **page.ref** flag. (more than double-faulted)
- page.ref set: (only for file-backed pages)
  - mark_page_accessed (sys_read/write())
  - At inactive-reclaim (**page_check_references**()), if page has referenced ptes
- page.ref clear:
  - At the first insertion to inactive or active list (first fault, first promotion)
  - At inactive-reclaim (**page_check_references**())
    - page.ref is cleared first
    - If the page has referenced ptes, set page.ref again

shrink_inactive_list()

shrink_inactive_list *(nr_to_scan,,,)* for INACTIVE_ANON & INACTIVE_FILE

- *lru_add_drain(); //* per CPU pagevec buffer -> *active or inactive list*

isolate_lru_pages()

Inactive list → ***tmp. page_list***

scan from tail

Move inactive pages to tmp. page_list by *isolate_mode* until *nr_scan* moves;

***isolate_mode*:**

- 0: all *nr_pages*; // serious mode
- *ISOLATED_UNMAPPED*: skip *page_mapped()*; // have ptes
- *ISOLATE_CLEAN*: skip dirty pages;
- In all cases: skip *PageWriteback()* pages; // page is in I/O
  skip *unevictable* pages;

shrink_page_list(tmp. page_list)

- Classifies pages ***into RECLAIM, RECLAIM_CLEAN, KEEP, ACTIVE***
- Free *RECLAIM, RECALIM_CELAN* pages; (*hot_cold_free*)
- Return *KEEP, ACTIVE* pages in tmp. page_list;

→ ***putback_inactive_lits(tmp. page_list)***

- Return pages to *ACTIVE or INACTIVE* list as determined in ***shrink_page_list();***

return

shrink_page_list()

shrink_page_list (final_list_to_reclaim);

page_check_references();

- get ***page_referenced();*** // get the number of pte's with ACCESSED among mapped ptes
  // and clear ACCESSED by traverse the reverse mapping. => ***refer_ptes***
- get ***page.ref*** and clear it; // TestClearPageReferenced();=
  //=> ***old_page.ref***, used only for file-backed pages.

if ***force_reclaim***, reclaim all.

refer_ptes > 0 ?

no, reclaim anyway

yes

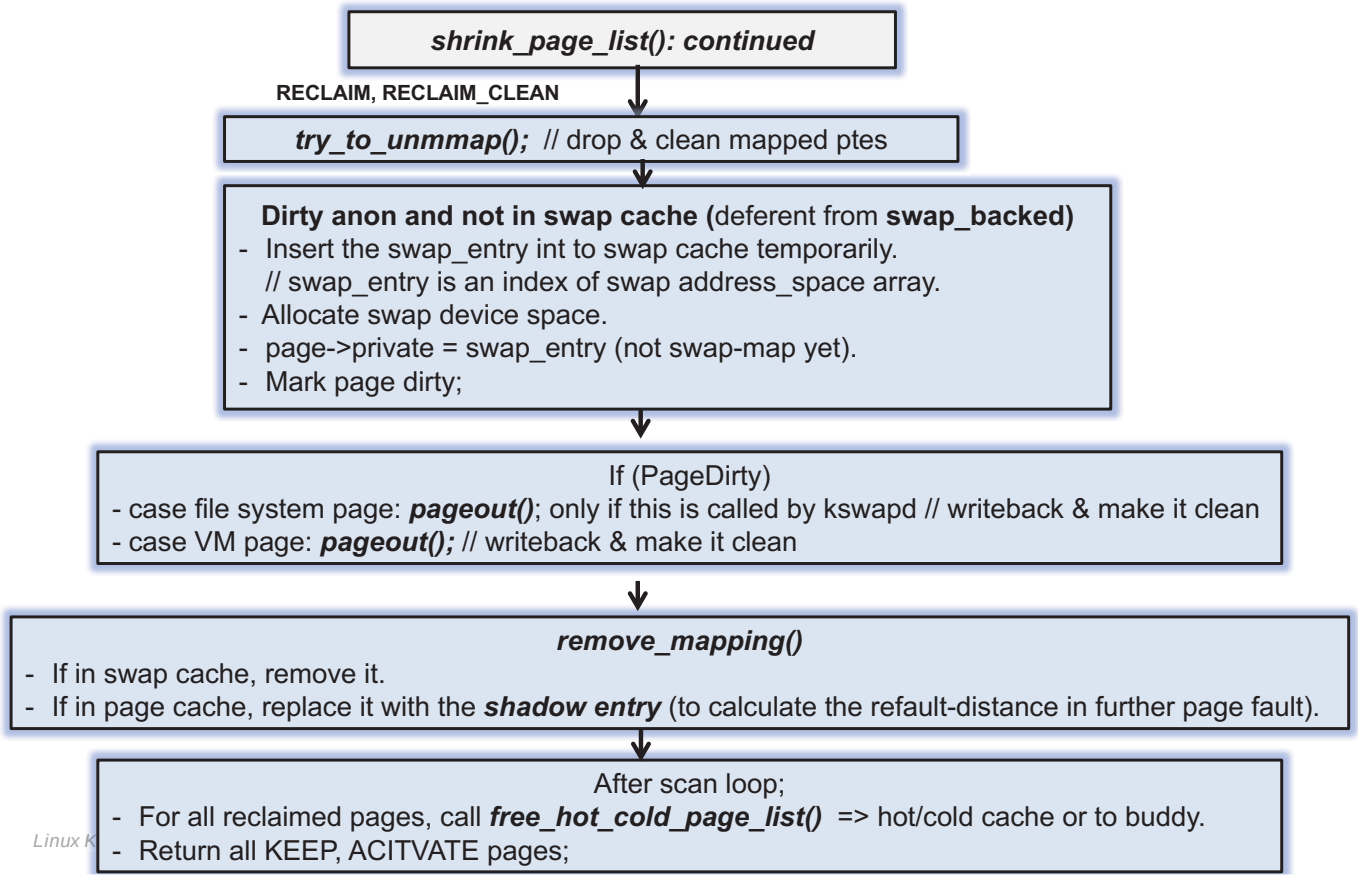
: case (refer_ptes > 0) //at least one pte referenced

- if (swap_backed): return ***ACTIVATE***; // accessed anon pages
- then file_backed,
  - SetPageReferenced(page);// set page.ref because it has referenced ptes;
  - if (***old_page.ref*** or ***refer_ptes*** > 1)
    return ***ACTIVATE***; // shared-referenced or many-faulted data page
  - if (VM_EXEC) return ***ACTIVATE***; // referenced text page;
  - return ***KEEP***; // data page not shared-referenced, keep in inactive list.

- return ***RECLAIM_CLEAN*** or ***RECLAIM***;
  // clean or dirty reclaim

force-reclaim: emergency, when contiguous allocation of a 2^{order}-sized page is impossible because of zone fragmentation, zone compaction (moving of pages) will be done after force_reclaim.

shrink_page_list(): reclaimbale page handling

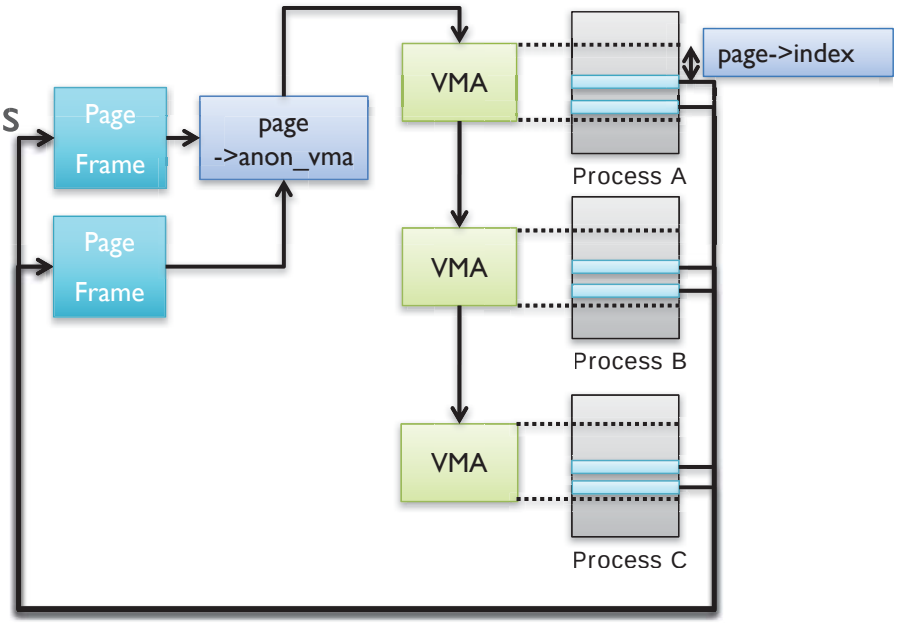


Unmap PTEs upon Page Eviction

- try_to_unmap(page): for drop ptes from a page.
  - try_to_unmap → **rmap_walk_anon**() or **rmap_walk_file**()
  - rmap_walk_anon:
    - by using the anon_vma, anon_vma_chain, vma, mm, get pte;
  - rmap_walk_file:
    - page->mapping (= address space)-> i_mmap (interval tree of VMAs)
  - For each pte, call try_to_unmap(pte);
    - clear pte & if pte DIRTY, set struct page as DIRTY
  - See rmap.c!

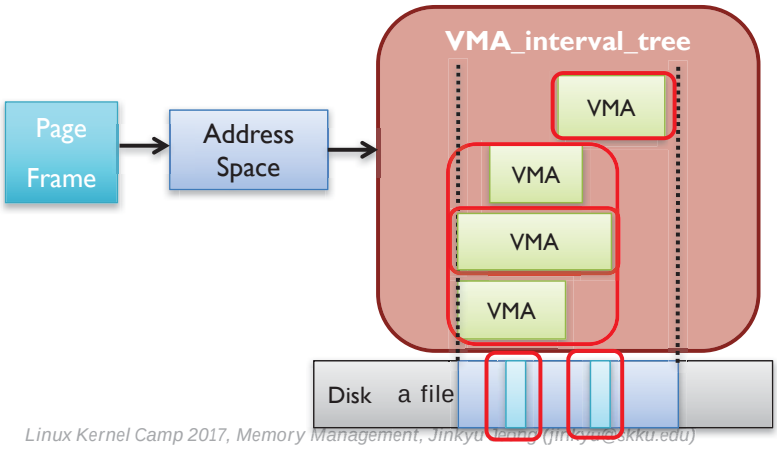
Reverse Mapping (revisited)

- For anonymous pages
  - anon_vma
    - Links VMAs
  - When to find PTEs
    - page->anon_vma
    - page->index



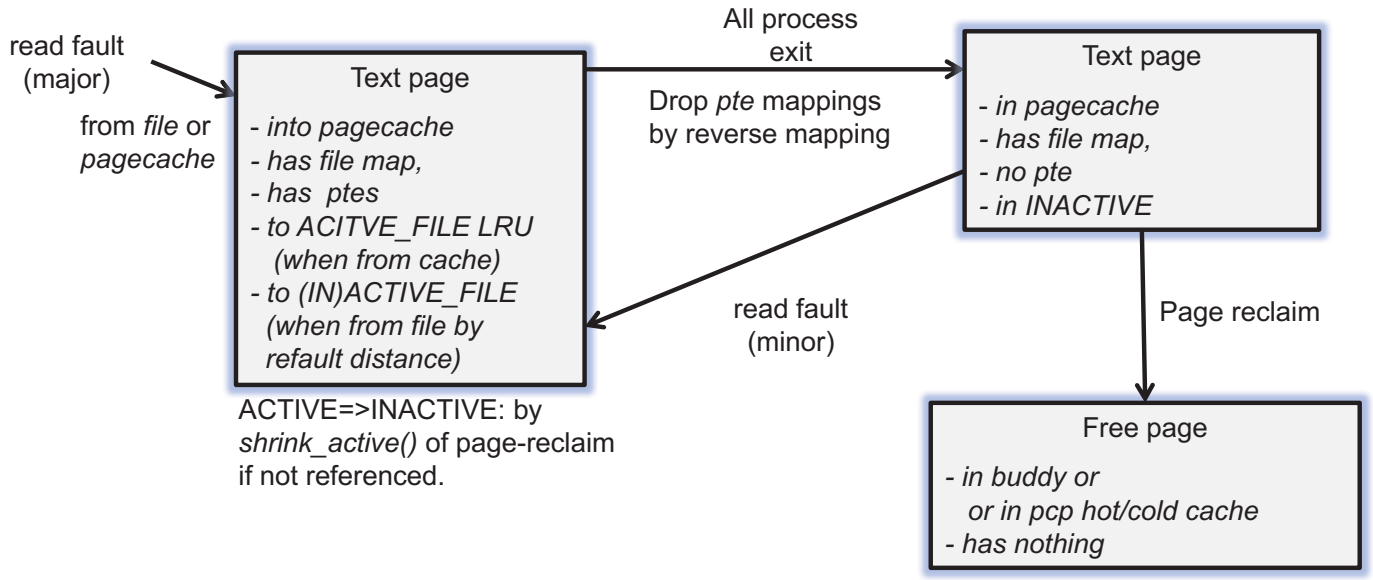
Reverse Mapping (revisited)

- For filemap pages
  - When forked, anonymous pages have the same VMA
  - But, filemap pages can be shared by not fork() but mmap()s
- VMA_interval_tree (rb_tree)
  - Represents VMAs for the same file
  - Aligned to start offset of each VMA
  - Offset in a file is a key to find VMAs

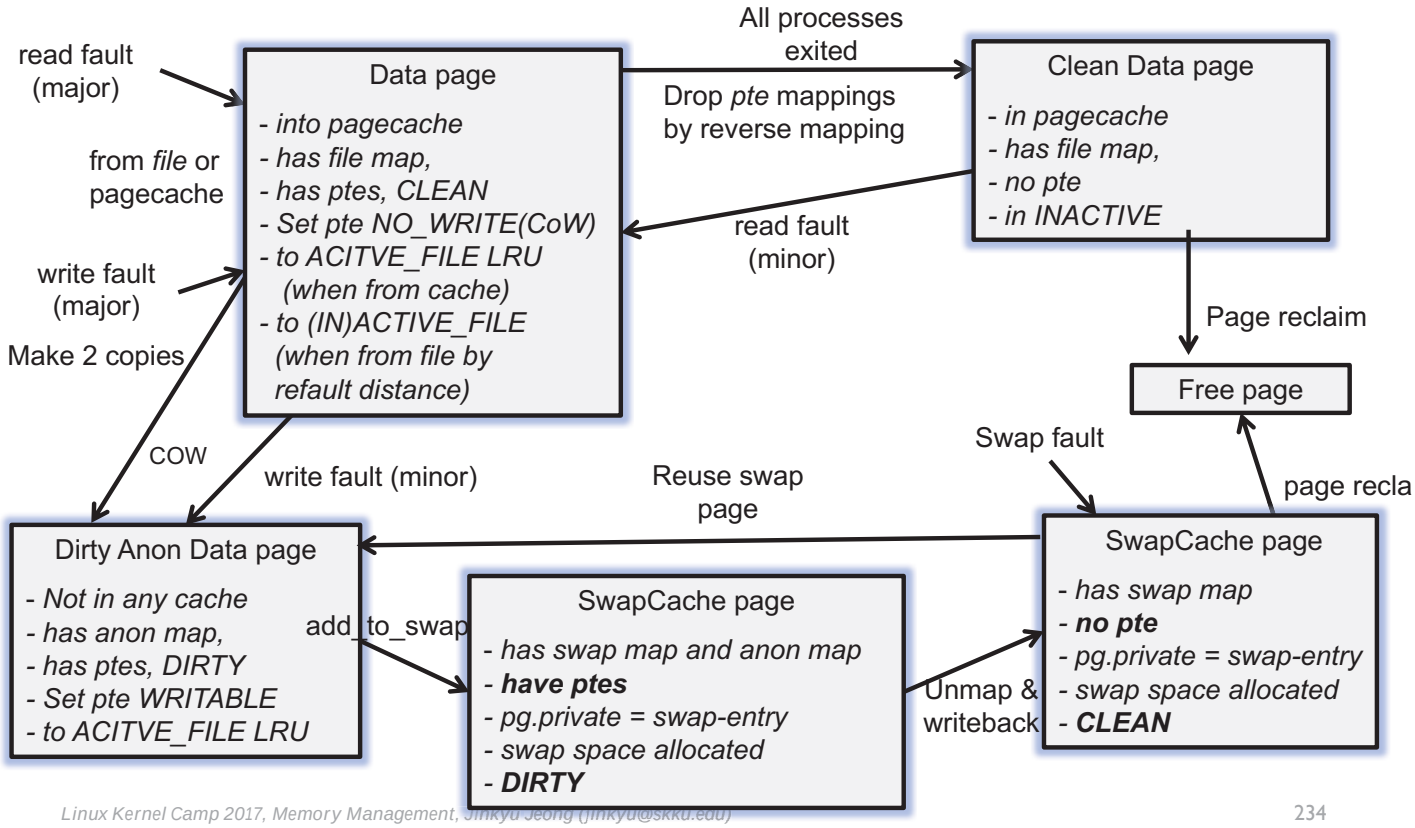



```
struct address_space {
    …
    struct rb_root   i_mmap;
/* tree of private and shared mappings */
    …
}
```

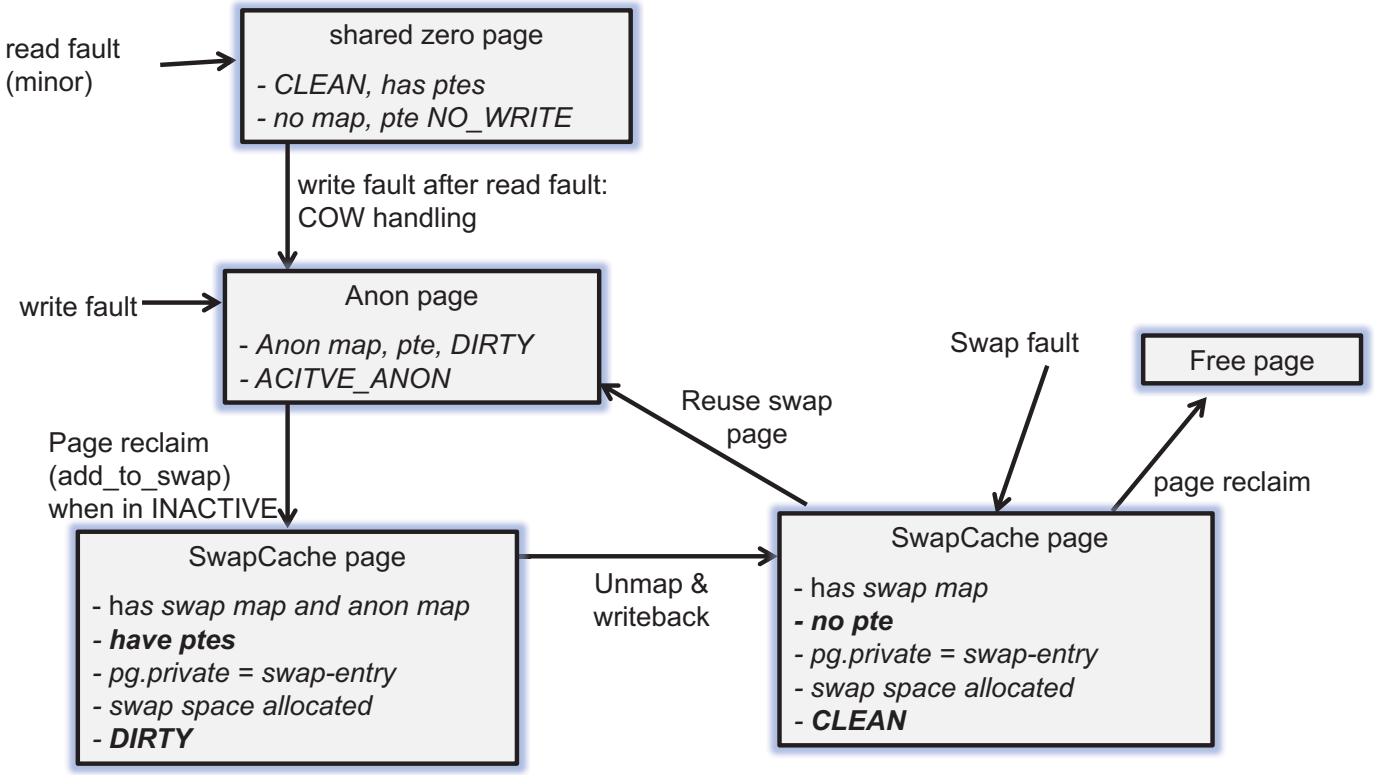
Text Page's Journey



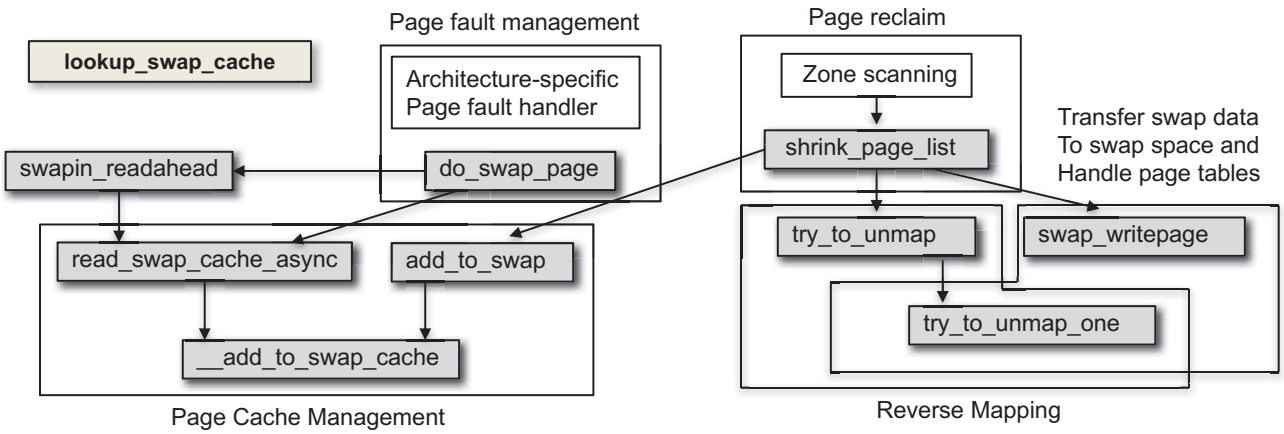
Data Page's Journey

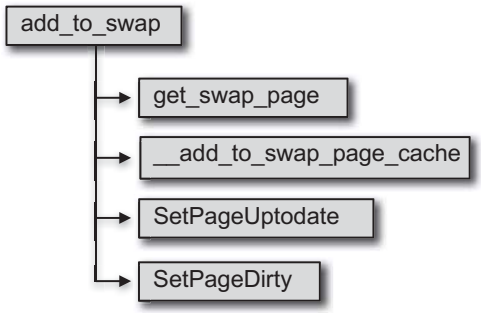


Anon Page's Journey



Adding/Finding Pages in Swap Cache





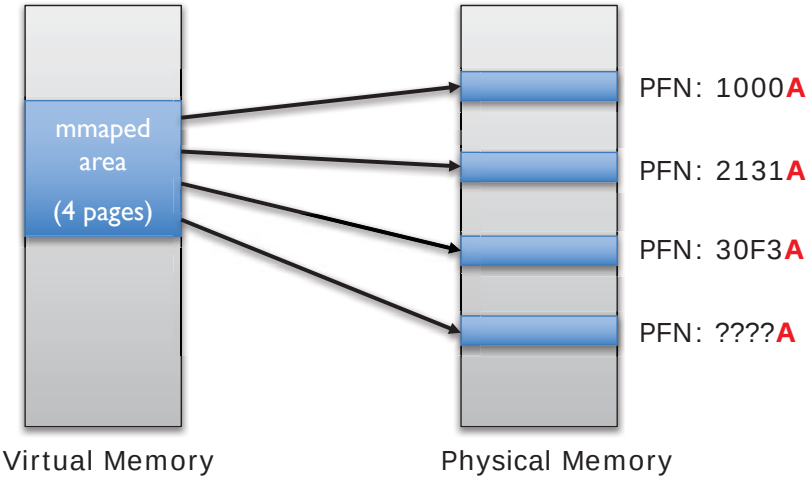
Source: Professional Linux® Kernel Architecture, Wolfgang Mauerer (2.6), Wiley Publishing, Inc.

Q&A

Practice: Controlling Physical Memory Allocation

Overview

- Control physical memory allocation
- In this practice
  - Assign a physical page of a specific modulo 16 value
  - In this example, the module 16 of PFNs is 0xA (10_{10})

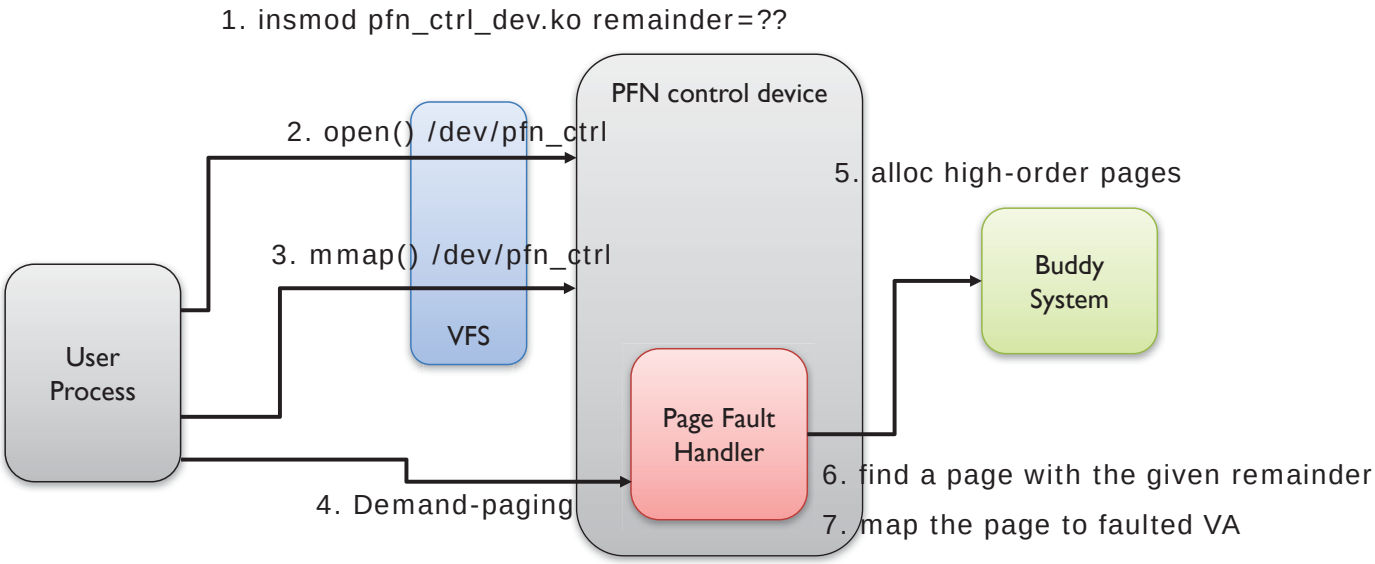


PFN in Modulo 16

- Two components
  - pfn_ctrl_dev.ko kernel module
    - Creates a device file (/dev/pfn_ctrl)
    - Assign a physical page upon a page fault
  - pfn_test.c C program
    - Open and mmap /dev/pfn_ctrl
    - Touch an mmaped area
    - Check PFNs of the mmaped area

PFN in Modulo 16

- Workflow



pfn_ctrl_dev.c Kernel Module (1)

```
static struct miscdevice pfn_ctrl_dev = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = "pfn_ctrl",
    .fops = &pfn_ctrl_fops
};


static int __init pfn_ctrl_dev_init(void)
{
    int err;


    err = misc_register(&pfn_ctrl_dev);
    if ( err ) {
        pr_err("cannot register pfn_ctrl_dev\n");
        return err;
    }
    pr_info("hello, pfn_ctrl_dev is ready\n");
    return 0;
}
```

pfn_ctrl_dev.c Kernel Module (2)

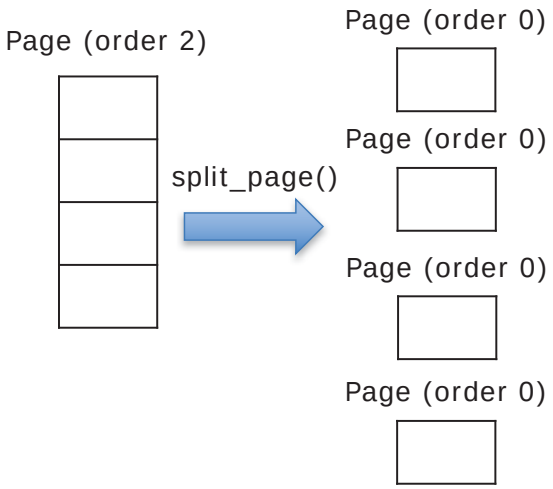
```
static int pfn_ctrl_open(struct inode* inode, struct file* file)
{
    pr_info("open success\n");
    __module_get(THIS_MODULE);
    return 0;
}
static int pfn_ctrl_release(struct inode* inode, struct file* file)
{
    pr_info("closed\n");
    module_put(THIS_MODULE);
    return 0;
}
static const struct file_operations pfn_ctrl_fops = {
    .owner = THIS_MODULE,
    .open = pfn_ctrl_open,
    .release = pfn_ctrl_release,
    .mmap = pfn_ctrl_mmap,
};
```


pfn_ctrl_dev.c Kernel Module (3)

```
static const struct vm_operations_struct pfn_ctrl_vm_ops = {
        .fault  = pfn_ctrl_fault,
};


static int pfn_ctrl_mmap(struct file* file, struct vm_area_struct* vma)
{
    vma->vm_ops = &pfn_ctrl_vm_ops;
    pr_info("mmap success\n");
    return 0;
}
```

pfn_ctrl_dev.c Kernel Module (4)

```
static int pfn_ctrl_fault(struct vm_area_struct* vma, struct vm_fault *vmf)
{
    int i;
    int ret;
    struct page* page;

    page = alloc_pages(GFP_HIGHUSER, 4);
    if ( page == NULL ) {
        ret = VM_FAULT_ERROR;
        goto exit;
    }

    split_page(page, 4);
```



pfn_ctrl_dev.c Kernel Module (5)

```
    for ( i = 0; i < 16; ++i ) {
        if ( i == remainder ) {
            vmf->page = page + i;
            pr_info("page allocated vaddr=%p pfn=%lx remainder=%d\n",
                vmf->virtual_address, page_to_pfn(page + i), remainder);
        } else
            __free_pages(page + i, 0);
    }

    ret = 0;
exit:
    return ret;
}
```

test_pfn.c (1)

- User-level application to mmap a virtual address range and print PFNs
  - open(/dev/pfn_ctrl)
  - mmap() to map a virtual memory region
  - Touch mmaped region (to cause page fault)
  - Read PFNs of mmapped region
    - Via /proc/[pid]/pagemap
    - But, read_pfn() function is exported and details are hidden
  - Usage
    - ./test_pfn nr_pages

test_pfn.c (2)

```
int main(int argc, char* argv[])
{
    void* ptr;
    int i, fd, nr_pages;


    if ( argc < 2 ) {
        printf("Usage: %s nr_pages\n", argv[0]);
        return 1;
    }
    nr_pages = strtoul(argv[1], NULL, 10);


    /* open /dev/pfn_ctrl */
    fd = open("/dev/pfn_ctrl", O_RDWR);
    if ( fd == -1 ) {
        perror("open /dev/pfn_ctrl");
        return 1;
    }
```

test_pfn.c (3)

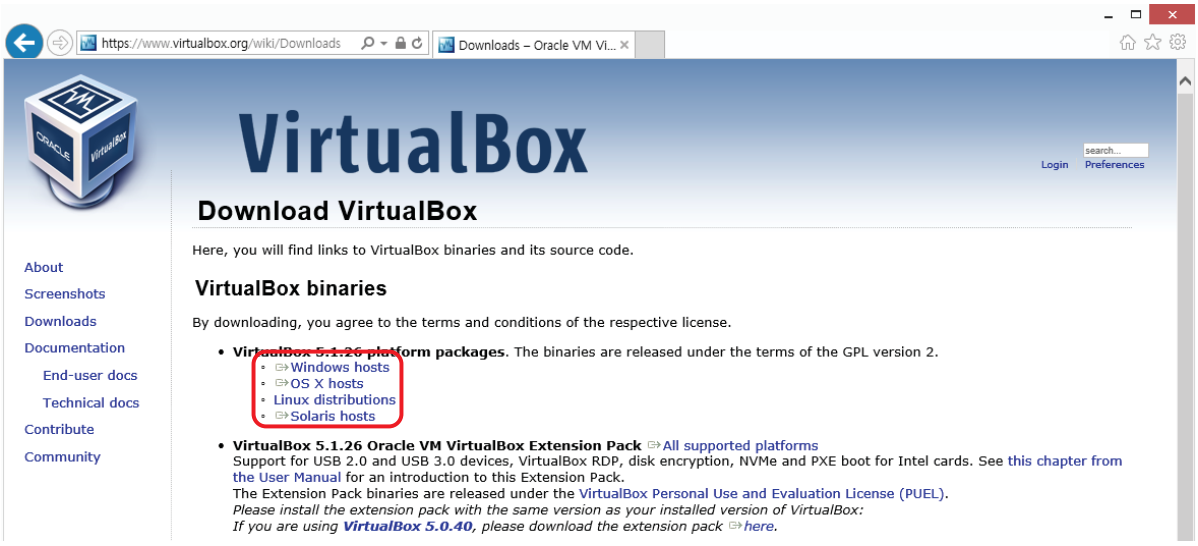
```
    /* mmap nr_pages * PAGE_SIZE */
    ptr = mmap(NULL, nr_pages * PAGE_SIZE, PROT_READ | PROT_WRITE,
                    MAP_SHARED, fd, 0);
    if ( ptr == MAP_FAILED ) {
        perror("mmap to /dev/pfn_ctrl");
        return 1;
    }

    printf("mmaped area %p~%p\n", ptr, ptr + nr_pages * PAGE_SIZE);


    /* touching mmapped area causing pfn_ctrl_fault() */
    for ( i = 0; i < nr_pages * PAGE_SIZE; i += PAGE_SIZE )
        *(int*)(ptr + i) = i;
```


test_pfn.c (4)

```
    /* printing pfns of the mmaped area */
    for ( i=0; i < nr_pages * PAGE_SIZE; i += PAGE_SIZE )
        printf("%p: %lx\n", ptr + i, read_pfn(ptr + i));


    return 0;
}
```

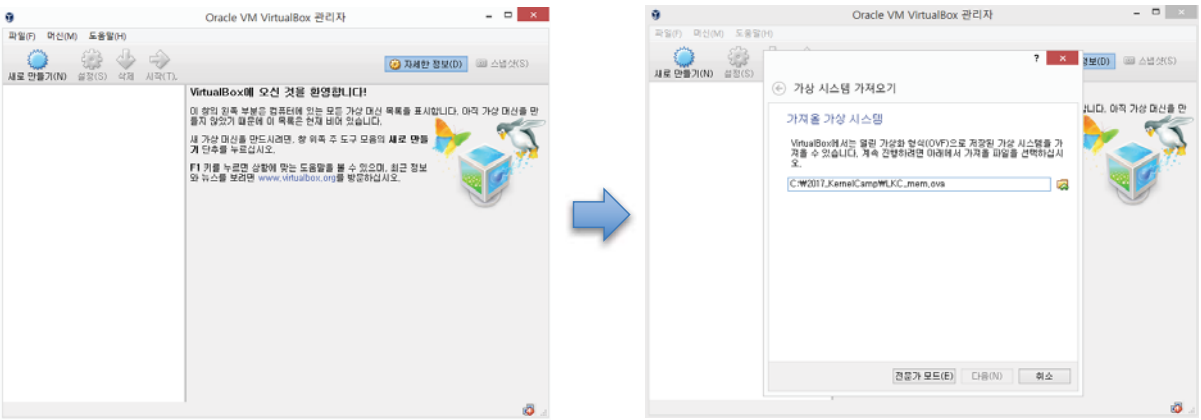
Practice in Virtual Box (1)

- Step 1: download & install VirtualBox
  - https://www.virtualbox.org/wiki/Downloads



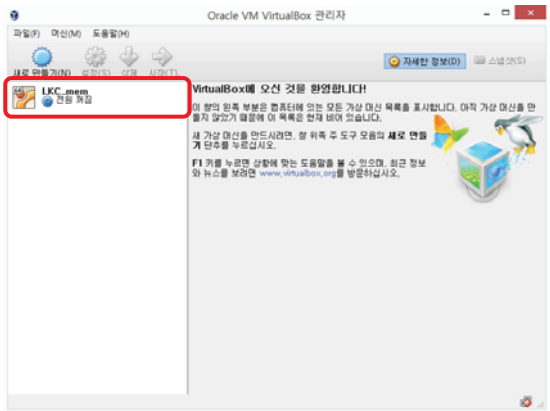
Practice in Virtual Box (2)

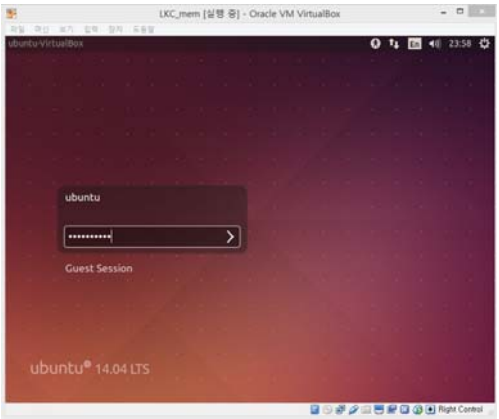
- Step 2: download VM image file
  - URL
- Step 3: run VirtualBox & import VM image
  - Press `ctrl + I`
  - Bring `LKC_mem.ova`



Practice in Virtual Box (3)

- Step 4: open VM & login
  - ID: ubuntu
  - Password: kernelcamp





Practice in Virtual Box (4)

- Step 5: run terminal
  - Click start menu and type in terminal
- Step 6: move to pfn_control directory
  - # sudo –s; # cd ~/pfn_control
- Step 7: make kernel module
  - # cd pfn_ctrl_dev; make; cd ../
- Step 8: load pfn_ctrl_dev.ko kernel module
  - # insmod pfn_ctrl_dev/pfn_ctrl_dev.ko remainder=??
- Step 9: run applications
  - # make
  - # ./test_pfn [nr_pages]
- Step 10: finish using module
  - # rmmod pfn_ctrl_dev.ko

Practice Result Example

Trouble Shooting

- Screen size
  - # sudo apt-get install virtualbox-guest-dkms
  - # sudo reboot

Storage, File System, I/O

- 리눅스 커널에서 다양한 파일 시스템들을 지원하기 위한 추상화 (abstraction) 이해
- 파일 시스템 (Layered architecture: system call, vfs, etc) 개념적 구조 이해
- 파일 시스템 소스수준 분석 필요성 및 구조
- FAT, F2FS 등 실제 파일 시스템 내부 구조 이해
- [실습] 파일시스템 FAT 및 F2FS 분석 및 구동 실습
- [실습] 파일시스템 성능 신뢰성 향상 기술

최종무 교수

(단국대학교)

Linux Kernel Camp 2017
- Part III: File system -

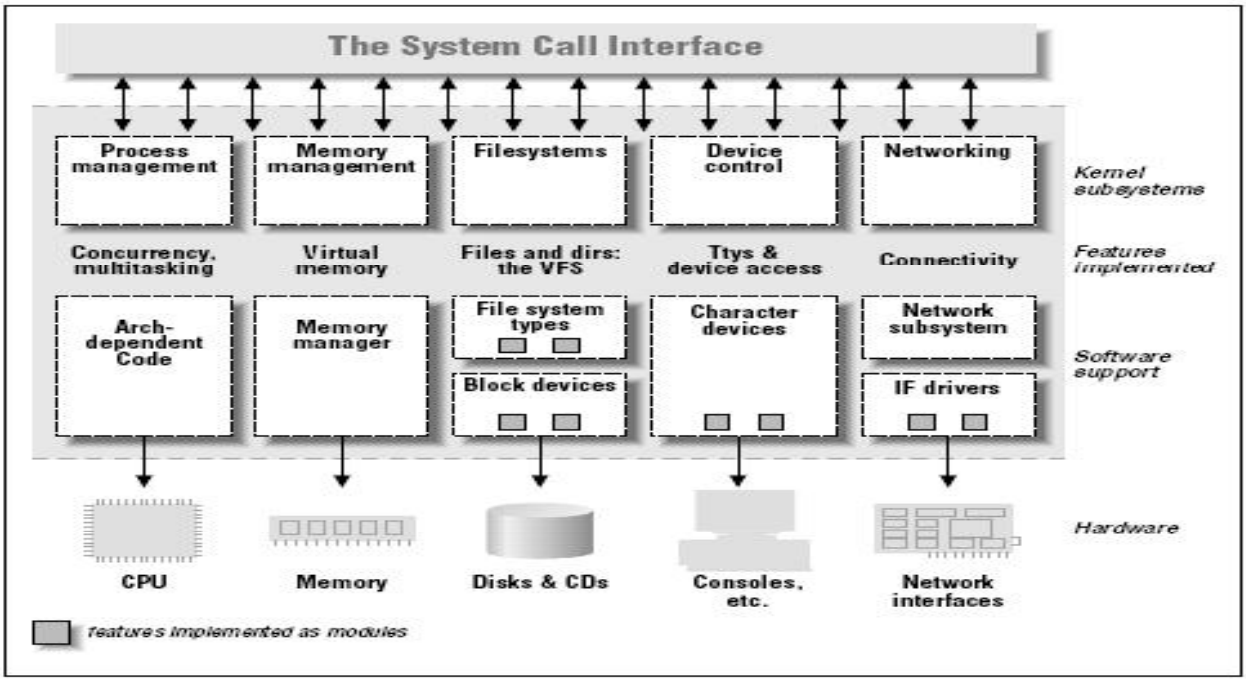
August 22, 2017.
Dankook University
Jongmoo Choi

Contents

- Introduction
  - ✓ Linux Kernel Overview
  - ✓ Why do we learn file systems at the source code level?
- Logical structure of file system
  - ✓ Layout
  - ✓ Case: UFS, Ext2 (FFS), Journaling, Ext4, FAT, LFS, F2FS, …
- File system abstraction
  - ✓ Interface
  - ✓ Case: System call, VFS, Specific FS, Page Cache, Block layer, Driver, …
- Discussion
  - ✓ How to make a new file system?
- File system Practice
  - ✓ 1) File system modification (FAT or F2FS or Ext3/4), 2) Performance evaluation, 3) Tools

Introduction

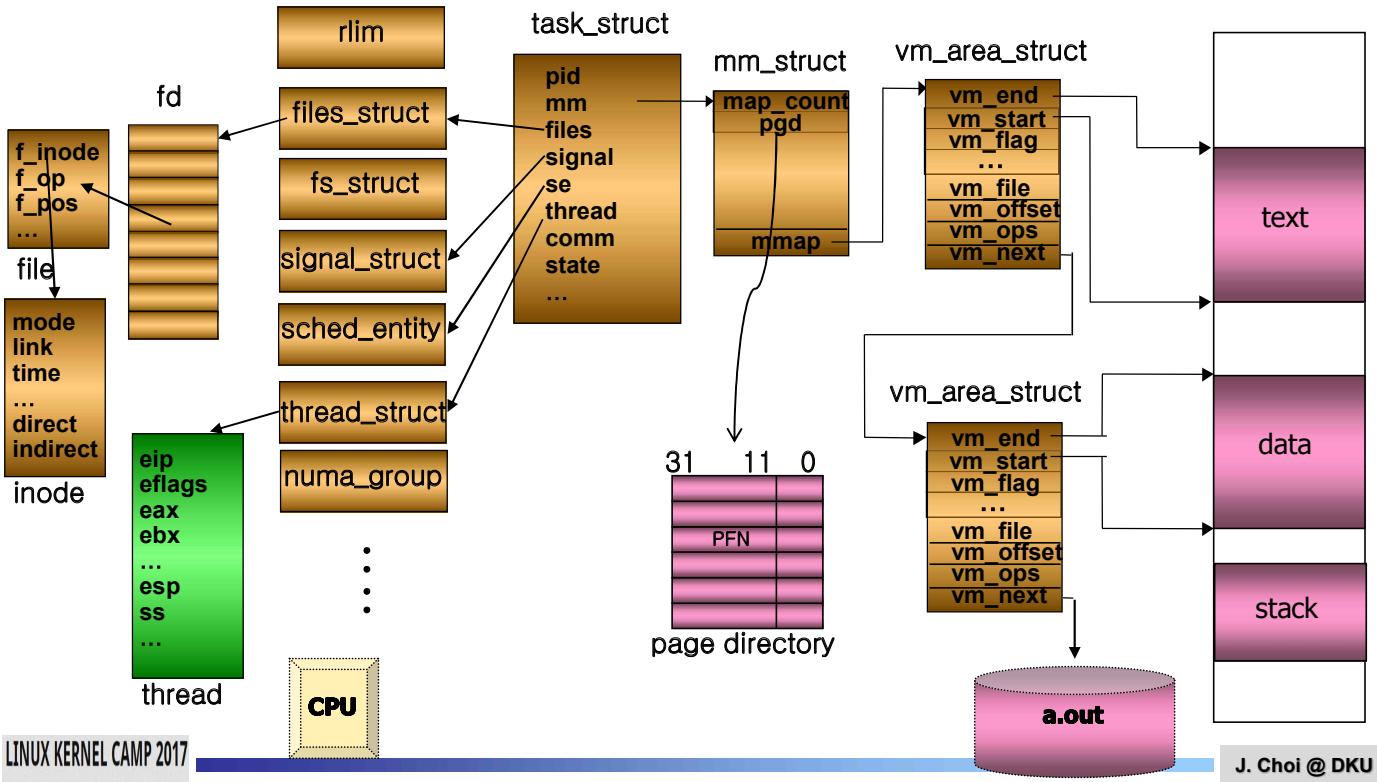
- Linux kernel: logical view
  - ✓ Overall structure



(Source: J. Corbet et al., Linux Device Driver, O'Reilly)

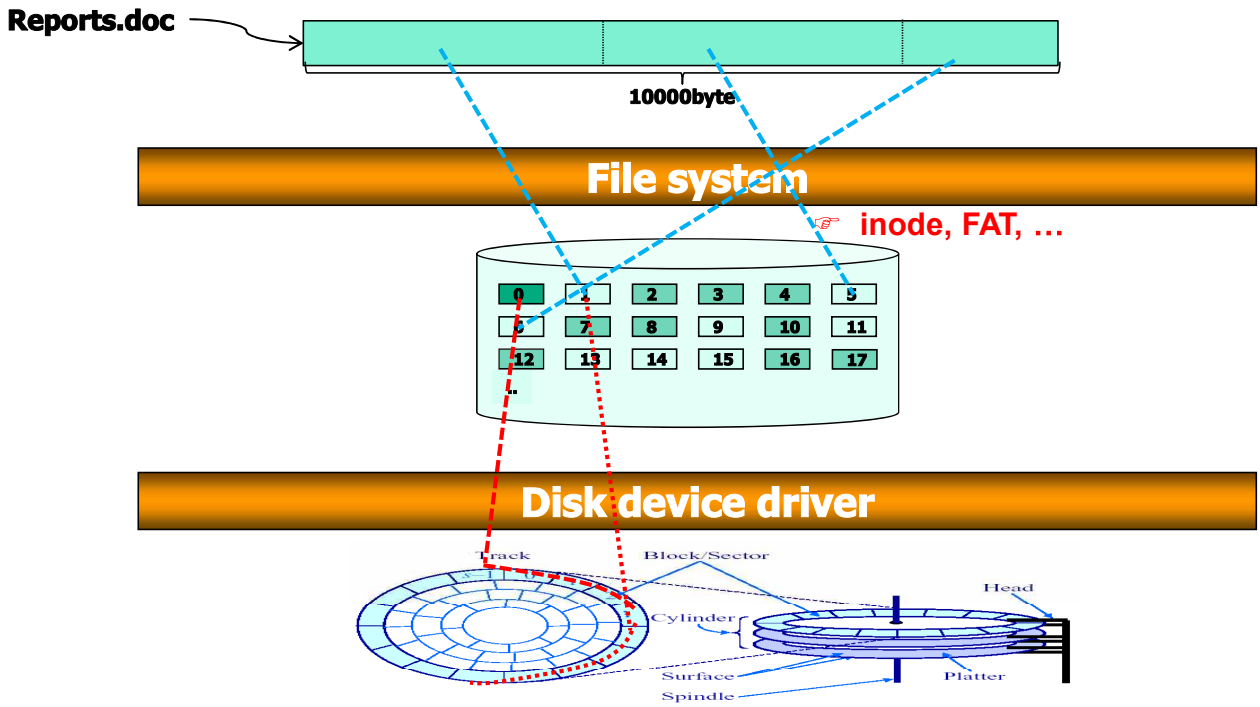
Introduction

- Linux kernel: logical view
  - ✓ Task structure (process control block)



Introduction

- Linux kernel: logical view
  - ✓ File system and Device driver

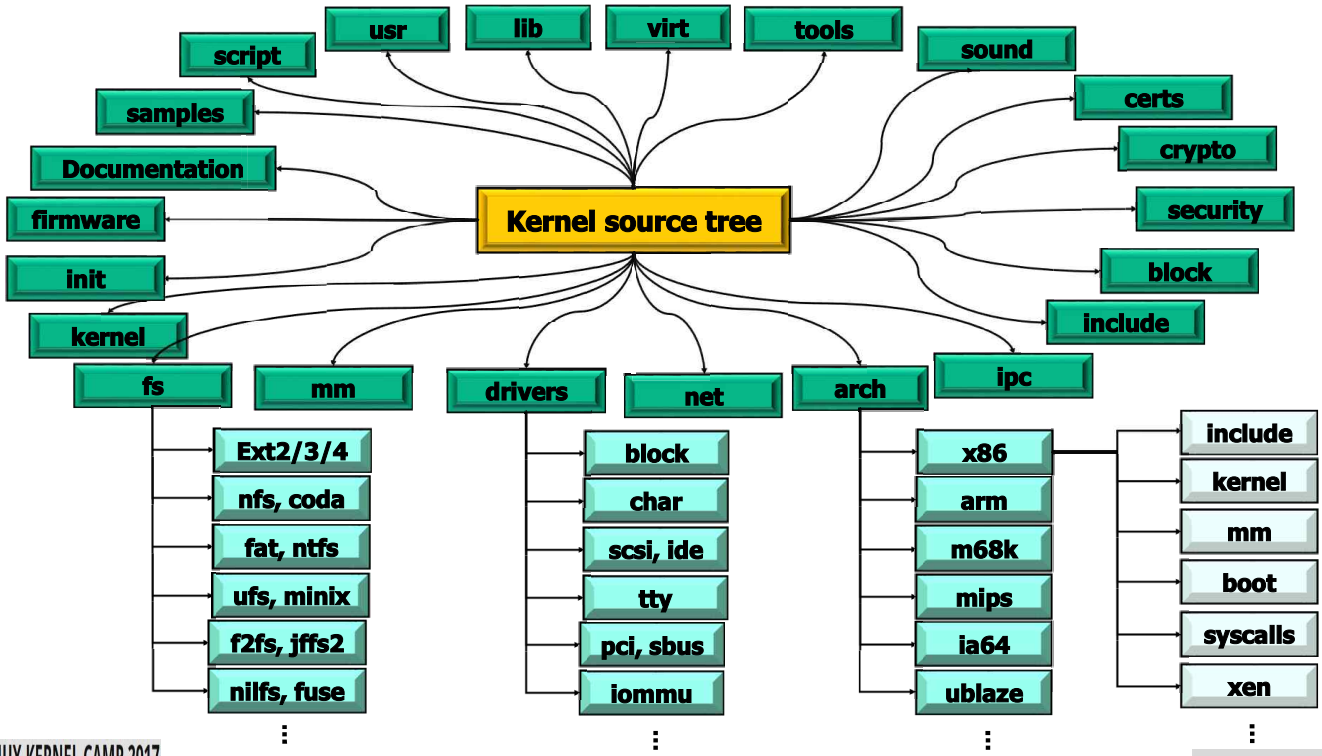


Introduction

- Linux kernel: code-level view

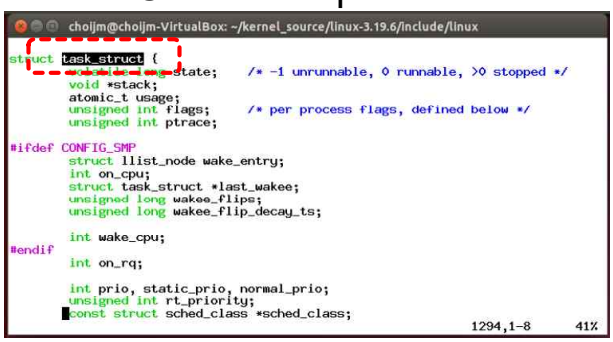
Introduction

- Linux kernel: code-level view
  - ✓ Directories in detail



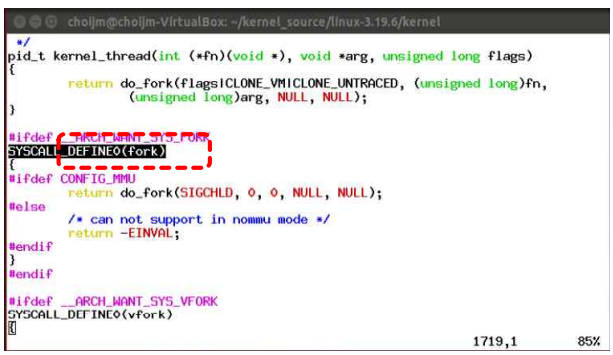
Introduction

- Linux kernel: code-level view
  - ✓ Code examples

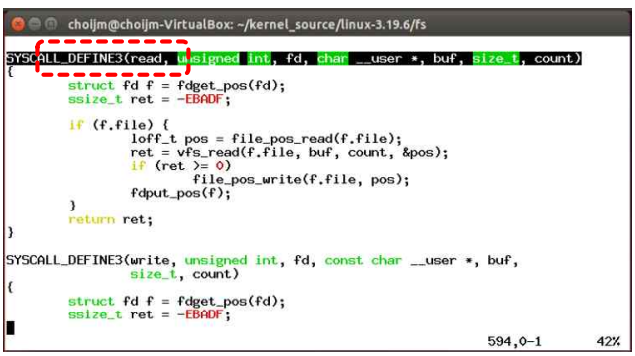


<include/linux/sched.h>

<include/linux/fs.h>



<kernel/fork.c>



<fs/read_write.c>

Introduction

- Linux kernel: code-level view
  - ✓ Code analysis: sys_mount

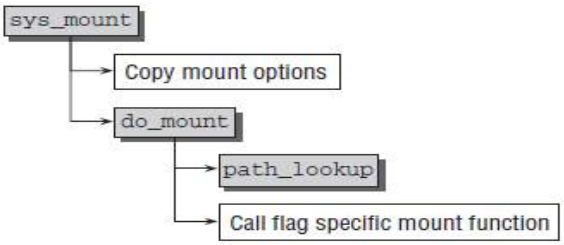


Figure 8-5: Code flow diagram for **sys_mount**.

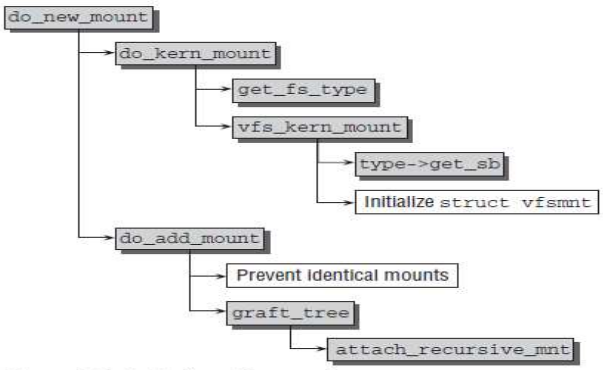


Figure 8-6: Code flow diagram for **do_new_mount**.

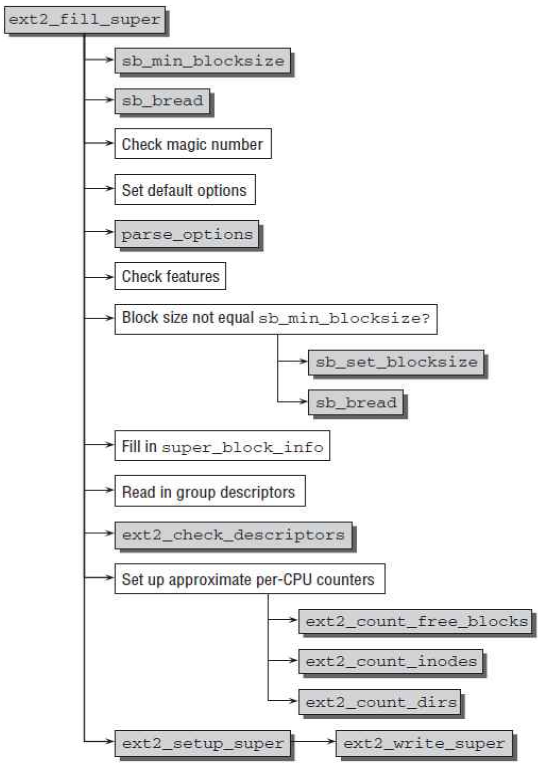
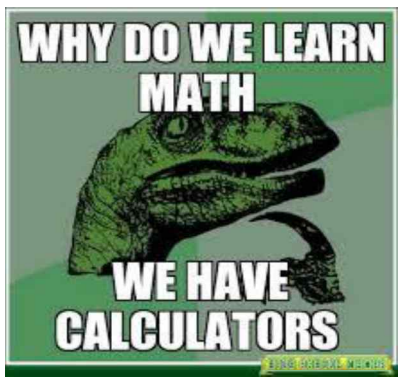


Figure 9-8: Code flow diagram for **ext2_fill_super**.

(Source: W. Mauerer, Professional Linux Kernel Architecture, Wrox)

Introduction

- Linux kernel is quite complex
- Understanding Linux kernel is a time-consuming job
- Why do we learn Linux kernel at the code level?



Why do we learn Linux kernel at the code level?

- Visit four papers
  - ✓ IRON file system
    - V. Prabhakaran, L. Bairavasundaram, N. Agrawal, H. Gunawi, A. Arpaci-dusseau and R. Arpaci-Dusseau
    - SOSP, 2005
  - ✓ F2FS (Flash-Friendly File System)
    - C. Lee, D. Sim, J. Hwang, and S. Cho
    - FAST, 2015
  - ✓ Understanding Manycore Scalability of File Systems
    - C. Min, S. Kashyap, S. Maass, W. Kang and T. Kim
    - ATC, 2016
  - ✓ Application Crash Consistency and Performance with CCFS
    - T. S. Pillai, R. Alagappan, L. Lu, V. Chidambaram, A. Arpaci-dusseau and R. Arpaci-Dusseau
    - FAST, 2017

IRON file system

- Motivation
  - ✓ How do the exist file systems react to storage failures? (not only fail-stop but also transient errors such as data corruption)
  - ✓ How can we make file systems more fault-resilient? (file systems with IRON: Internal RObustNess)
- Proposal
  - ✓ Fault injection
  - ✓ Detection strategies
  - ✓ Recovery strategies
  - ✓ Propose ixt3 (IRON ext3)

Level	Technique	Comment
D_{Zero}	No detection	Assumes disk works
$D_{ErrorCode}$	Check return codes from lower levels	Assumes lower level can detect errors
D_{Sanity}	Check data structures for consistency	May require extra space per block
$D_{Redundancy}$	Redundancy over one or more blocks	Detect corruption in end-to-end way

Table 1: The Levels of the IRON Detection Taxonomy.

Level	Technique	Comment
R_{Zero}	No recovery	Assumes disk works
$R_{Propagate}$	Propagate error	Informs user
R_{Stop}	Stop activity (crash, prevent writes)	Limit amount of damage
R_{Guess}	Return "guess" at block contents	Could be wrong; failure hidden
R_{Retry}	Retry read or write	Handles failures that are transient
R_{Repair}	Repair data structs	Could lose data
R_{Remap}	Remaps block or file to different locale	Assumes disk informs FS of failures
$R_{Redundancy}$	Block replication or other forms	Enables recovery from loss/corruption

Table 2: The Levels of the IRON Recovery Taxonomy.

(Source: Iron file system, SOSP'05)

IRON file system

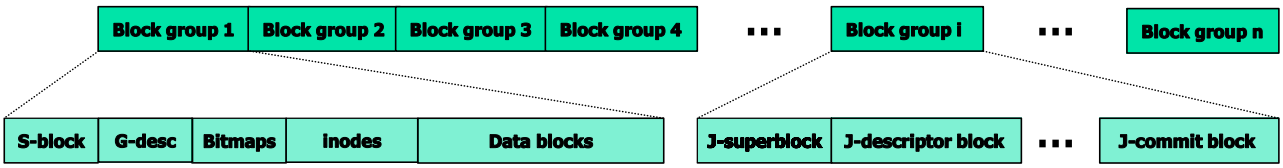
- Analysis results
  - ✓ Detection and Recovery capabilities on 3 file systems

Figure 2: File System Failure Policies. The tables indicate both detection and recovery policies of ext3, ReiserFS, and JFS for read, write, and corruption faults injected for each block type across a range of workloads. The workloads are a: path traversal b: access,chdir,chroot,stat,statfs,lstat,open c: chmod,chown,utimes d: read e: readlink f: getdirentries g: creat h: link i: mkdir j: rename k: symlink l: write m: truncate n: rmdir o: unlink p: mount q: fsync,sync r: umount s: FS recovery t: log write operations. A gray box indicates that the workload is not applicable for the block type. If multiple mechanisms are observed, the symbols are superimposed.

(Source: Iron file system, SOSP'05)

IRON file system

- How to materialize?
  - ✓ Type-aware faults injections: Consider file system layout
    - Ext3 case

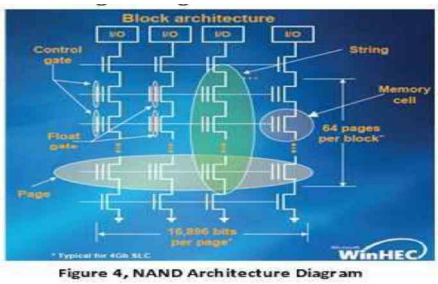


  - ✓ File system aware detection/recovery analysis: Consider file system interface
    - **a**: path traverse, **b**: access, stat, open, chdir, **c**: chmod, chown
    - **d**: read, **f**: getdirentries, **g**: creat, **h**: link, **i**: mkdir,
    - **j**: rename, **k**: symlink, **l**: write, **m**: truncate, **n**: rmdir,
    - **o**: unlink, **p**: mount, **q**: fsync, sync, **r**: umount,
    - **s**: FS recovery, **t**: log write operations

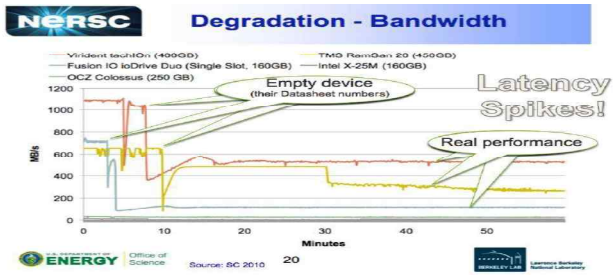
☞ File system layout and interface are the main topics of this lecture

F2FS

- Motivation
  - ✓ Want to make a file system optimized for flash memory
  - ✓ Flash characteristics
    - Erase-before write requirement: page vs. block
    - Limited P/E cycles: limited lifespan
    - Diverse errors: retention, disturbance, FTL related, …
  - ✓ Performance considerations
    - Multi-channel/way architecture (become larger in NVMe SSDs)
    - Reduce garbage collection overhead (write cliff)
      - Especially for random writes



(Source: www.simmtester.com)



(Source: www.theregister.co.uk)

F2FS

- Proposal
  - ✓ Layout
    - Three configurable units: zone, section, segment ➔ aware of the erase unit of Flash memory (alignment)
    - Six logical areas: SB (FS info), CP(consistent status), SIT(information), NAT (node blocks location), SSA (pointers for node/data used for cleaning), Main area (node and data)
  - ✓ Node block: inode, direct, indirect node
    - Wandering tree problem ➔ direct node and NAT update only
  - ✓ Multi-head logging
    - Apply logging so that FTL do switch merge in most case (like LFS)
    - Six log areas: hot/cold separation
  - ✓ Cleaning (Garbage collection)
    - Foreground vs background, greedy vs cost-benefit
  - ✓ Adaptive logging (high utilization)
    - Normal logging vs. threaded logging

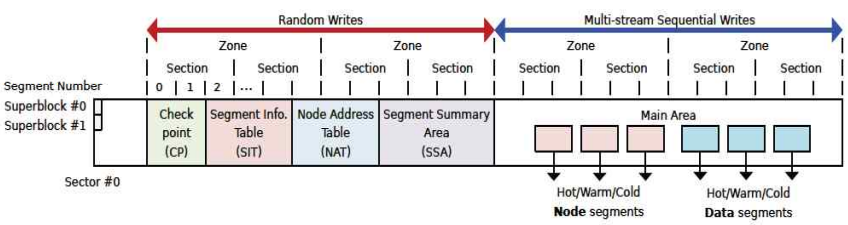


Figure 1: On-disk layout of F2FS.

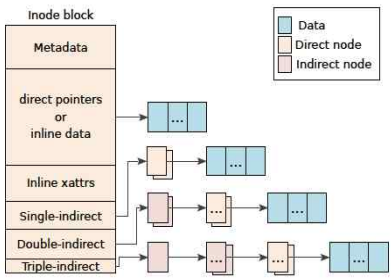


Figure 2: File structure of F2FS.

(Source: F2FS, FAST'15)

F2FS

- How to materialize?

```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/f2fs
choijm@choijm-VirtualBox:/usr/src/linux-3.19.6/fs/f2fs$
choijm@choijm-VirtualBox:/usr/src/linux-3.19.6/fs/f2fs$ pwd
/usr/src/linux-3.19.6/fs/f2fs
choijm@choijm-VirtualBox:/usr/src/linux-3.19.6/fs/f2fs$ ls -l
합계 5232
-rw-rw-r-- 1 choijm choijm    2499  4월 29  2015 Kconfig
-rw-rw-r-- 1 choijm choijm     282  4월 29  2015 Makefile
-rw-rw-r-- 1 choijm choijm    9068  4월 29  2015 acl.c
-rw-rw-r-- 1 choijm choijm    1251  4월 29  2015 acl.h
-rw-rw-r-- 1 choijm choijm   27778  4월 29  2015 checkpoint.c
-rw-rw-r-- 1 choijm choijm   28256  4월 29  2015 data.c
-rw-rw-r-- 1 choijm choijm   11981  4월 29  2015 debug.c
-rw-rw-r-- 1 choijm choijm   19400  4월 29  2015 dir.c
-rw-rw-r-- 1 choijm choijm   50554  4월 29  2015 f2fs.h
-rw-r--r-- 1 root   root   4834153  5월  4  2015 f2fs.ko
-rw-r--r-- 1 root   root     14611  5월  4  2015 f2fs.mod.c
-rw-rw-r-- 1 choijm choijm   24632  4월 29  2015 file.c
-rw-rw-r-- 1 choijm choijm   18810  4월 29  2015 gc.c
-rw-rw-r-- 1 choijm choijm    3181  4월 29  2015 gc.h
-rw-rw-r-- 1 choijm choijm    2142  4월 29  2015 hash.c
-rw-rw-r-- 1 choijm choijm   12586  4월 29  2015 inline.c
-rw-rw-r-- 1 choijm choijm   10042  4월 29  2015 inode.c
-rw-r--r-- 1 root   root       23  5월  4  2015 modules.order
-rw-rw-r-- 1 choijm choijm   17560  4월 29  2015 namei.c
-rw-rw-r-- 1 choijm choijm   51226  4월 29  2015 node.c
-rw-rw-r-- 1 choijm choijm   10922  4월 29  2015 node.h
-rw-rw-r-- 1 choijm choijm   13705  4월 29  2015 recovery.c
-rw-rw-r-- 1 choijm choijm   59747  4월 29  2015 segment.c
-rw-rw-r-- 1 choijm choijm   22682  4월 29  2015 segment.h
-rw-rw-r-- 1 choijm choijm   32344  4월 29  2015 super.c
-rw-rw-r-- 1 choijm choijm   14988  4월 29  2015 xattr.c
-rw-rw-r-- 1 choijm choijm    4600  4월 29  2015 xattr.h
choijm@choijm-VirtualBox:/usr/src/linux-3.19.6/fs/f2fs$
```

For configuration
For build
Access control list
Make checkpoint
Data management,
Directory management
f2fs main header
File interface
Garbage collection management
inode management
Directory traverse
Node management
Rollback/forward recovery
Super block management
Segment management

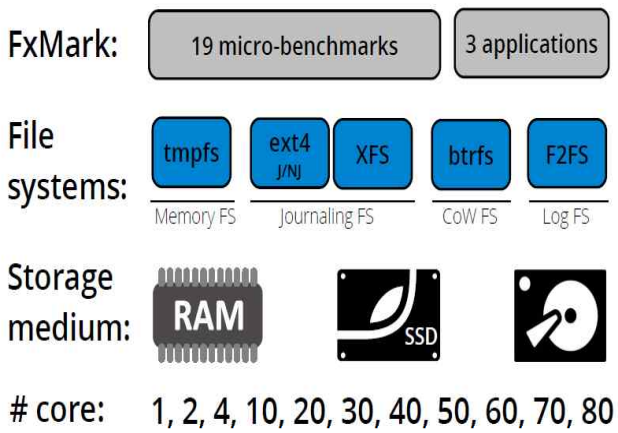
F2FS

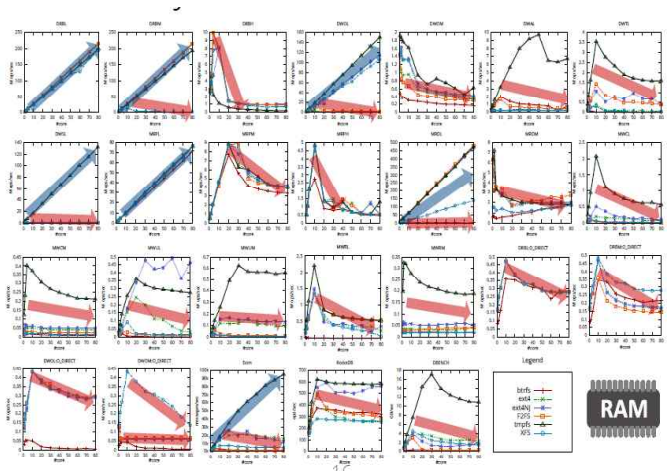
- How to materialize?
  - ✓ Kernel version 4.12 (still evolved: performance, reliability, security)

```
choijm@choijm-VirtualBox: ~/linux-4.12.2/fs/f2fs
choijm@choijm-VirtualBox:~/linux-4.12.2/fs/f2fs$ pwd
/home/choijm/linux-4.12.2/fs/f2fs
choijm@choijm-VirtualBox:~/linux-4.12.2/fs/f2fs$ ls -l
합계 732
-rw-rw-r-- 1 choijm choijm  3285  7월 15 20:09 Kconfig
-rw-rw-r-- 1 choijm choijm   359  7월 15 20:09 Makefile
-rw-rw-r-- 1 choijm choijm  9061  7월 15 20:09 acl.c
-rw-rw-r-- 1 choijm choijm  1210  7월 15 20:09 acl.h
-rw-rw-r-- 1 choijm choijm 34442  7월 15 20:09 checkpoint.c
-rw-rw-r-- 1 choijm choijm 53431  7월 15 20:09 data.c
-rw-rw-r-- 1 choijm choijm 17353  7월 15 20:09 debug.c
-rw-rw-r-- 1 choijm choijm 22326  7월 15 20:09 dir.c
-rw-rw-r-- 1 choijm choijm 19546  7월 15 20:09 extent_cache.c
-rw-rw-r-- 1 choijm choijm 85530  7월 15 20:09 f2fs.h
-rw-rw-r-- 1 choijm choijm 56206  7월 15 20:09 file.c
-rw-rw-r-- 1 choijm choijm 26187  7월 15 20:09 gc.c
-rw-rw-r-- 1 choijm choijm  2847  7월 15 20:09 gc.h
-rw-rw-r-- 1 choijm choijm  2231  7월 15 20:09 hash.c
-rw-rw-r-- 1 choijm choijm 16002  7월 15 20:09 inline.c
-rw-rw-r-- 1 choijm choijm 12689  7월 15 20:09 inode.c
-rw-rw-r-- 1 choijm choijm 26320  7월 15 20:09 namei.c
-rw-rw-r-- 1 choijm choijm 68363  7월 15 20:09 node.c
-rw-rw-r-- 1 choijm choijm 12423  7월 15 20:09 node.h
-rw-rw-r-- 1 choijm choijm 14996  7월 15 20:09 recovery.c
-rw-rw-r-- 1 choijm choijm 86688  7월 15 20:09 segment.c
-rw-rw-r-- 1 choijm choijm 24502  7월 15 20:09 segment.h
-rw-rw-r-- 1 choijm choijm  3347  7월 15 20:09 shrinker.c
-rw-rw-r-- 1 choijm choijm 61574  7월 15 20:09 super.c
-rw-rw-r-- 1 choijm choijm  3549  7월 15 20:09 trace.c
-rw-rw-r-- 1 choijm choijm  1013  7월 15 20:09 trace.h
-rw-rw-r-- 1 choijm choijm 17035  7월 15 20:09 xattr.c
-rw-rw-r-- 1 choijm choijm  4850  7월 15 20:09 xattr.h
choijm@choijm-VirtualBox:~/linux-4.12.2/fs/f2fs$
```

Understanding Manycore Scalability of File Systems

- Motivation
  - ✓ How scalable file systems on Manycore?
  - ✓ If not, why they are not scalable?
- Proposal
  - ✓ FxMark: benchmark for file system scalability test
  - ✓ Valuable observations (low/medium/high-level sharing)





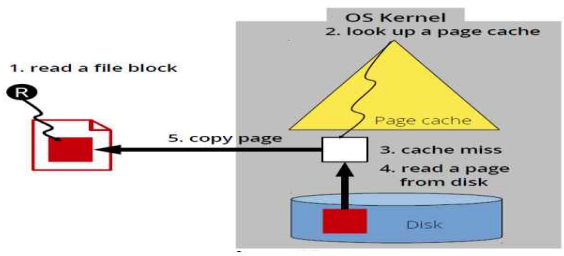
(Source: Understanding Manycore Scalability of File Systems, ATC'16)

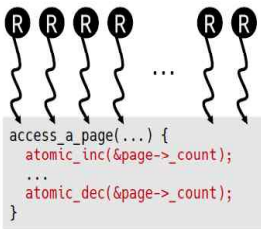
Understanding Manycore Scalability of File Systems

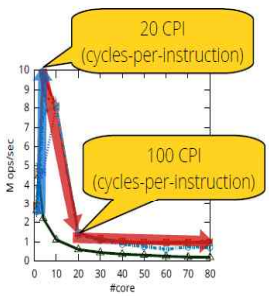
- Reasoning
  - ✓ 25 scalability bottlenecks: Table 3
  - ✓ Case study
    - Page cache: utilize temporal locality
    - Replacement: based on page reference counts (shared resource)

FS	Bottleneck	Sync. object	Scope	Operation
VFS	Rename lock	rename_lock	System	rename()
	inode list lock	inode_sb_list_lock	System	File creation and deletion
	Directory access lock	inode->i_mutex	Directory	All directory operations
	Page reference counter	page->_count	Page	Page cache access
	dentry lockref [58]	dentry->d_lockref	dentry	Path name resolution
btrfs	Acquiring a write lock for a B-tree node	btrfs_tree_lock()	File system	All write operations
	Acquiring a read lock for a B-tree node	btrfs_set_lock_blocking_rw()	File system	All read operations
	Checking data free space	data_info->lock	File system	File append
	Reserving data blocks	delalloc_block_rsv->lock	File system	File append
	File write lock	inode->i_mutex	File	File write
ext4	Acquiring a read lock for a journal	journal->j_state_lock	Journal	Heavy metadata update operations
	Atomic increment of a transaction counter	t_handle_count	Journal	Heavy metadata update operations
	Orphan inode list	sbi->s_orphan	File system	File deletion
	Block group lock	bgl->locks	Block group	File creation and deletion
	File write lock	inode->i_mutex	File	File write
F2FS	Single-threaded writing	sbi->cp_rwsem	File system	File write
	SIT (segment information table)	sit_i->sentry_lock	File system	File write
	NAT (node address table)	nm->nat_tree_lock	File system	File creation, deletion, and write
	File write lock	inode->i_mutex	File	File write
tmpfs	Cgroup page reference counter	counter->count	cgroup	File truncate
	Capacity limit check (per-CPU counter)	sbinfo->used_blocks	File system	File write near disk full
	File write lock	inode->i_mutex	File	File write
XFS	Journal writing	log->l_icloglock	File system	Heavy metadata update operations
	Acquiring a read lock of XFS inode	ip->i_iolock	File	File read
	File write lock	inode->i_mutex	File	File write

Table 3: The identified scalability bottlenecks in tested file systems with FxMark.







Understanding Manycore Scalability of File Systems

- Source level viewpoint

```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs
 */
struct dentry *lock_rename(struct dentry *p1, struct dentry *p2)
{
        struct dentry *p;

        if (p1 == p2) {
                mutex_lock_nested(&p1->d_inode->i_mutex, I_MUTEX_PARENT);
                return NULL;
        }
        mutex_lock(&p1->d_inode->i_sb->s_vfs_rename_mutex);

        p = d_ancestor(p2, p1);
        if (p) {
                mutex_lock_nested(&p2->d_inode->i_mutex, I_MUTEX_PARENT);
                mutex_lock_nested(&p1->d_inode->i_mutex, I_MUTEX_CHILD);
                return p;
        }

        p = d_ancestor(p1, p2);
        if (p) {
                mutex_lock_nested(&p1->d_inode->i_mutex, I_MUTEX_PARENT);
                mutex_lock_nested(&p2->d_inode->i_mutex, I_MUTEX_CHILD);
                return p;
"namei.c" 4507 줄 --55%--                                    2486,3        55
```

<fs/namei.c>

```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs
}

/**
 * inode_sb_list_add - add inode to the superblock list of inodes
 * @inode: inode to add
 */
void inode_sb_list_add(struct inode *inode)
{
        spin_lock(&inode_sb_list_lock);
        list_add(&inode->i_sb_list, &inode->i_sb->s_inodes);
        spin_unlock(&inode_sb_list_lock);
}
EXPORT_SYMBOL_GPL(inode_sb_list_add);

static inline void inode_sb_list_del(struct inode *inode)
{
        if (!list_empty(&inode->i_sb_list)) {
                spin_lock(&inode_sb_list_lock);
                list_del_init(&inode->i_sb_list);
                spin_unlock(&inode_sb_list_lock);
        }
}

static unsigned long hash(struct super_block *sb, unsigned long hashval)
"inode.c" 1931 줄 --23%--                                    453,28
```

<fs/inode.c>

```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6
 * it to keep track of whatever it is we are using the page for at the
 * moment. Note that we have no way to track which tasks are using
 * a page, though if it is a pagecache page, rmap structures can tell us
 * who is mapping it.
 *
 * The objects in struct page are organized in double word blocks in
 * order to allows us to use atomic double word operations on portions
 * of struct page. That is currently only used by slub but the arrangement
 * allows the use of atomic double word operations on the flags/mapping
 * and lru list pointers also.
 */
struct page {
        /* First double word block */
        unsigned long flags;            /* Atomic flags, some possibly
                                         * updated asynchronously */
        ...
        /* Second double word */
        struct {
                ...
                        atomic_t _count;                /* Usage count, see below. */
                };
                        unsigned int active;    /* SLAB */
        /* Third double word block */
        union {
                struct list_head lru;   /* Pageout list, eg. active_list
"include/linux/mm_types.h" [비쥬얼]읽기 전용] 473 줄 --12%--                57,2-10
```

<include/linux/mm_types.h>

```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/jbd2
        }

        jbd_debug(3, "New handle %p going live.\n", handle);

        /*
         * We need to hold j_state_lock until t_updates has been incremented,
         * for proper journal barrier handling
         */
repeat:
        read_lock(&journal->j_state_lock);
        BUG_ON(journal->j_flags & JBD2_UNMOUNT);
        if (is_journal_aborted(journal) ||
            (journal->j_errno != 0 && !(journal->j_flags & JBD2_ACK_ERR))) {
                read_unlock(&journal->j_state_lock);
                jbd2_journal_free_transaction(new_transaction);
                return -EROFS;
        }

        /*
         * Wait on the journal's transaction barrier if necessary. Specifically
         * we allow reserved handles to proceed because otherwise commit could
         * deadlock on page writeback not being able to complete.
         */
        if (!handle->h_reserved && journal->j_barrier_count) {
"transaction.c" 2480 줄 --12%--                                    305,7         12%
```

<fs/jbd2/transaction .c>

CCFS (Crash-Consistent File System)

- Motivation
  - ✓ How FS behavior affect applications crash consistency?
  - ✓ What are required for crash consistency?
  - ✓ How to obtain both consistency and performance?

File systems

Applications	Ext2-like FS	Btrfs	Ext3-DJ
LevelDB-1.10	10	4	1
LevelDB-1.15	6	3	1
LMDB	1		
GDBM	5	4	2
HSQLDB	10	4	
SQLite-Roll	1	1	1
SQLite-WAL	0		
PostgreSQL	1		
Git	9	5	2
Mercurial	10	8	3
VMWare	1		
HDFS	2	1	
ZooKeeper	4	1	
Total	60	31	10
Ordering	✗	✗	✔
Atomicity	✗	✔	✔

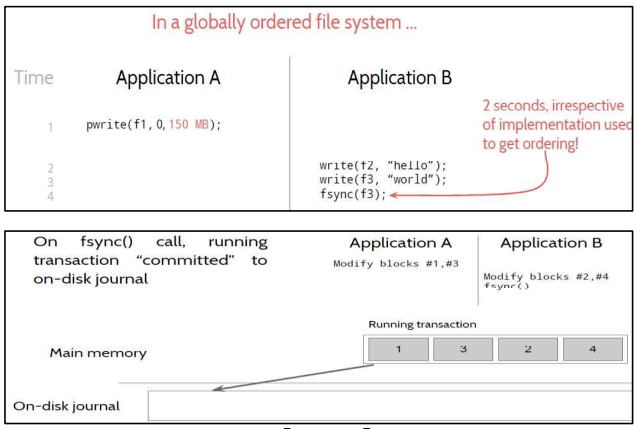
Vulnerabilities under safest application configuration

File-system behavior

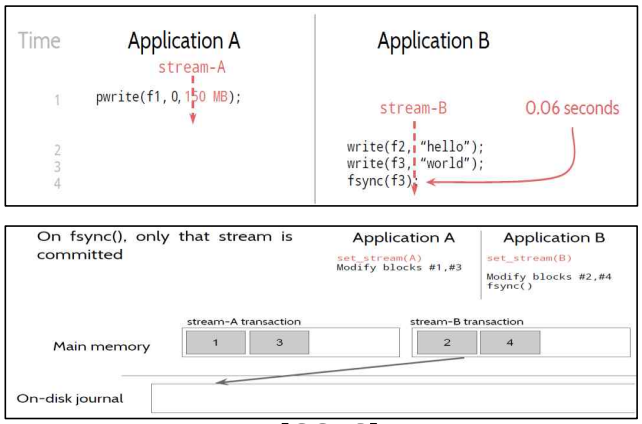
(Source: Application Crash Consistency and Performance with CCFS, FAST'17)

CCFS (Crash-Consistent File System)

- What issues for preserving ordering?
  - ✓ Performance overhead due to limited buffering and O3 execution
- Proposal
  - ✓ Idea: Remove false dependency
  - ✓ Design: new abstraction ➔ stream
  - ✓ Implementation: multiple transactions in Ext4
  - ✓ New techniques: hybrid journaling, delta journaling, Pointless Data structure, order-preserving delayed allocation, …



(Ext4)

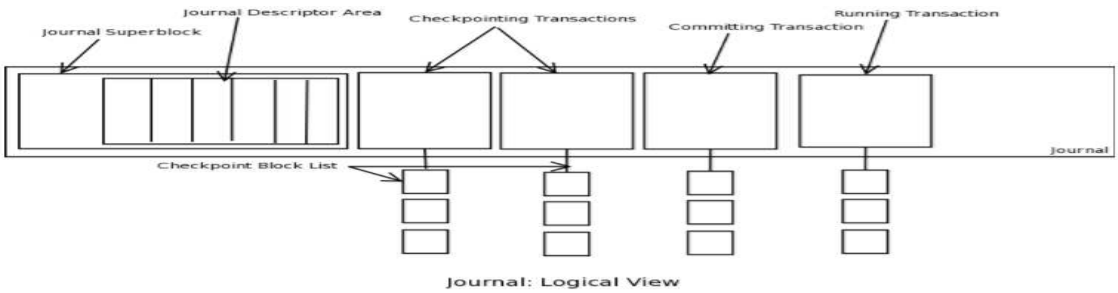


(CCFS)

(Source: Application Crash Consistency and Performance with CCFS, FAST'17)

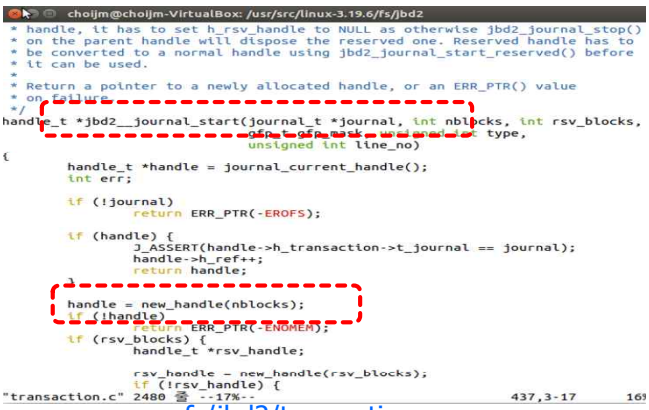
CCFS (Crash-Consistent File System)

- How to materialize?



(Source: Linux: The Journaling Block Device, http://0758jx.weebly.com/uploads/6/6/8/9/6689589/linux_the_journaling_block_device.pdf)

<fs/jbd2/commit.c>



<fs/jbd2/transaction .c>

Why do we learn Linux kernel at the code level?

- To realize our idea
- To evaluate our proposal
- To observe a certain behavior
- To explore problems and/or issues
- To make an outstanding paper
- Just for fun
- …

☞ Require expertise!!

Contents

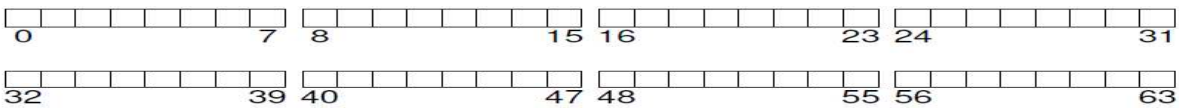
- Introduction
  - ✓ Linux Kernel Overview
  - ✓ Why do we learn file system at the source code level?
- Logical structure of file system
  - ✓ Layout
  - ✓ Case: UFS, Ext2 (FFS), Journaling, Ext4, FAT, LFS, F2FS, …
- File system abstraction
  - ✓ Interface
  - ✓ Case: System call, VFS, Specific FS, Page Cache, Block layer, …
- Discussion
  - ✓ How to make a new file system?
- File system Practice
  - ✓ 1) FAT file system (with recovery), 2) F2FS (with MTD), 3) Performance evaluation

Basic layout

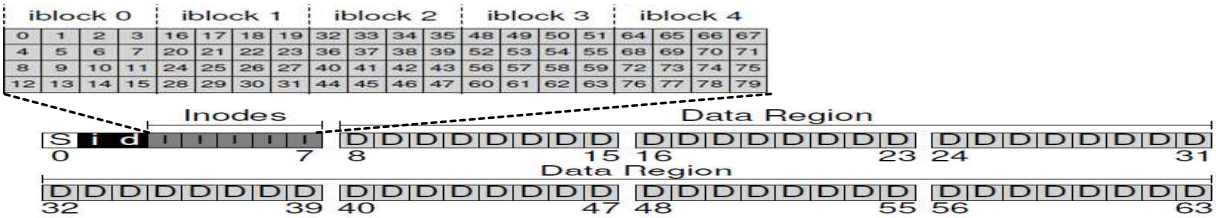
- File system
  - ✓ User's viewpoint: files and directories
  - ✓ System programmer's viewpoint: data and metadata
- Metadata
  - ✓ Information used for managing files and file systems
  - ✓ Superblock
    - a record of the characteristics of a file system
    - total size, block size, location of inodes, root directory, usage, …
  - ✓ inode
    - represent an object such as a file, a directory, a device, …
    - stores the attributes and disk block locations
  - ✓ Bitmap
    - data structure to differentiate the free/used entry of inodes/data blocks
  - ✓ Others
    - group descriptor, journal, segment, …

Basic layout

- Simple file system example
  - ✓ Assume that there are 64 disk blocks (4KB)



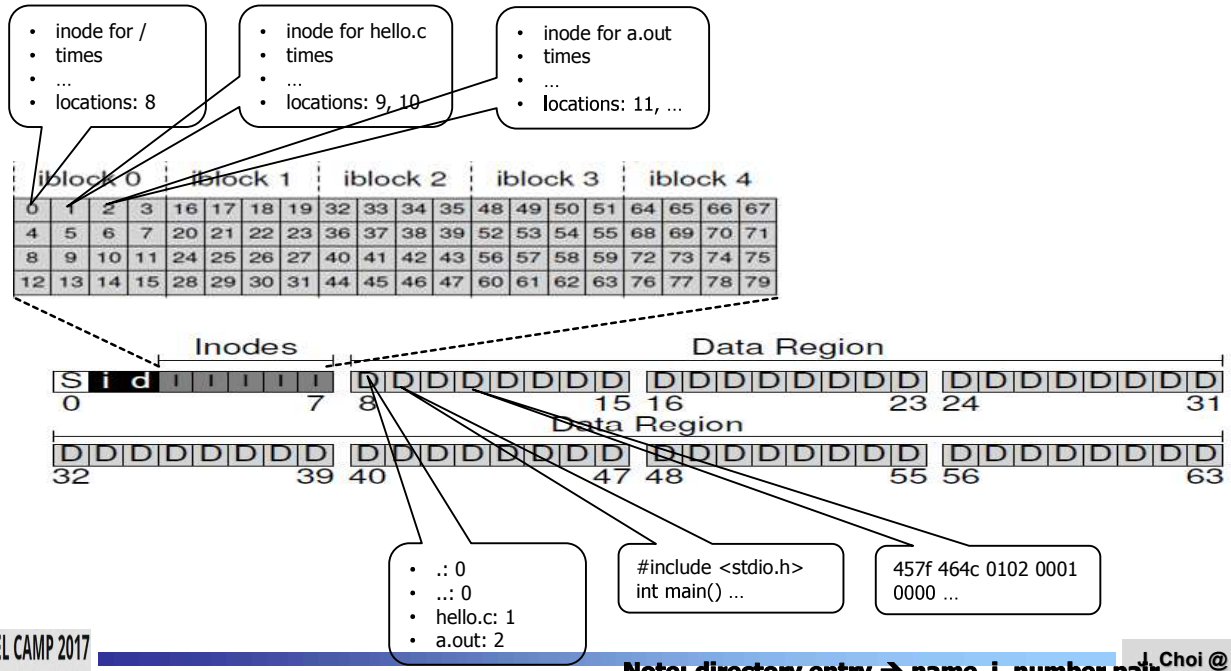
  - ✓ Usage
    - Data region for user data: from 8 to 63 (56 blocks)
    - Inode: from 3 to 7, 256byte per inode ➔ 16 inodes per block, total 80 inodes
    - Inode bitmap (i), Data bitmap (d)
    - Superblock (S): where is the inode bitmap? how many blocks? …



(Source: Remzi, "Operating Systems: Three Easy Pieces")

Basic layout

- Simple file system example (cont')
  - ✓ When we create a new file (named hello.c whose size is 7KB) in a root directory?
  - ✓ Then, we compile it?



Note: directory entry → name, i_number pair

Basic layout

- Operation example
  - ✓ Create a file "bar" in the "/foo" directory, write data (10KB) and close it
  - ✓ Timeline

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data[0]	bar data[1]	bar data[2]
			read							
						read				
				read						
							read			
create (/foo/bar)		read								
		write								
							write			
					read					
					write					
				write						
					read					
write()	read									
	write									
								write		
					write					
					read					
write()	read									
	write									
									write	
					write					
					read					
write()	read									
	write									
										write
					write					

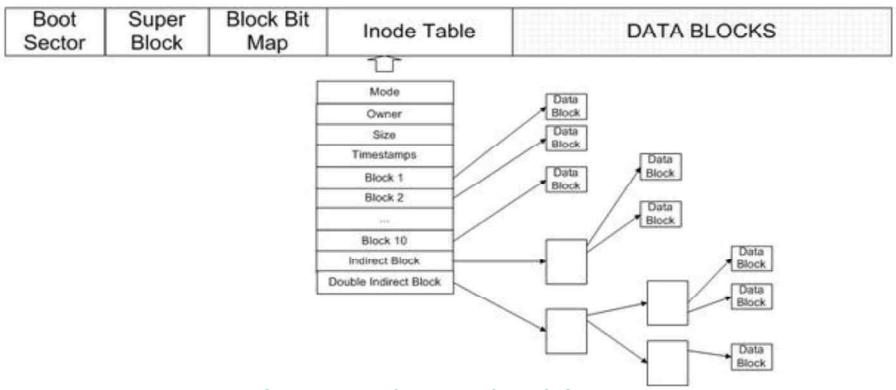
Figure 40.4: File Creation Timeline (Time Increasing Downward)

- Open: 1) create a new inode for bar and update i-bitmap, 2) insert a new entry into foo's data block (10 I/Os for just creating a file)
- Write: 5 I/Os per a write (d-bitmap read/update, inode read/update, actual user data write)

☞ Notice the importance of cache and delayed writes

UFS (Unix file system)

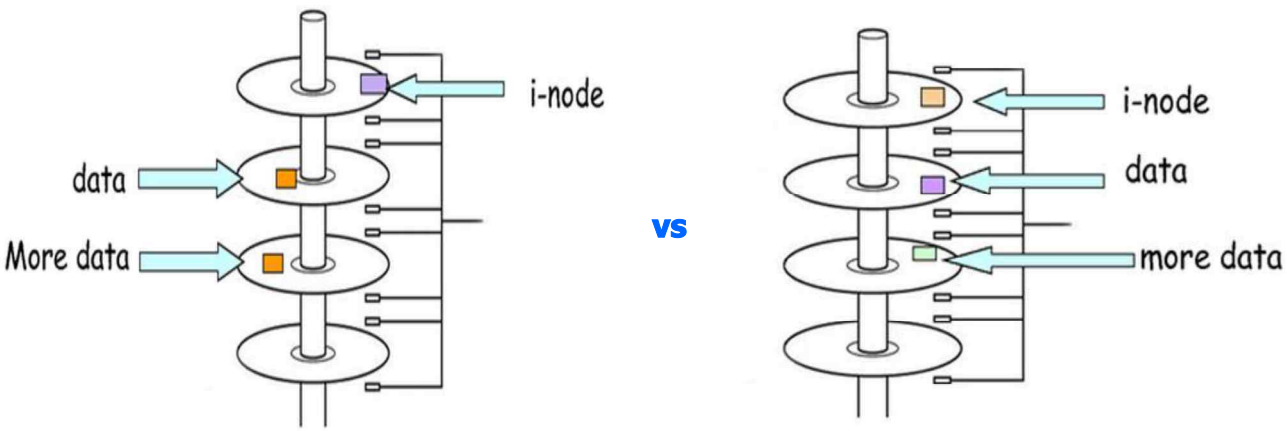
- Features
  - ✓ From: Dennis Ritche and Ken Thompson, “The Unix Time Sharing System”, CACM’74
  - ✓ Various files: ordinary file, directory, special file, pipe, …
  - ✓ Directory entry contains a file name and its i-number
  - ✓ i-number is used to find an inode which has several information for the file including data block locations
  - ✓ The first Linux file system(and Minix FS) has a similar layout



(Source: pike.cs.ucla.edu)

Performance consideration

- Issues of the UFS (and the first Linux file system)
  - ✓ Performance
    - Especially the long seek time between an inode and data blocks (common access cases➔ see page 30)
  - ✓ Scalability
    - Need to support more larger partition as storage size increases

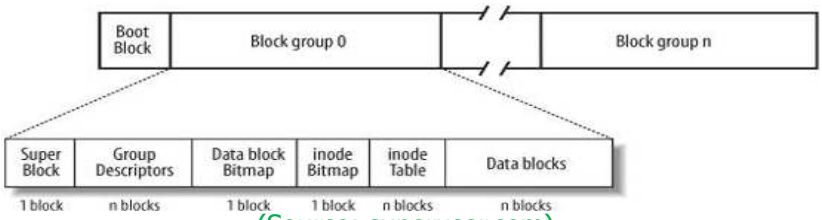


(Source: docstoc.com)

Ext2 file system

- Features
  - ✓ From: R. Card, T. Ts'o and S. Tweedie, "Design and Implementation of the second extended FS", http://e2fsprogs.sourceforge.net/ext2intro.html
  - ✓ Performance enhancement
    - Inode and associate data blocks in adjacent cylinders
    - pre-allocation: usually 8 adjacent blocks
    - Read-ahead during sequential reads
  - ✓ Scalability: increase FS size, file size, name size in directory
  - ✓ New features: fast symbolic link, mount options, various block size, hash-indexed directories, …

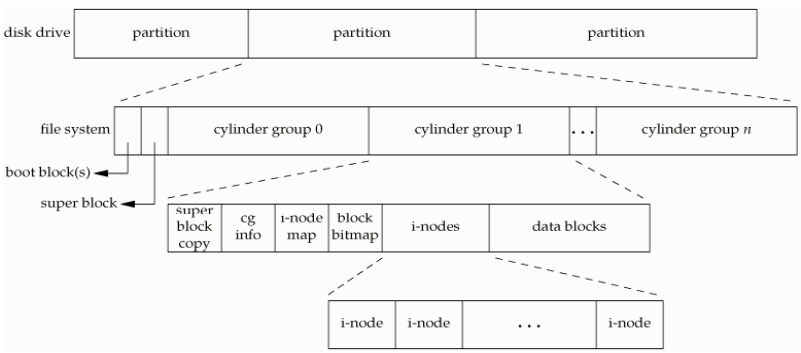
Figure 18-1. Layouts of an Ext2 partition and of an Ext2 block group



(Source: superuser.com)

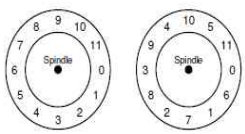
FFS (Fast File System)

- Features
  - ✓ From: M. Mckusick et al., "A Fast File system", ACM TOCS'84
    - Have been cited around 1000 times
  - ✓ Cylinder group: one or more consecutive cylinders
  - ✓ Large I/O unit (>= 4KB)
  - ✓ Fragment management for small files
  - ✓ File system parameterization (e.g. RPM)



Bits in map	XXXX	XX00	00XX	0000
Fragment numbers	0-3	4-7	8-11	12-15
Block numbers	0	1	2	3

Example layout of blocks and fragments in a 4096/1024 file system.



FFS: Standard Versus Parameterized Placement

(Source: CS 5204: Operating systems, Virginia Tech and Remzi's book)

FFS (Fast File System)

- Allocation in FFS
  - ✓ Idea: keep related stuff together
    - Data and related inode, file and its related directory, …
  - ✓ Allocation rules
    - Rule 1. Directory: place it into a cylinder group with a low number of allocated directories and a high number of free inodes
      - To balance directories across groups
      - To be able to allocate a bunch of files subsequently
    - Rule 2. File: 1) put files in the cylinder group of the directory they are in, 2) allocate data blocks of a file in the same group as its inode
    - E.g.) create four files, /a/c /a/d, /a/e, /b/f
      - Assumption: 1) Group: 10 inodes and data blocks. 2) Directory: 1 block, 3) file: 2 blocks
      - ➔ FFS place the first three files near (same group) and the fourth far away (in some other group)

```
group inodes      data
    0 /---------- /----------
    1 acde------- accddee---
    2 bf--------- bff-------
    3 ---------- ----------
    4 ---------- ----------
    5 ---------- ----------
    6 ---------- ----------
    7 ---------- ----------
    ...
```

(FFS allocation)

```
group inodes      data
    0 /---------- /----------
    1 a---------- a----------
    2 b---------- b----------
    3 c---------- cc---------
    4 d---------- dd---------
    5 e---------- ee---------
    6 f---------- ff---------
    7 ---------- ----------
    ...
```

(Even allocation)

FFS (Fast File System)

- How to handle a large file for allocation in FFS?
  - ✓ Issue: Large file ➔ fill up a cylinder group with its own data ➔ undesirable with the consideration of the namespace locality
  - ✓ Allocation rules
    - Rule 3. For a large file: Allocate some number of blocks in a group. Then, go to another group and allocate some number of blocks there. Then, move another one. …
      - Some number: Direct blocks, blocks that pointed by an index block
    - E.g.) create a file that has 30 blocks
      - Assumption: 1) Group: 10 inodes, 40 data blocks ➔ 5 blocks per a group
      - Pros) locality among files, Cons) locality in a file
      - Can amortize with large chunk size between seek (Fig 41.2, Seek=10ms, Bandwidth = 40MB/s: if (4MB per group) ➔ 100ms for transmission: 90% bandwidth, if (400K per group) ➔ 10ms for transmission: 50% bandwidth)

```
group inodes      data
    0 /a--------- /aaaaa----- ---------- ---------- ----------
    1 ---------- aaaaa----- ---------- ---------- ----------
    2 ---------- aaaaa----- ---------- ---------- ----------
    3 ---------- aaaaa----- ---------- ---------- ----------
    4 ---------- aaaaa----- ---------- ---------- ----------
    5 ---------- aaaaa----- ---------- ---------- ----------
    6 ---------- ---------- ---------- ---------- ----------
    ...
```

(FFS allocation)

```
group inodes      data
    0 /a--------- /aaaaaaaaa aaaaaaaaaa aaaaaaaaaa a---------
    1 ---------- ---------- ---------- ---------- ----------
    2 ---------- ---------- ---------- ---------- ----------
    ...
```

(Without Rule 3)

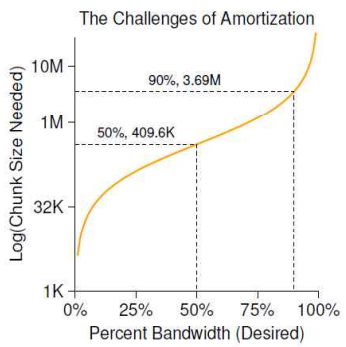
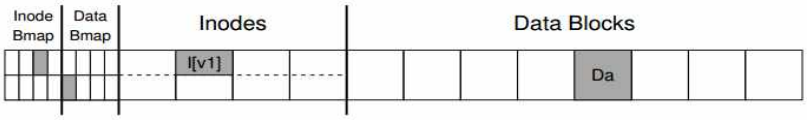


Figure 41.2: Amortization: How Big Do Chunks Have To Be?

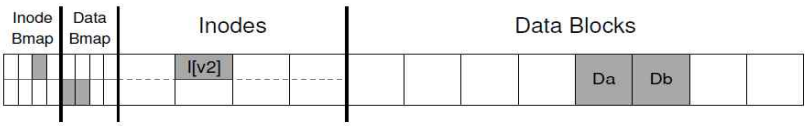
Reliability consideration

- Crash-consistency problem
  - ✓ Assume that we append data into a file in the simple file system
    - Before appending




```
owner       : remzi
permissions : read-write
size        : 1
pointer     : 4
pointer     : null
pointer     : null
pointer     : null
```

    - After appending

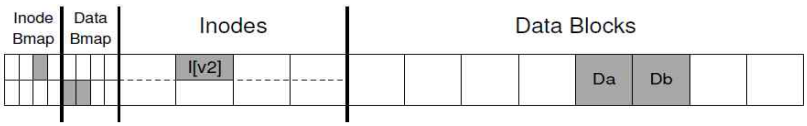



```
owner       : remzi
permissions : read-write
size        : 2
pointer     : 4
pointer     : 5
pointer     : null
pointer     : null
```

(Source: Remzi, "Operating Systems: Three Easy Pieces")

Reliability consideration

- Crash-consistency problem
  - ✓ Assume that we append data into a file in the simple file system



  - ✓ For this transition, we need three operations
    - 1) inode: I[v2], 2) Data block: Db, 3) Data bitmap: B[v]
  - ✓ Errors happen at any time
    - Some operations have done while others not (e.g. due to buffering)
      - Db written only case: no problem in the viewpoint of file system
      - B[v] written only case: space leak
      - I[v2] written only case: two problems (inconsistent state and garbage data read)
      - I[v2] and Db written case: inconsistent
      - B[v2] and Db written case: inconsistent

Reliability consideration

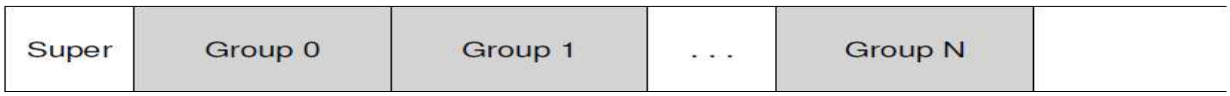
- Traditional solution: fsck (file system checker)
  - ✓ Consist of several passes
    - Superblock: usually sanity check
    - Free blocks: check all inodes and their used blocks. If there is an inconsistent case in bitmaps, correct it.
    - Inode state: validity check in each inode
    - Inode links: link counts check by scanning the entire directory tree. Move the missed file (there is an inode but no directory entry points it) into the lost+found directory
    - Duplicates: find blocks which are pointed by two or more inodes
    - Bad blocks: pointer that points outside its valid ranges
    - Directory checks: fs-specific knowledge based directory check
  - ✓ Issue: too slow
    - Remzi says that "the fsck looks like that, even though you drop the key in your bedroom, you start a search-the-entire-house-for-key algorithm, scanning from the basement, kitchen, and every room."

Reliability consideration

- Alternatives
  - ✓ fsck: A lazy approach
  - ✓ Journaling: An active approach
    - Ext3, Reiserfs, IBM's JFS, …
  - ✓ Soft update
    - Suggested by G. Ganger and Y. Patt
    - Carefully order all writes so that on-disk structures are never left in an inconsistent state (e.g. data block is always written before its inode)
    - Soft update is not easy to implement since it requires intricate knowledge about file system (On contrary, journaling can be implemented with relatively little knowledge about FS)
  - ✓ COW
    - Used in Btrfs and Sun's ZFS
    - LFS can be considered as an early example of a COW
  - ✓ Optimistic crash consistency
    - Enhance performance by issuing as many writes to disk as possible
    - Exploit checksum as well as a few other techniques

Ext3 file system

- Features
  - ✓ From: T. Ts'o and S. Tweedie, "Planned Extensions to the Linux Ext2/Ext3 Filesystem", USENIX ATC, 2002.
  - ✓ Integrate journaling into ext2 file system
    - A kind of WAL (Write-ahead logging)
    - Key idea: When updating disks, before overwriting the structure in place, first write down a little note to somewhere in a well-known location, describing what you are about to do.
    - Three types: journal (data journal), ordered (metadata journal, ordered, default), writeback (metadata journal, non-ordered)



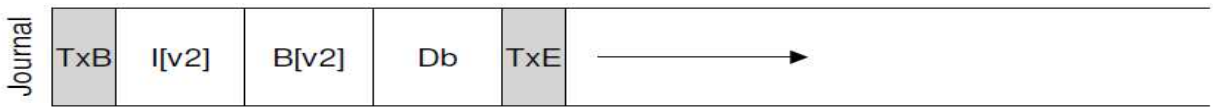
(Ext2 disk layout)

(Ext3 disk layout)

Ext3 file system

- Data Journaling
  - ✓ Assume we want to do three writes (I[v2], B[v], and Db)
  - ✓ Before writing them to their final locations, we first write them to the log (journal).

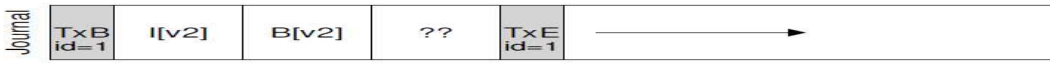
(Source: Remzi, "Operating Systems: Three Easy Pieces")



    - TxB: Transaction begin, include Tid and writes information
    - Log
      - Physical logging: same contents to the final locations
      - Logical logging: intent (save space, but more complex)
    - TxE: End with Tid
  - ✓ After making this transaction safe on disk, we are ready to update the original data ➔ checkpointing
    - Case 1: failures btw journaling and checkpointing, we can replay (redo) journal
    - Case 2: failures before finishing journaling ➔ skip transaction (undo)
  - ✓ Conclusion ➔ 2 steps: 1) Journal write, and 2) Checkpoint

Ext3 file system

- How to write journal safely?
  - ✓ For journaling, we need to write a set of blocks
    - e.g. TxB, i[v2], B[v2], Db, TxE
    - issue all writes at once ➔ Unsafe, might be loss some requests



    - Transaction looks valid (it has begin and end). Thus, replaying journal leads wrong data to be updated.
  - ✓ Approach 1: issue each request at a time, wait for each to complete, then issuing the next ➔ Too slow
  - ✓ Approach 2: issue all writes at once and apply checksum using all contents in the journal
  - ✓ Approach 3: employ commit
    - Separate TxE from all other writes
    - Recovery: 1) not committed ➔ skip it, 2) committed, but not in the original locations ➔ redo logging

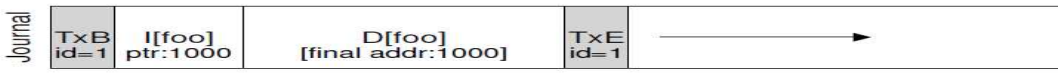
Ext3 file system

- Metadata Journaling
  - ✓ Data journaling writes data twice, which increases I/O traffic (reducing performance), especially painful for sequential writes
  - ✓ Metadata journaling
    - User data is not written to the journal (metadata only)

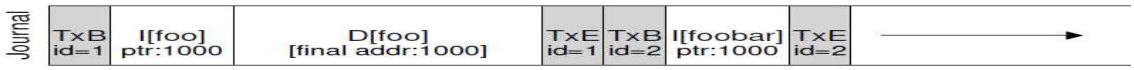
  - ✓ Question?
    - Does the writing order btw user data and journal become matter? ➔ Yes, writing journal before user data causes problems (garbage pointing)
  - ✓ Conclusion: metadata journaling
    - Data write ➔ Journal metadata write ➔ Journal commit ➔ Checkpoint ➔ Free
  - ✓ Real world
    - Ext3: support both ordered and non-ordered (writeback)
    - Windows NTFS and SGI's XFS use non-ordered metadata journaling

Ext3 file system

- Revoke record in journal: for block reuse handling
  - ✓ Scenario: 1) there is a directory called foo, 2) a user adds an entry to foo (create a file), 3) foo's contents are written to block 1000, 4) log are like the following figure (note that directory is metadata, which is also logged)



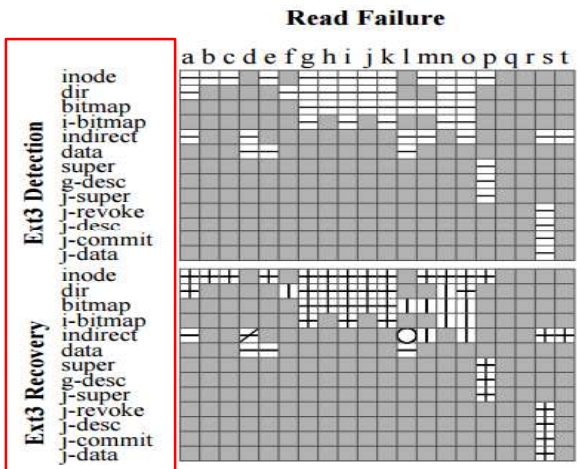
  - ✓ 5) The user deletes the foo (and its subfiles), 6) The user creates another file (say foobar), which uses the block 1000, 7) Writes for foobar are logged (note that file contents themselves are not logged)



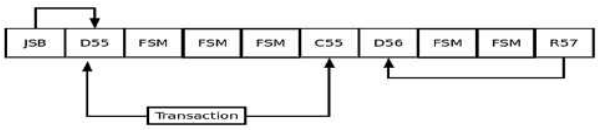
  - ✓ 8) At this point, a crash occurs. 9) recovery performs "redo" from the beginning of the log. 10) overwrites the user data of the file foobar with the old directory contents.
  - ✓ Solution
    - Ext3 adds a new type of record, a revoke record, for the deleted file or directory. When do replaying, any revoked records are not redo

Ext3 file system

- Revisit the IRON file system
  - ✓ Layout awareness: data + journal
  - ✓ Data: inode, bitmap, indirect, dir, data, g-desc
  - ✓ Journal
    - Transaction: consist of j-desc, j-data, j-revoke and j-commit
    - j-super: use to designate the first descriptor block in the journal



(Source: IRON FS, SOSP'05)



Key

Abbreviation	Meaning
JSB	Journal Super Block
D[X]	Descriptor Block X
FSM	File System Metadata
C[X]	Commit Block X
R[X]	Revoke Block X

(Source: K. Fairbanks, "An Analysis of Ext4 for Digital forensics", Digital Investigation, 2012)

Ext4 file system

- Features
  - ✓ From: A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, L. Vivier, "The new ext4 FS: current status and future plans", www3.cs.stonybrook.edu/~prade/Teaching/Fall14/prez/L22/ext4.pdf
  - ✓ Larger file system capacity with 64-bit
    - Supports huge file size (from 16GB to 16TB) and file system (from 2^32 to 2^64 blocks)
    - Directory can contain up to 64,000 subdirs (32,000 in ext3)
  - ✓ Extent based mapping
    - Extent: a range of contiguous physical blocks (like segment in VM)
    - Details: https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout

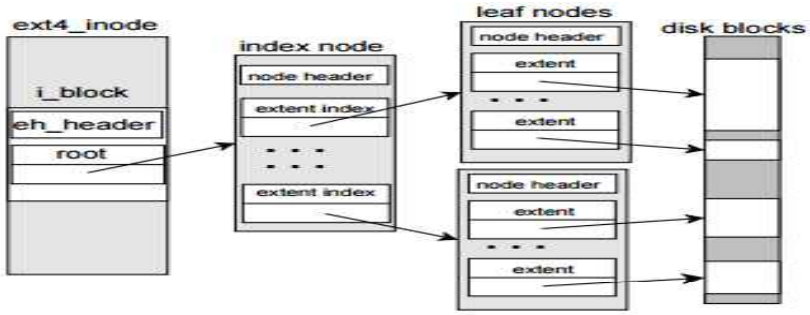


Figure 2: Ext4 extent tree layout

Ext4 file system

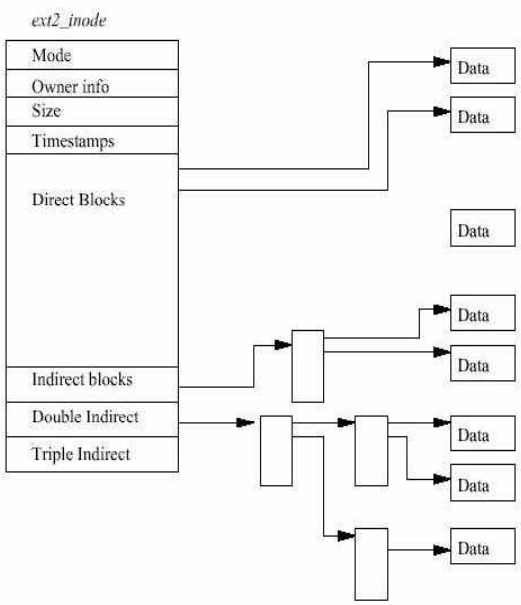
- Features (cont')
  - ✓ Persistent pre-allocation
    - Preallocate blocks for making I/Os sequential
    - Preallocation is persistent across reboots, unlike block reservations
  - ✓ Delayed and multiple blocks allocation
    - Allocations are delayed to the page flush time, instead of write operation ➔ give an opportunity to make a single request
    - Multiple block allocation for buffered I/O ➔ eliminating multiple calls to ext4_get_blocks
  - ✓ Using JBD2, a fork of JBD (Journaling block device) for 64-bit support
    - Journal checksum (Approach 3 for reducing journaling overhead)
  - ✓ Directory indexing
    - Use tree (Htree) instead of double linked list (for large # of files)
  - ✓ Fast fsck
    - Exploit hints about unused inode, skip uninitialized block group,
  - ✓ Nanosecond information for create/access/modify times

inode consideration

- inode structure

Size	Name	What is this inode field for?
2	mode	can this file be read/written/executed?
2	uid	who owns this file?
4	size	how many bytes are in this file?
4	time	what time was this file last accessed?
4	ctime	what time was this file created?
4	mtime	what time was this file last modified?
4	dtime	what time was this inode deleted?
2	gid	which group does this file belong to?
2	links_count	how many hard links are there to this file?
4	blocks	how many blocks have been allocated to this file?
4	flags	how should ext2 use this inode?
4	osd1	an OS-dependent field
60	block	a set of disk pointers (15 total)
4	generation	file version (used by NFS)
4	file_acl	a new permissions model beyond mode bits
4	dir_acl	called access control lists
4	faddr	an unsupported field
12	i_osd2	another OS-dependent field

Figure 40.1: **The Ext2 Inode**



☞ Why do we use both direct and indirect blocks (imbalanced tree)?

inode consideration

- Observation

Most files are small	Roughly 2K is the most common size
Average file size is growing	Almost 200K is the average
Most bytes are stored in large files	A few big files use most of the space
File systems contains lots of files	Almost 100K on average
File systems are roughly half full	Even as disks grow, file systems remain ~50% full
Directories are typically small	Many have few entries; most have 20 or fewer

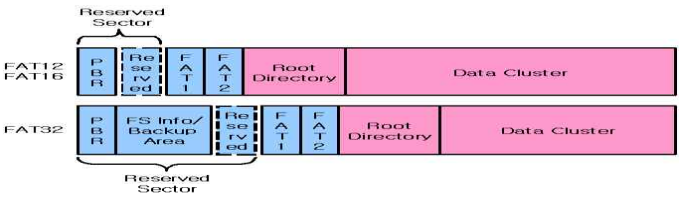
Figure 40.2: **File System Measurement Summary**

(Source: Remzi, "Operating Systems: Three Easy Pieces")

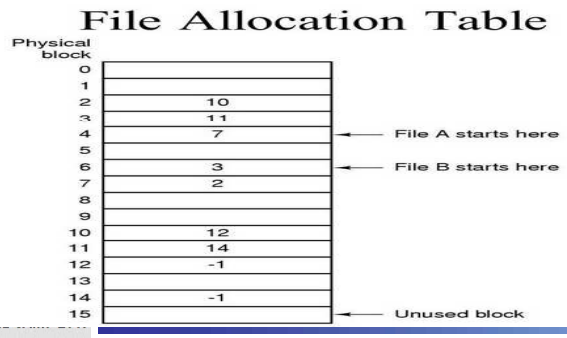
- Several ways to realize inode
  - ✓ Approach1: Multi-level index: direct block pointer, indirect block pointers (single/double/triple) ➔ hybrid for both small and large file
  - ✓ Approach 2: Extents instead of block pointers
    - Like segment in the virtual memory, less flexible but more compact
  - ✓ Approach 3: linked-based approach
    - FAT file system
  - ✓ Approach 4: reposition at each update (even inode too)
    - LFS, NILFS, yaffs, …

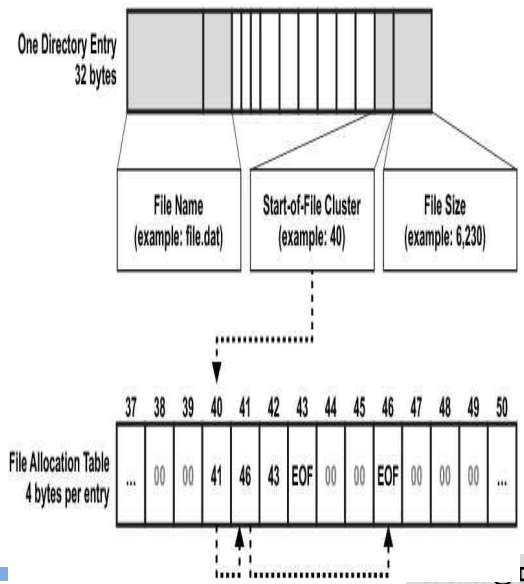
FAT file system

- Features
  - ✓ Simple and lightweight implementation
    - Designed by Marc McDonald, based on a series of discussion with Bill Gates (from wiki)
  - ✓ Today, commonly found in flash-based mobile storage



(Source: http://torystory.tistory.com/28)

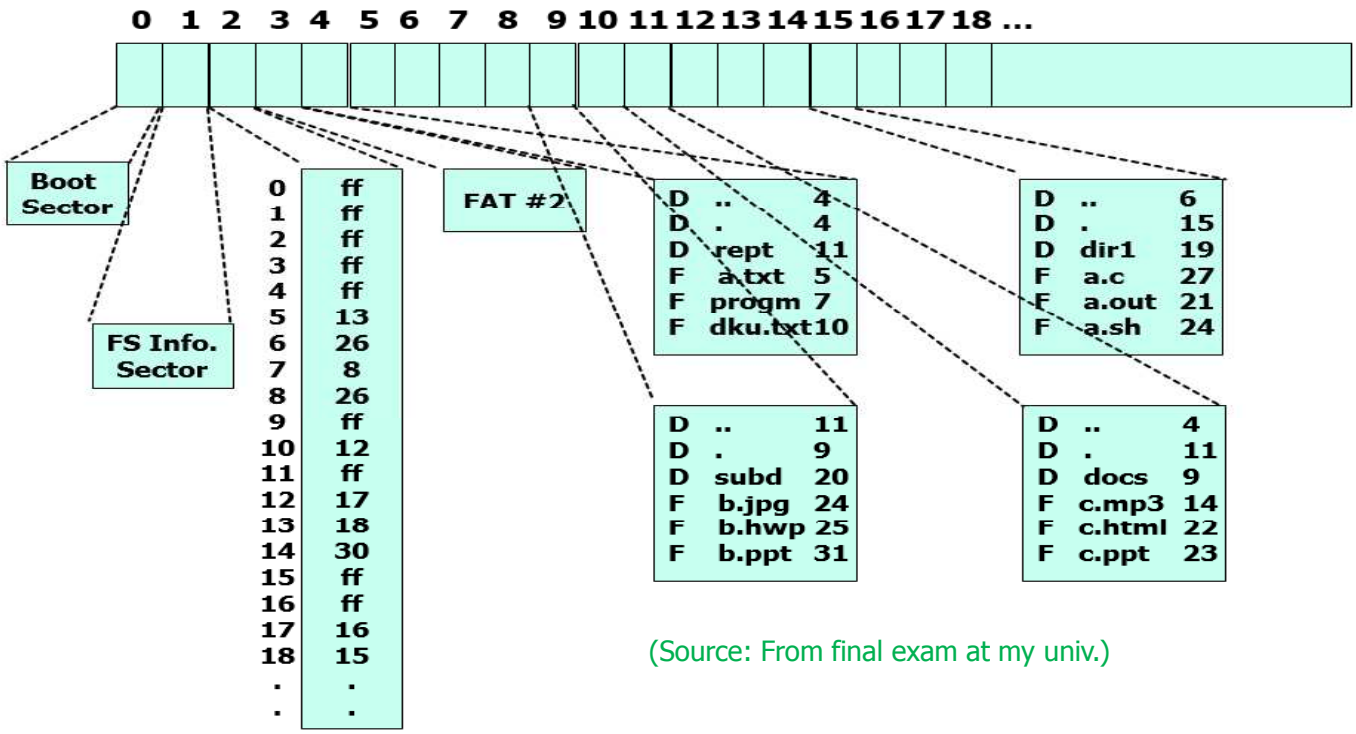




(Source: A. Silberschatz, "Operating System Concepts")

FAT file system

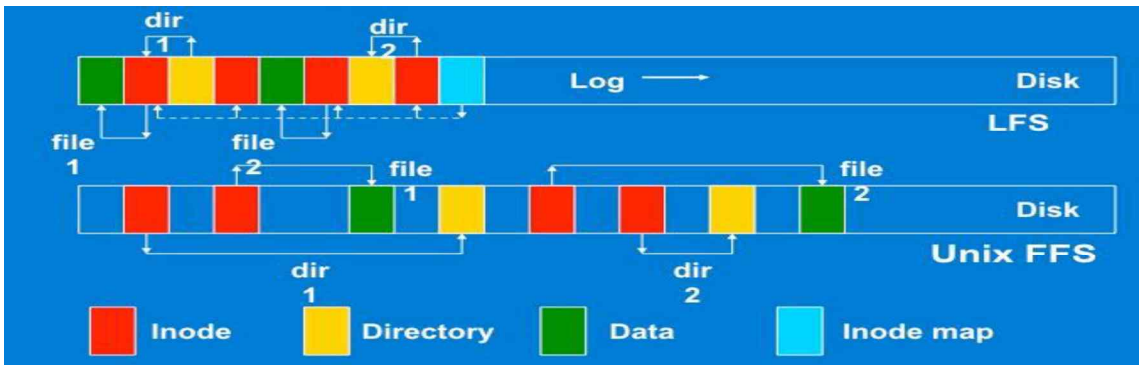
- Overall structure
  - ✓ Assume that the root is in block 4 (FS info.)



(Source: From final exam at my univ.)

LFS (Log-structured file system)

- Features
  - ✓ From M. Rosenblum and J. Ousterhout, “The Design and Implementation of a LFS”, ACM ToCS, 1992
  - ✓ Log-structure: write all modifications in a log-like structure
    - Assume two files are created, named dir1/file1 and dir2/file2
      - FFS: need 8 non-sequential writes (note 1) inode first, 2) cylinder group)
      - LFS: need a single sequential write
    - inode is also relocated

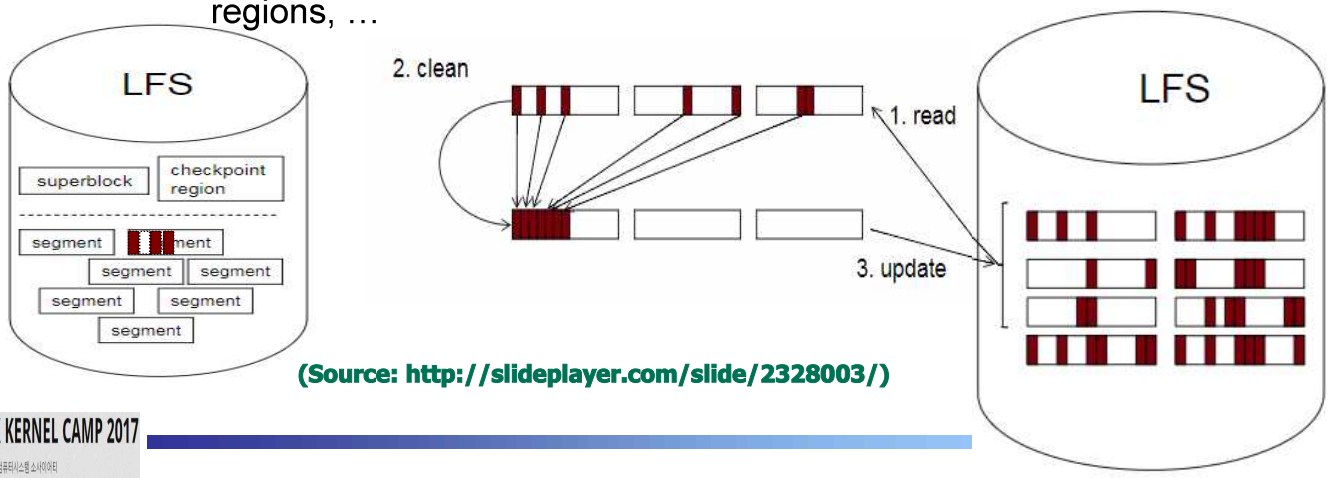


(Source: M. Rosenblum’s ToCS paper)

LFS (Log-structured file system)

- How to manage log?
  - ✓ Introduce segment concept
    - Write sequentially in a segment
  - ✓ Segment cleaning
    - Generate free segment
    - A.k.a garbage collection
  - ✓ New data structure
    - Inode map, segment summary, segment usage table, checkpointing regions, …

Data structure	Purpose	Location	Section
Inode	Locates blocks of file, holds protection bits, modify time, etc.	Log	3.1
Inode map	Locates position of inode in log, holds time of last access plus version number.	Log	3.1
Indirect block	Locates blocks of large files.	Log	3.1
Segment summary	Identifies contents of segment (file number and offset for each block).	Log	3.2
Segment usage table	Counts live bytes still left in segments, stores last write time for data in segments.	Log	3.6
Superblock	Holds static configuration information such as number of segments and segment size.	Fixed	None
Checkpoint region	Locates blocks of inode map and segment usage table, identifies last checkpoint in log.	Fixed	4.1
Directory change log	Records directory operations to maintain consistency of reference counts in inodes.	Log	4.2



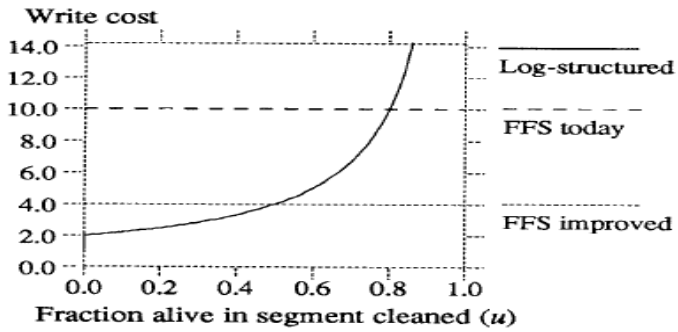
(Source: http://slideplayer.com/slide/2328003/)

LFS (Log-structured file system)

- Issues
  - ✓ How to model the write cost?
    - Write cost

$$\text{write cost} = \frac{\text{total bytes read and written}}{\text{new data written}} = \frac{\text{read segs} + \text{write live} + \text{write new}}{\text{new data written}} = \frac{N + N*u + N*(1-u)}{N*(1-u)} = \frac{2}{1-u}$$

    - Implication
      - Performance of LFS depends on the utilization (less than 0.5 for being compatible with the improved FFS)
      - Segment selection policy is important while cleaning

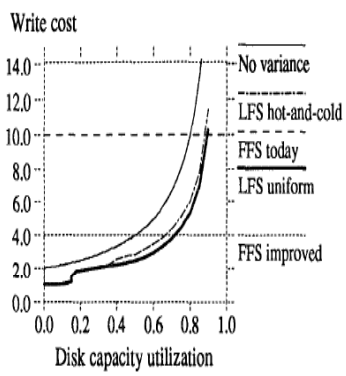


LFS (Log-structured file system)

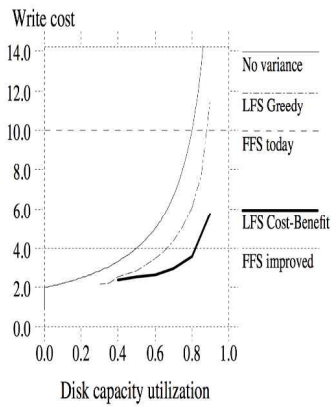
- Issues
  - ✓ How to choose a victim segment?
    - Greedy: choose a segment which has the least utilization
      - uniform access pattern vs. hot-cold access pattern ➔ little difference
    - Cost-benefit: choose the most beneficial one

$$\frac{\text{benefit}}{\text{cost}} = \frac{\text{free space generated} * \text{age of data}}{\text{cost}} = \frac{(1-u)*\text{age}}{1+u}$$

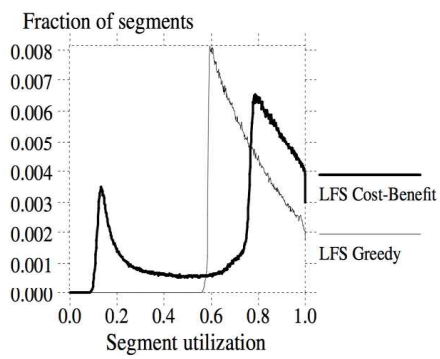
      - Make a bimodal segment distribution (mostly full or nearly empty)



Greedy results



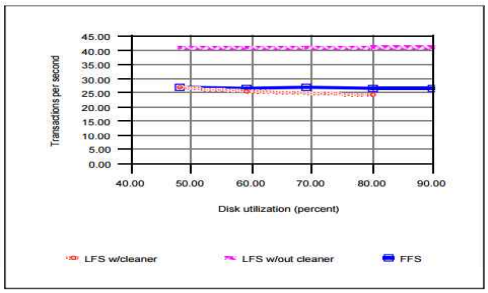
Cost-benefit results

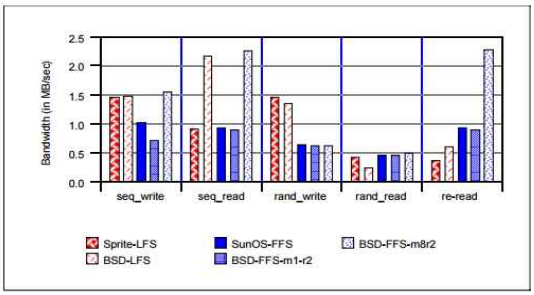


Segment utilization distribution

LFS (Log-structured file system)

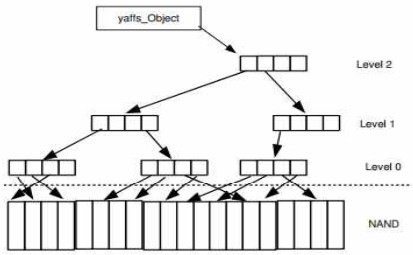
- Discussion
  - ✓ Some concerns: cleaning overhead, sequential re-read

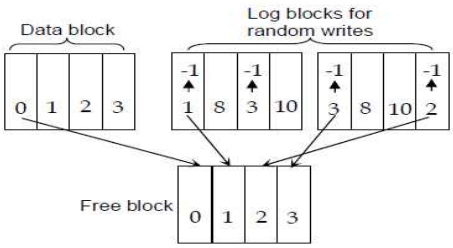




(Source: M. Seltzer et al., "FS logging vs Clustering: A performance comparison", USENIX'95)

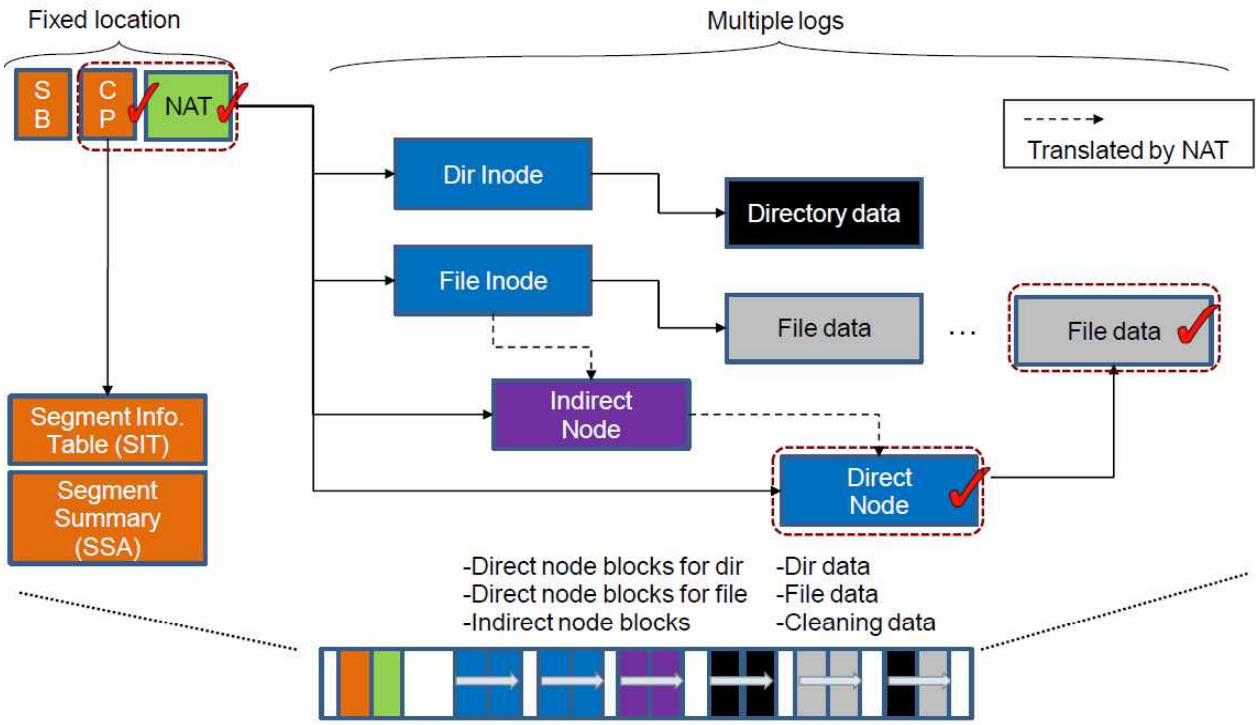
  - ✓ Good property
    - Log-structured ➔ no overwrite ➔ match well with Flash
    - Many Flash FS (e.g. YAFFS2, ELF) and FTL (e.g. BAST, FAST) are based on the Concept of LFS





F2FS

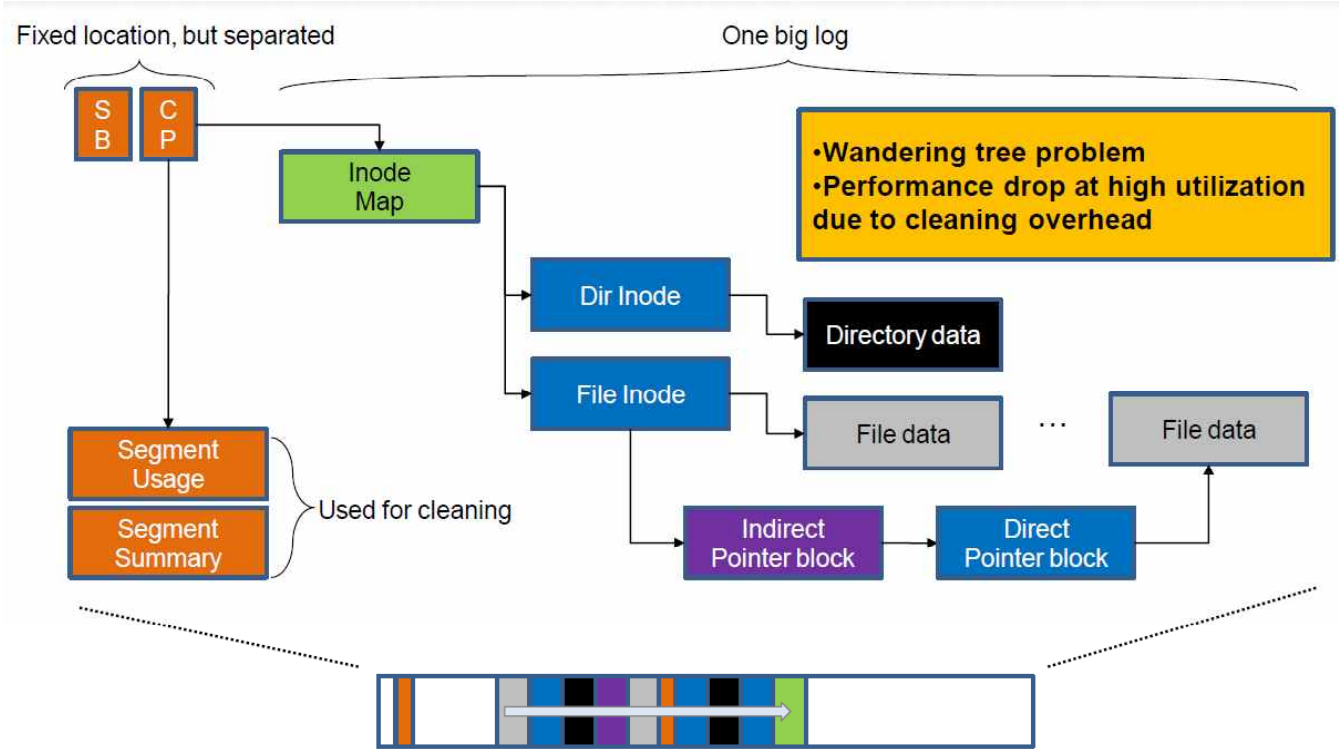
- Structure



(Source: J. Hwang, "Flash Friendly File System", 2013)

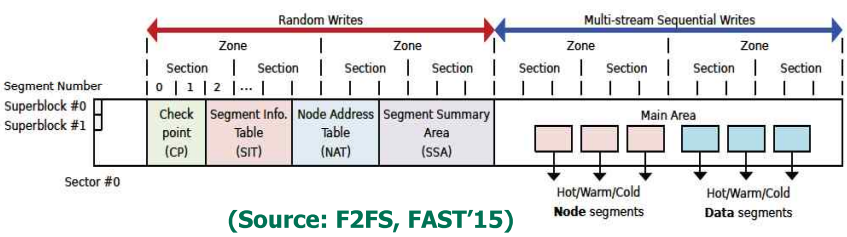
F2FS

- Comparison with the traditional LFS



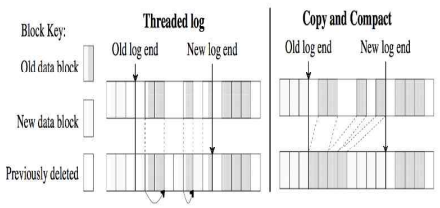
F2FS

- Proposal (revisit)
  - ✓ Layout
    - Three configurable units: zone, section, segment ➔ aware of the erase unit of Flash memory (alignment)
    - Six logical areas: SB (FS info), CP(consistent status), SIT(valid blocks), NAT (node blocks location), SSA (pointers for node/data used for cleaning), Main area (node and data)
  - ✓ Node block: inode, direct, indirect node
    - Wandering tree problem ➔ direct node and NAT update only
  - ✓ Multi-head logging
    - Apply logging so that FTL do switch merge in most case (like LFS)
    - Six log areas: hot/cold separation to reduce cleaning overhead
  - ✓ Cleaning (Garbage collection)
    - Foreground vs background, greedy vs cost-benefit
  - ✓ Adaptive logging (high utilization)
    - Normal logging vs. threaded logging



(Source: F2FS, FAST'15)

Figure 1: On-disk layout of F2FS.



(Source: M. Rosenblum's ToCS paper)

Take out lessons

- Issues for file system evolution
  - ✓ Basic requirement (layout)
  - ✓ Performance consideration
  - ✓ Reliability consideration
  - ✓ Storage awareness
  - ✓ Bigdata consideration
  - ✓ Workload consideration
  - ✓ …

Patch Type	Typical Cases	XFS	Ext4	Btrfs	Ext3	Reiser	JFS
Semantic	forget sync	17	11	6	11	5	1
	forget config	43	43	23	16	8	1
	early enospc	5	9	14	7		
	wrong log credit	6	8	1	1	1	
Concurrency	lock inode update	6	5	2	4	4	2
	lock sleep	8	8	1	1	8	
	wrong kmalloc flag	20	3	3	2		1
	miss unlock	10	7	4	2	2	4
Memory	leak on failure	14	21	16	11	1	3
	leak on exit	1		1	4		1
Error Code	miss I/O error	10	11	8	15	4	1
	miss mem error	4	2	13	1	1	
	bad error access		3	8			2
Performance	remove lock	17	14	14	8	5	1
	avoid redun write	6	4	5	4		2
	check before work	8	5	15	2	1	
	save struct mem	3	9		1	3	
Reliability	metadata validation	12	9	1	7	2	1
	graceful handle	8	6	5	5	1	4

Table 9: **Common File System Patches.** *This table shows the classification and count of common patches across all file systems.*

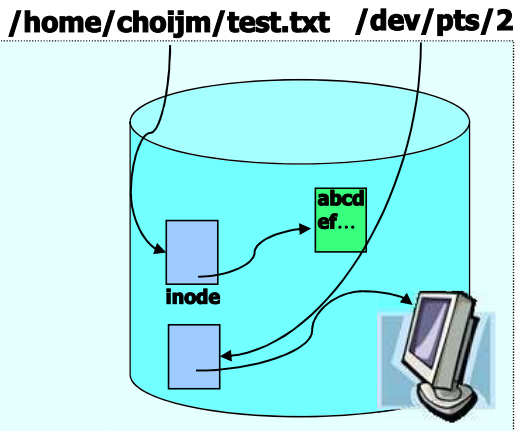
(Source: L. Lu et al., "A Study of Linux FS evolution", FAST'13)

Contents

- Introduction
  - ✓ Linux Kernel Overview
  - ✓ Why do we learn file system at the source code level?
- Logical structure of file system
  - ✓ Layout
  - ✓ Case: UFS, Ext2 (FFS), Journaling, Ext4, FAT, LFS, F2FS, …
- File system abstraction
  - ✓ Interface
  - ✓ Case: System call, VFS, Specific FS, Page Cache, Block layer, …
- Discussion
  - ✓ How to make a new file system?
- File system Practice
  - ✓ 1) FAT file system (with recovery), 2) F2FS (with MTD), 3) Performance evaluation

File system Interface

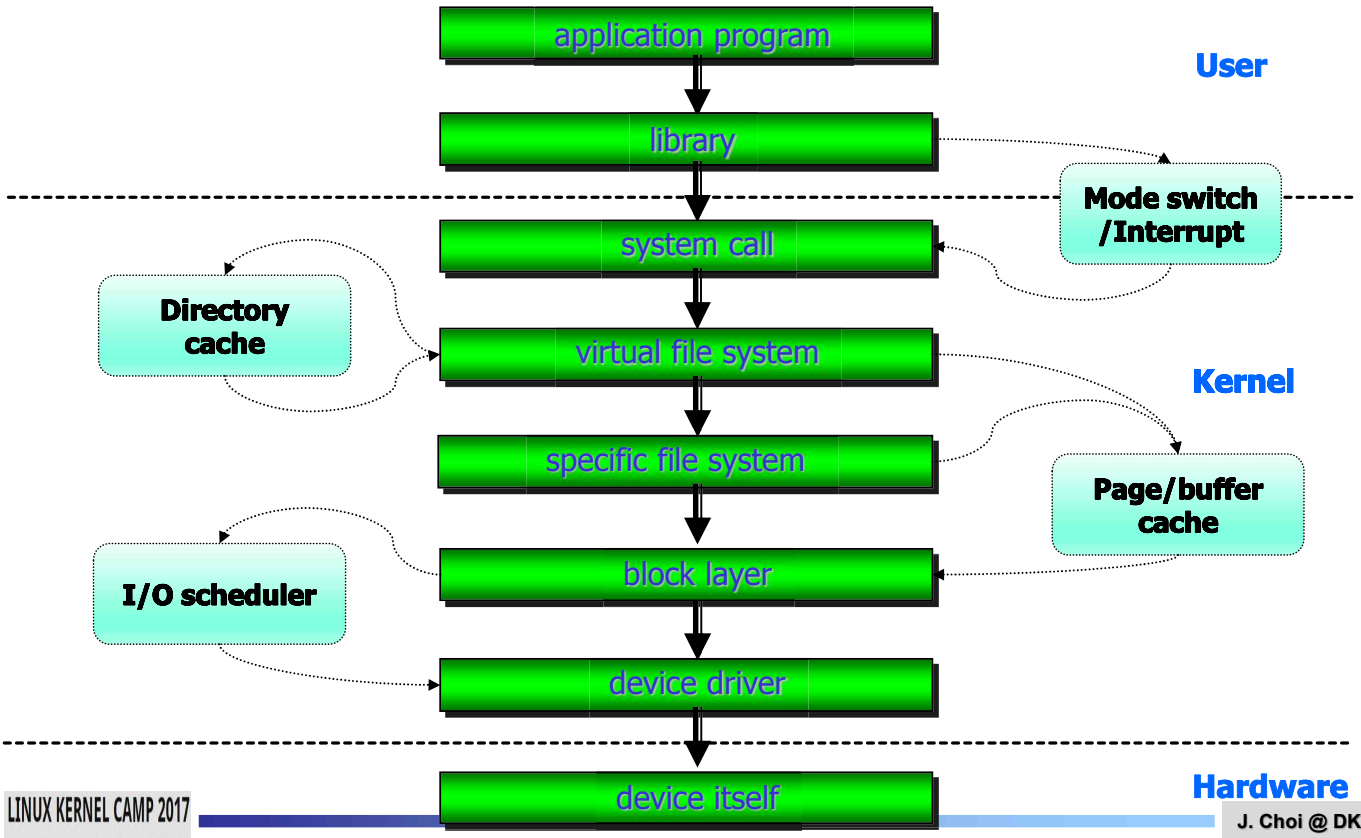
- Interface example




```
/* file_device.c, by choijm. choijm@dku.edu */
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define MAX_BUF 4
char fname[] = "/home/choijm/test.txt";
char tmp_data[] = "abcdefghijklmn";

int main()
{
        int fd, size;
        char buf[MAX_BUF];

        fd = open(fname, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
        write(fd, tmp_data, sizeof(tmp_data));
        close(fd);

        fd = open(fname, O_RDONLY);
        lseek(fd, 5, SEEK_SET);
        size = read(fd, buf, MAX_BUF);
        close(fd);

        fd=open("/dev/pts/2", O_WRONLY);
        write(fd, buf, MAX_BUF);
        close(fd);
}
```

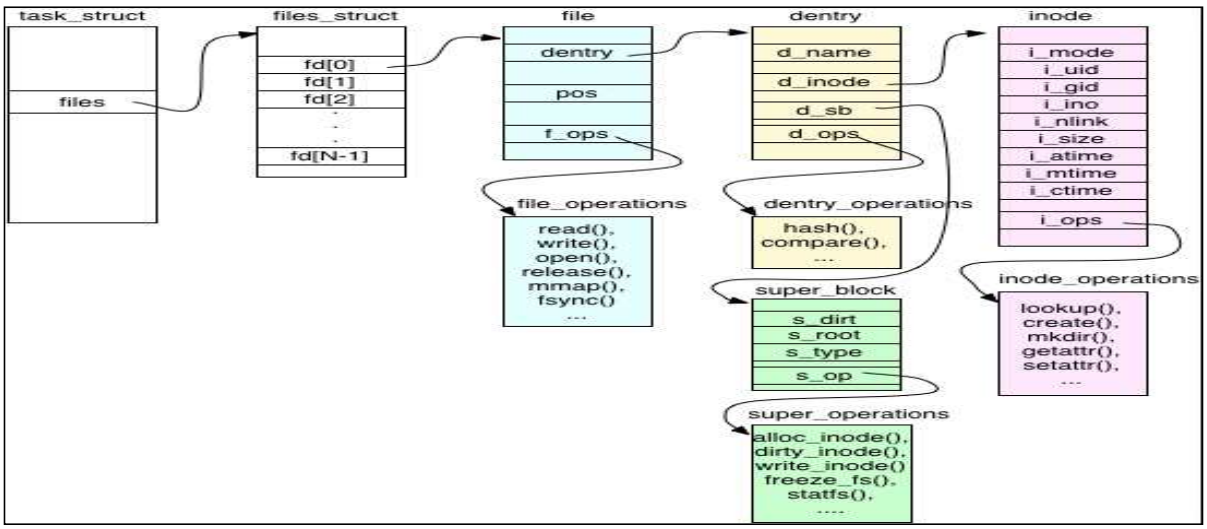

File system Interface

- Layered architecture for abstraction



File system Interface

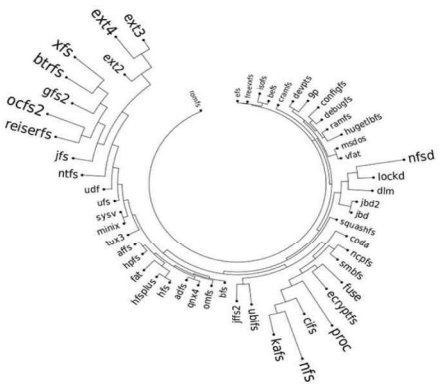
- Before we dive into
  - ✓ the control flow of open, read, write, mount, etc
- Prerequisite 1
  - ✓ What we expect after finishing open
  - ✓ 1) task, 2) files_struct (file descriptors), 3) file, 4) dentry, 5) inode



(Source: http://www.coins.tsukuba.ac.jp/~yas/classes/os2-2012/2013-02-19/index.html)

File system Interface

- Prerequisite 2
  - ✓ What is the VFS?
    - A layer between user processes (or the standard library) and the real filesystem implementation
    - Support various native filesystems
      - provide uniform ways of manipulating files, directories, and other objects
      - Use function pointers instead of fixed functions



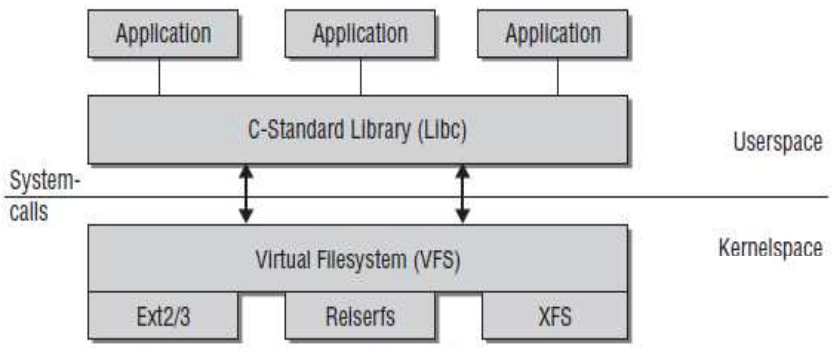


Figure 8-1: VFS layer for filesystem abstraction.

(Source: http://infosthetics.com/archives/2009/06/a_visual_expedition_inside_the_linux_file_systems.html)

(Source: Professional Linux Kernel Architecture, Wrox)

File system Interface

- Prerequisite 2
  - ✓ Five key data structures in VFS

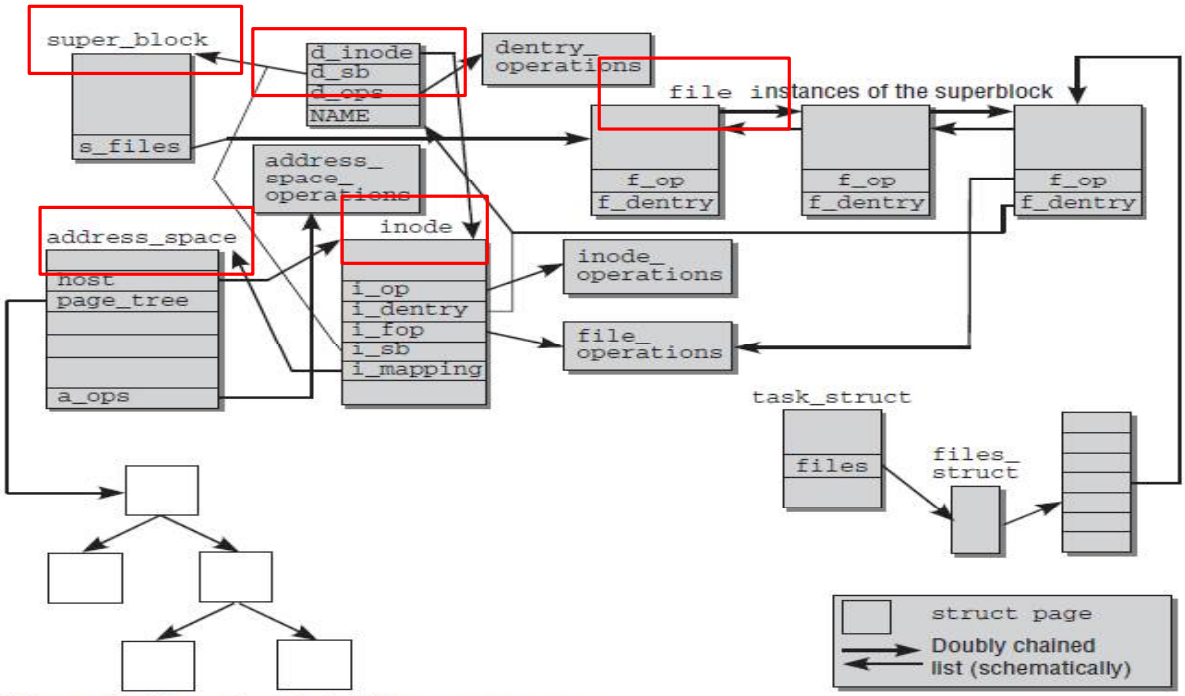


Figure 8-3: Interplay of the VFS components.

(Source: Professional Linux Kernel Architecture, Wrox)

File system Interface

- Prerequisite 2
  - ✓ Five key operations
  - ✓ file operations: read, write, lseek, open, …
    - Managed by the file structure
  - ✓ inode operations: create, delete, rename, lookup, …
    - Managed by the inode structure
  - ✓ dentry operations: d_hash, d_compare
    - Managed by the dentry structure
    - Link btw a file name and a inode. used for fast lookup operations
  - ✓ address_space operations: readpage, writepage, …
    - Managed by the address_space structure
    - Connect file system and page cache for I/Os
  - ✓ superblock operations: fill_superblock, inode manipulation, …
    - Managed by the super block
    - Related to file system

File system Interface

- Prerequisite 3
  - ✓ Two caches
    - Directory cache (dcache)
      - maintain a partial in-memory image of the FS hierarchy
    - Page cache
      - responsible for most caching work for block devices

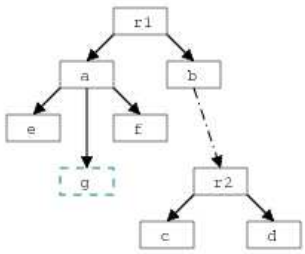


Figure 2. Example Filesystem Tree

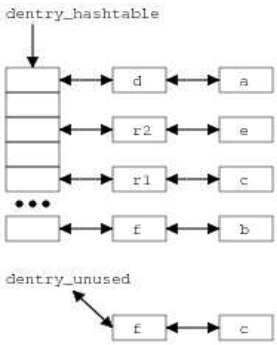


Figure 4. dentry Hash Table

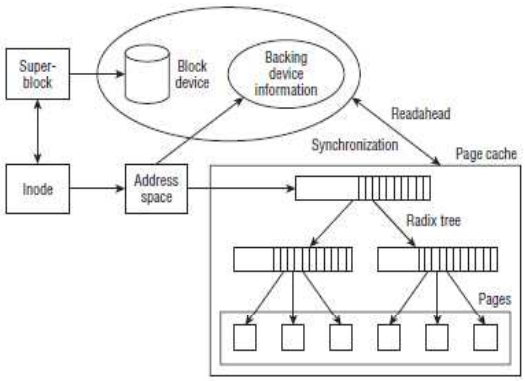


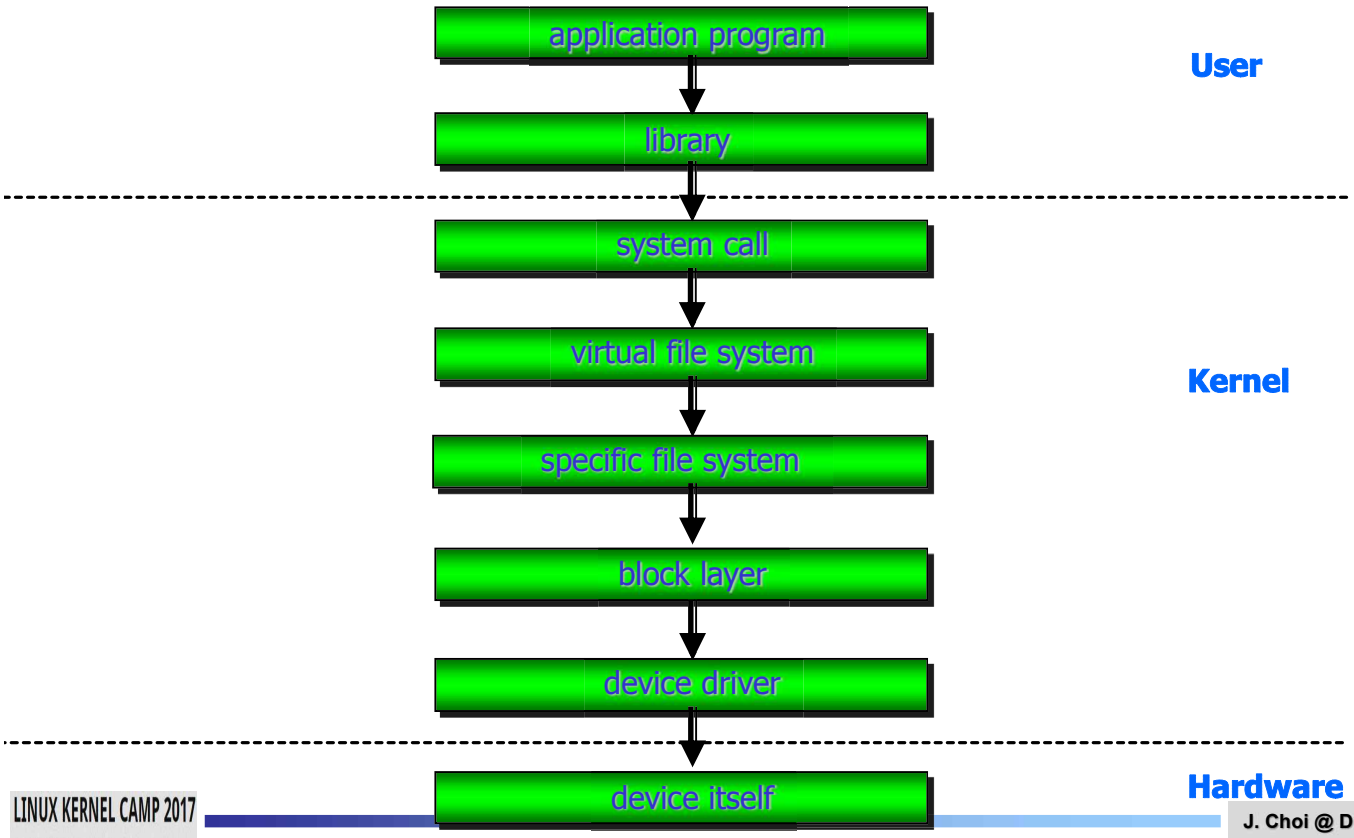
Figure 16-3: Address spaces and their connection with central kernel data structures and subsystems.

(Source: http://www.linuxjournal.com/article/7124)

(Source: Professional Linux Kernel Architecture, Wrox)

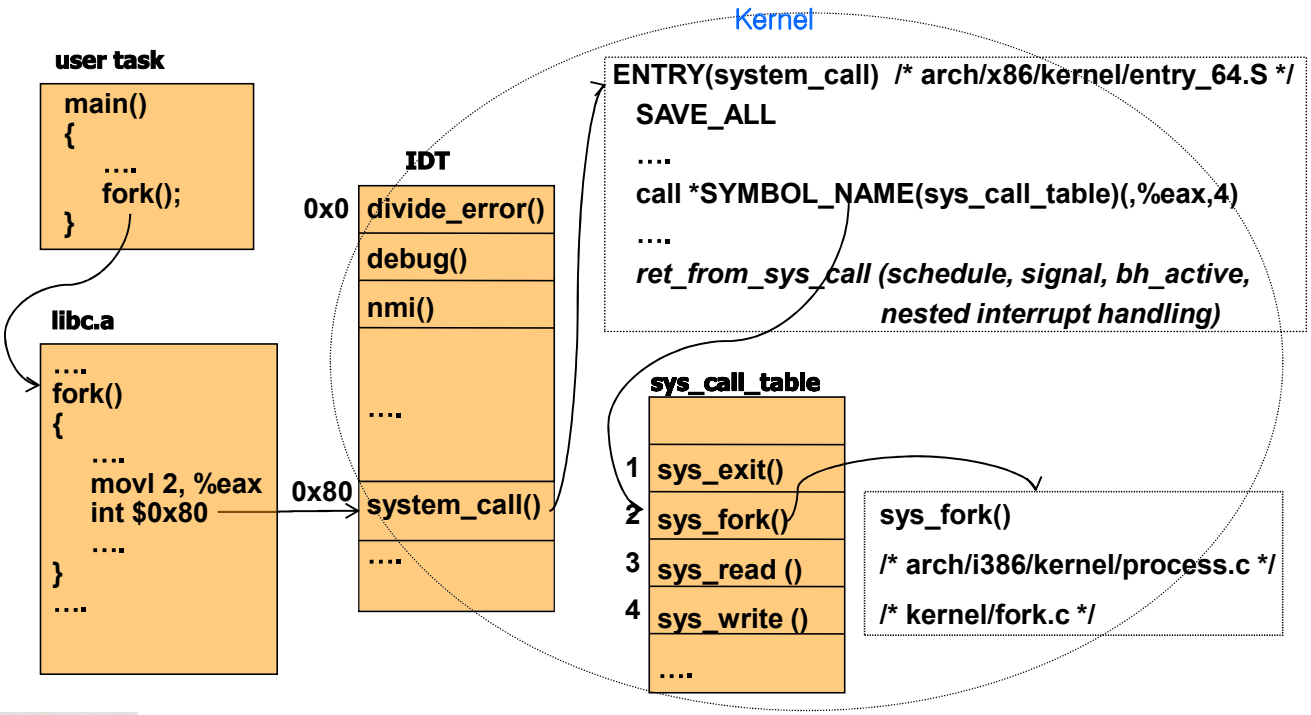
System call: Introduction

- First step for going into the Linux kernel code



System call: x86_32

- Overall: IDT and sys_call_table
  - ✓ Syscall need a mode switch (into ring 0)



System call: x86_32

- Code examples: system call registration and invocation

```
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/arch/x86/include/asm
 */
#define IRQ_MOVE_CLEANUP_VECTOR         FIRST_EXTERNAL_VECTOR

#define IA32_SYSCALL_VECTOR             0x80
#ifdef CONFIG_X86_32
# define SYSCALL_VECTOR                 0x80
#endif

/*
 * Vectors 0x30-0x3f are used for ISA interrupts.
 *   round up to the next 16-vector boundary
 */
#define IRQ0_VECTOR                     ((FIRST_EXTERNAL_VECTOR + 16) & ~15)

#define IRQ1_VECTOR                     (IRQ0_VECTOR +  1)
#define IRQ2_VECTOR                     (IRQ0_VECTOR +  2)
#define IRQ3_VECTOR                     (IRQ0_VECTOR +  3)
#define IRQ4_VECTOR                     (IRQ0_VECTOR +  4)
#define IRQ5_VECTOR                     (IRQ0_VECTOR +  5)
#define IRQ6_VECTOR                     (IRQ0_VECTOR +  6)
#define IRQ7_VECTOR                     (IRQ0_VECTOR +  7)
#define IRQ8_VECTOR                     (IRQ0_VECTOR +  8)
#define IRQ9_VECTOR                     (IRQ0_VECTOR +  9)
"irq_vectors.h" 172 [] --39%--                          68,10         30%
```

<arch/x86/include/asm/irq_vectors.h>

```
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/arch/x86/syscalls
#
# 32-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point> <compat entry point>
#
# The abi is always "i386" for this file.
#
0       i386    restart_syscall         sys_restart_syscall
1       i386    exit                    sys_exit
2       i386    fork                    sys_fork                        stub32_f
ork
3       i386    read                    sys_read
4       i386    write                   sys_write
5       i386    open                    sys_open                        compat_s
ys_open
6       i386    close                   sys_close
7       i386    waitpid                 sys_waitpid                     sys32_wa
itpid
8       i386    creat                   sys_creat
9       i386    link                    sys_link
10      i386    unlink                  sys_unlink
11      i386    execve                  sys_execve                      stub32_e
xecve
12      i386    chdir                   sys_chdir
```

<arch/x86/syscalls/syscall_32.tbl &
arch/x86/kernel/syscall_32.c>

```
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/arch/x86/kernel
#endif
}

void __init trap_init(void)
{
        int i;

        ...
        set_intr_gate(X86_TRAP_DE, divide_error);
        set_intr_gate_ist(X86_TRAP_NMI, &nmi, NMI_STACK);
        /* int4 can be called from all */
        set_system_intr_gate(X86_TRAP_OF, &overflow);
        set_intr_gate(X86_TRAP_BR, bounds);
        set_intr_gate(X86_TRAP_UD, invalid_op);
        set_intr_gate(X86_TRAP_NM, device_not_available);
        ...

        /* Reserve all the builtin and the syscall vector: */
        for (i = 0; i < FIRST_EXTERNAL_VECTOR; i++)
                set_bit(i, used_vectors);

#ifdef CONFIG_X86_32
        set_system_trap_gate(SYSCALL_VECTOR, &system_call);
        set_bit(SYSCALL_VECTOR, used_vectors);
#endif
```

<arch/x86/kernel/traps.c>

```
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/arch/x86/kernel
ENDPROC(ia32_sysenter_target)

        # system call handler stub
ENTRY(system_call)
        RINGO_INT_FRAME                 # can't unwind into user space anyway
        ASM_CLAC
        pushl_cfi %eax                  # save orig_eax
        SAVE_ALL
        GET_THREAD_INFO(%ebp)
                                        # system call tracing in operation / emu
lation
        testl $_TIF_WORK_SYSCALL_ENTRY,TI_flags(%ebp)
        jnz syscall_trace_entry
        cmpl $(NR_syscalls), %eax
        jae syscall_badsys
syscall_call:
        call *sys_call_table(,%eax,4)
syscall_after_call:
        movl %eax,PT_EAX(%esp)          # store the return value
syscall_exit:
        LOCKDEP_SYS_EXIT
        DISABLE_INTERRUPTS(CLBR_ANY)    # make sure we don't miss an interrupt
                                                        507,8-15      34%
```

<arch/x86/kernel/entry_32.S>

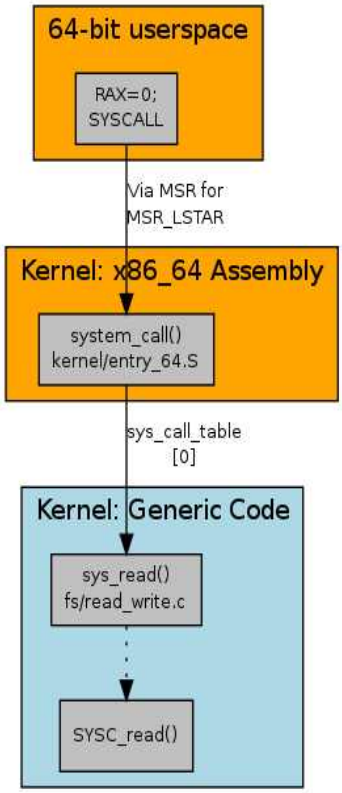
System call: x86_64

- Overall: syscall instruction
  - ✓ User program
    - Using special instruction instead of int 0x80
      - syscall/sysret, sysenter/sysexit
      - rax for syscall #, others for parameters
  - ✓ system_call entry point
    - Using Model-specific register

SYSCALL and SYSRET			
name	63 … 48	47 … 32	31 … 0
STAR C000_0081h	base selector for SYSRET CS/SS	base selector for SYSCALL CS/SS	target EIP
LSTAR C000_0082h	target RIP for PM64 callers		
CSTAR C000_0083h	target RIP for CM callers		
FMASK C000_0084h	reserved		RFLAGS mask for SYSCALL

  - ✓ sys_call_table

```
asmlinkage const sys_call_ptr_t sys_call_table[__NR_syscall_max+1] = {
    [0 ... __NR_syscall_max] = &sys_ni_syscall,
    [0] = sys_read,
    [1] = sys_write,
    /*... */
};
```



(Source: https://lwn.net/Articles/604287/)

System call: x86_64

- Code examples system call registration and invocation

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/arch/x86

```
#ifndef _ASM_X86_MSR_INDEX_H
#define _ASM_X86_MSR_INDEX_H

/* CPU model specific register (MSR) numbers */

/* x86-64 specific MSRs */
#define MSR_EFER                0xc0000080 /* extended feature register */
#define MSR_STAR                0xc0000081 /* legacy mode SYSCALL target */
#define MSR_LSTAR               0xc0000082 /* long mode SYSCALL target */
#define MSR_CSTAR               0xc0000083 /* compat mode SYSCALL target */
#define MSR_SYSCALL_MASK        0xc0000084 /* EFLAGS mask for syscall */
#define MSR_FS_BASE             0xc0000100 /* 64bit FS base */
#define MSR_GS_BASE             0xc0000101 /* 64bit GS base */
#define MSR_KERNEL_GS_BASE      0xc0000102 /* SwapGS GS shadow */
#define MSR_TSC_AUX             0xc0000103 /* Auxiliary TSC */

/* EFER bits: */
#define _EFER_SCE               0  /* SYSCALL/SYSRET */
#define _EFER_LME               8  /* Long mode enable */
#define _EFER_LMA               10 /* Long mode active (read-only) */
#define _EFER_NX                11 /* No execute enable */
#define _EFER_SVME              12 /* Enable virtualization */
"include/uapi/asm/msr-index.h" 624L, 23647C                    10,7
```

<arch/x86/include/uapi/asm/msr-index.h>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/arch/x86/syscalls

```
# <number> <abi> <name> <entry point>
#
# The abi is "common", "64" or "x32" for this file.
#
0       common  read                    sys_read
1       common  write                   sys_write
2       common  open                    sys_open
3       common  close                   sys_close
4       common  stat                    sys_newstat
5       common  fstat                   sys_newfstat
6       common  lstat                   sys_newlstat
7       common  poll                    sys_poll
8       common  lseek                   sys_lseek
9       common  mmap                    sys_mmap
10      common  mprotect                sys_mprotect
11      common  munmap                  sys_munmap
12      common  brk                     sys_brk
13      64      rt_sigaction            sys_rt_sigaction
14      common  rt_sigprocmask          sys_rt_sigprocmask
15      64      rt_sigreturn            stub_rt_sigreturn
16      64      ioctl                   sys_ioctl
17      common  pread64                 sys_pread64
"syscall_64.tbl" 370 [] --7%--                                  26,1
```

<arch/x86/syscalls/syscall_64.tbl>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/arch/x86/kernel

```
static DEFINE_PER_CPU_PAGE_ALIGNED(char, exception_stacks
        [(N_EXCEPTION_STACKS - 1) * EXCEPTION_STKSZ + DEBUG_STKSZ]);

/* May not be marked __init: used by software suspend */
void syscall_init(void)
{
        /*
         * LSTAR and STAR live in a bit strange symbiosis.
         * They both write to the same internal register. STAR allows to
         * set CS/DS but only a 32bit target. LSTAR sets the 64bit rip.
         */
        wrmsrl(MSR_STAR,  ((u64)__USER32_CS)<<48  | ((u64)__KERNEL_CS)<<32);
        wrmsrl(MSR_LSTAR, system_call);
        wrmsrl(MSR_CSTAR, ignore_sysret);

#ifdef CONFIG_IA32_EMULATION
        syscall32_cpu_init();
#endif

        /* Flags to clear on syscall */
        wrmsrl(MSR_SYSCALL_MASK,
               X86_EFLAGS_TF|X86_EFLAGS_DF|X86_EFLAGS_IF|
               X86_EFLAGS_IOPL|X86_EFLAGS_AC|X86_EFLAGS_NT);
}
```

<arch/x86/kernel/cpu/common.c>

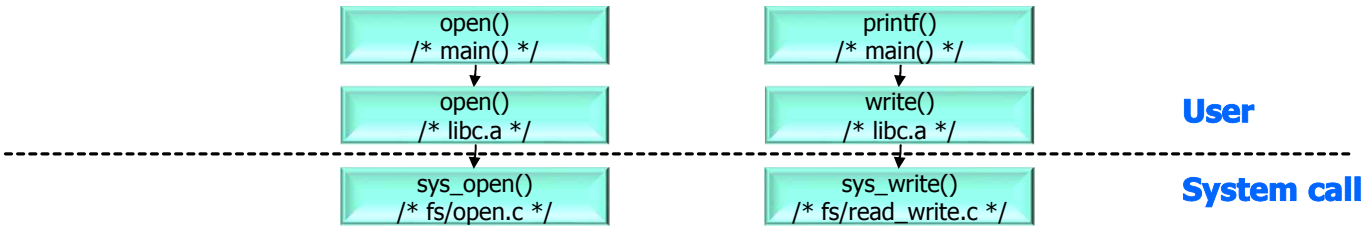
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/arch/x86/kernel

```
ENTRY(system_call)
        CFI_STARTPROC   simple
        CFI_SIGNAL_FRAME
        CFI_DEF_CFA     rsp,KERNEL_STACK_OFFSET
        CFI_REGISTER    rip,rcx
        ...
GLOBAL(system_call_after_swapgs)

        movq    %rsp,PER_CPU_VAR(old_rsp)
        movq    PER_CPU_VAR(kernel_stack),%rsp
        ...
system_call_fastpath:
#if __SYSCALL_MASK == ~0
        cmpq $__NR_syscall_max,%rax
#else
        andl $__SYSCALL_MASK,%eax
        cmpl $__NR_syscall_max,%eax
#endif
        ja ret_from_sys_call  /* and return regs->ax */
        movq %r10,%rcx
        call *sys_call_table(,%rax,8)  # XXX:    rip relative
        movq %rax,RAX-ARGOFFSET(%rsp)
/*
 * Syscall return path ending with SYSRET (fast path)
```

<arch/x86/kernel/entry_64.S>

File open

- Flow control



File open

- Code example: syscall

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
        return fd;
}

SYSCALL_DEFINE3(open, const char __user *, filename, int, flags, umode_t, mode)
{
        if (force_o_largefile())
                flags |= O_LARGEFILE;

        return do_sys_open(AT_FDCWD, filename, flags, mode);
}

SYSCALL_DEFINE4(openat, int, dfd, const char __user *, filename, int, flags,
                umode_t, mode)
{
        if (force_o_largefile())
                flags |= O_LARGEFILE;

        return do_sys_open(dfd, filename, flags, mode);
}

#ifndef __alpha__
```

<fs/open.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/include/linux

```
        asmlinkage long sys_##sname(void)

#define SYSCALL_DEFINE1(name, ...) SYSCALL_DEFINEx(1, _##name, __VA_ARGS__)
#define SYSCALL_DEFINE2(name, ...) SYSCALL_DEFINEx(2, _##name, __VA_ARGS__)
#define SYSCALL_DEFINE3(name, ...) SYSCALL_DEFINEx(3, _##name, __VA_ARGS__)
#define SYSCALL_DEFINE4(name, ...) SYSCALL_DEFINEx(4, _##name, __VA_ARGS__)
#define SYSCALL_DEFINE5(name, ...) SYSCALL_DEFINEx(5, _##name, __VA_ARGS__)
#define SYSCALL_DEFINE6(name, ...) SYSCALL_DEFINEx(6, _##name, __VA_ARGS__)

#define SYSCALL_DEFINEx(x, sname, ...)                          \
        SYSCALL_METADATA(sname, x, __VA_ARGS__)                 \
        __SYSCALL_DEFINEx(x, sname, __VA_ARGS__)

#define __PROTECT(...) asmlinkage_protect(__VA_ARGS__)
#define __SYSCALL_DEFINEx(x, name, ...)                         \
        asmlinkage long sys##name(__MAP(x,__SC_DECL,__VA_ARGS__))       \
                __attribute__((alias(__stringify(SyS##name))));         \
        static inline long SYSC##name(__MAP(x,__SC_DECL,__VA_ARGS__));  \
        asmlinkage long SyS##name(__MAP(x,__SC_LONG,__VA_ARGS__));      \
        asmlinkage long SyS##name(__MAP(x,__SC_LONG,__VA_ARGS__))       \
        {                                                               \
                long ret = SYSC##name(__MAP(x,__SC_CAST,__VA_ARGS__));  \
                __MAP(x,__SC_TEST,__VA_ARGS__);                         \
                __PROTECT(x, ret,__MAP(x,__SC_ARGS,__VA_ARGS__));       \
                return ret;                                             \
        }                                                               \
```

<include/linux/syscalls.h>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
long do_sys_open(int dfd, const char __user *filename, int flags, umode_t mode)
{
        struct open_flags op;
        int fd = build_open_flags(flags, mode, &op);
        struct filename *tmp;

        if (fd)
                return fd;

        tmp = getname(filename);
        if (IS_ERR(tmp))
                return PTR_ERR(tmp);

        fd = get_unused_fd_flags(flags);
        if (fd >= 0) {
                struct file *f = do_filp_open(dfd, tmp, &op);
                if (IS_ERR(f)) {
                        put_unused_fd(fd);
                        fd = PTR_ERR(f);
                } else {
                        fsnotify_open(f);
                        fd_install(fd, f);
                }
        }
        putname(tmp);
        return fd;
}
```

<fs/open.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
        }
        return file;
}

struct file *do_filp_open(int dfd, struct filename *pathname,
                const struct open_flags *op)
{
        struct nameidata nd;
        int flags = op->lookup_flags;
        struct file *filp;

        filp = path_openat(dfd, pathname, &nd, op, flags | LOOKUP_RCU);
        if (unlikely(filp == ERR_PTR(-ECHILD)))
                filp = path_openat(dfd, pathname, &nd, op, flags);
        if (unlikely(filp == ERR_PTR(-ESTALE)))
                filp = path_openat(dfd, pathname, &nd, op, flags | LOOKUP_REVAL
;
        return filp;
}

struct file *do_file_open_root(struct dentry *dentry, struct vfsmount *mnt,
                const char *name, const struct open_flags *op)
{
        struct nameidata nd;
        struct file *file;
        struct filename filename = { .name = name };
        int flags = op->lookup_flags | LOOKUP_ROOT;
```

<fs/namei.c>

File open

- Code example: name resolution

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
static struct file *path_openat(int dfd, struct filename *pathname,
                struct nameidata *nd, const struct open_flags *op, int flags)
{
        struct file *file;
        struct path path;

        file = get_empty_filp();

        file->f_flags = op->open_flag;
        ...

        error = path_init(dfd, pathname->name, flags, nd);
        if (unlikely(error))
                goto out;

        error = do_last(nd, &path, file, op, &opened, pathname);
        while (unlikely(error > 0)) { /* trailing symlink */
                ..
                error = may_follow_link(&link, nd);
                ..
                put_link(nd, &link, cookie);
        }
out:
        path_cleanup(nd);
        ...
        return file;
}
```

<fs/namei.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
static int path_init(int dfd, const char *name, unsigned int flags,
                struct nameidata *nd)
{
        nd->last_type = LAST_ROOT; /* if there are only slashes... */
        ...
        nd->m_seq = read_seqbegin(&mount_lock);
        if (*name=='/') {
                ..
                set_root(nd);
                path_get(&nd->root);
                nd->path = nd->root;
        } else if (dfd == AT_FDCWD) {
                ..
                get_fs_pwd(current->fs, &nd->path);
        } else {
                /* Caller must check execute permissions on the starting path c
mponent */
                dentry = f.file->f_path.dentry;
                if (*name) {
                        if (!d_can_lookup(dentry)) {
                                fdput(f);
                                return -ENOTDIR;
                        }
                }
        }

        nd->inode = nd->path.dentry->d_inode;
        ...
        return -ECHILD;
done:
        current->total_link_count = 0;
        return link_path_walk(name, nd);
}
```

<fs/namei.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
/*
 * Name resolution.
 * This is the basic name resolution function, turning a pathname into
 * the final dentry. We expect 'base' to be positive and a directory.
 *
 * Returns 0 and nd will have valid dentry and mnt on success.
 * Returns error and drops reference to input namei data on failure.
 */
static int link_path_walk(const char *name, struct nameidata *nd)
{
        struct path next;
        int err;

        while (*name=='/')
                name++;
        if (!*name)
                return 0;

        /* At this point we know we have a real path component. */
        for(;;) {
                u64 hash_len;
                int type;

                hash_len = hash_name(name);

                ...
                err = walk_component(nd, &next, LOOKUP_FOLLOW);
                if (err) {
                        err = nested_symlink(&next, nd);
                }
                ..
        }
        terminate_walk(nd);
        return err;
}
```

<fs/namei.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
static inline int walk_component(struct nameidata *nd, struct path *path,
                int follow)
{
        struct inode *inode;
        int err;

        ..
        err = lookup_fast(nd, path, &inode);
        if (unlikely(err)) {
                err = lookup_slow(nd, path);
                if (err < 0)
                        goto out_err;

                inode = path->dentry->d_inode;
        }

        if (should_follow_link(path->dentry, follow)) {
                if (nd->flags & LOOKUP_RCU) {
                        if (unlikely(unlazy_walk(nd, path->dentry))) {
                                err = -ECHILD;
                                goto out_err;
                        }
                }
                return 1;
        }
        path_to_nameidata(path, nd);
        nd->inode = inode;
        return 0;

out_path_put:
        path_to_nameidata(path, nd);
out_err:
        terminate_walk(nd);
        return err;
}
```

<fs/namei.c>

File open

- Code example: name resolution

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
/* Fast lookup failed, do it the slow way */
static int lookup_slow(struct nameidata *nd, struct path *path)
{
        struct dentry *dentry, *parent;
        int err;

        parent = nd->path.dentry;
        BUG_ON(nd->inode != parent->d_inode);

        mutex_lock(&parent->d_inode->i_mutex);
        dentry = __lookup_hash(&nd->last, parent, nd->flags);
        mutex_unlock(&parent->d_inode->i_mutex);
        if (IS_ERR(dentry))
                return PTR_ERR(dentry);
        path->mnt = nd->path.mnt;
        path->dentry = dentry;
        err = follow_managed(path, nd->flags);
        if (unlikely(err < 0)) {
                path_put_conditional(path, nd);
                return err;
        }
        if (err)
                nd->flags |= LOOKUP_JUMPED;
        return 0;
}
```

<fs/namei.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
/*
 * Call i_op->lookup on the dentry.  The dentry must be negative and
 * unhashed.
 *
 * dir->d_inode->i_mutex must be held
 */
static struct dentry *lookup_real(struct inode *dir, struct dentry *dentry,
                        unsigned int flags)
{
        struct dentry *old;

        /* Don't create child dentry for a dead directory. */
        if (unlikely(IS_DEADDIR(dir))) {
                dput(dentry);
                return ERR_PTR(-ENOENT);
        }

        old = dir->i_op->lookup(dir, dentry, flags);
        if (unlikely(old)) {
                dput(dentry);
                dentry = old;
        }
        return dentry;
}

static struct dentry *__lookup_hash(struct qstr *name,
                struct dentry *base, unsigned int flags)
{
        bool need_lookup;
        struct dentry *dentry;

        dentry = lookup_dcache(name, base, flags, &need_lookup);
        if (!need_lookup)
                return dentry;

        return lookup_real(base->d_inode, dentry, flags);
}
```

<fs/namei.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs/ext2

```
}

const struct inode_operations ext2_dir_inode_operations = {
        .create         = ext2_create,
        .lookup         = ext2_lookup,
        .link           = ext2_link,
        .unlink         = ext2_unlink,
        .symlink        = ext2_symlink,
        .mkdir          = ext2_mkdir,
        .rmdir          = ext2_rmdir,
        .mknod          = ext2_mknod,
        .rename         = ext2_rename,
#ifdef CONFIG_EXT2_FS_XATTR
        .setxattr       = generic_setxattr,
        .getxattr       = generic_getxattr,
        .listxattr      = ext2_listxattr,
        .removexattr    = generic_removexattr,
#endif
        .setattr        = ext2_setattr,
        .get_acl        = ext2_get_acl,
        .set_acl        = ext2_set_acl,
        .tmpfile        = ext2_tmpfile,
};
```

<fs/ext2/namei.c>

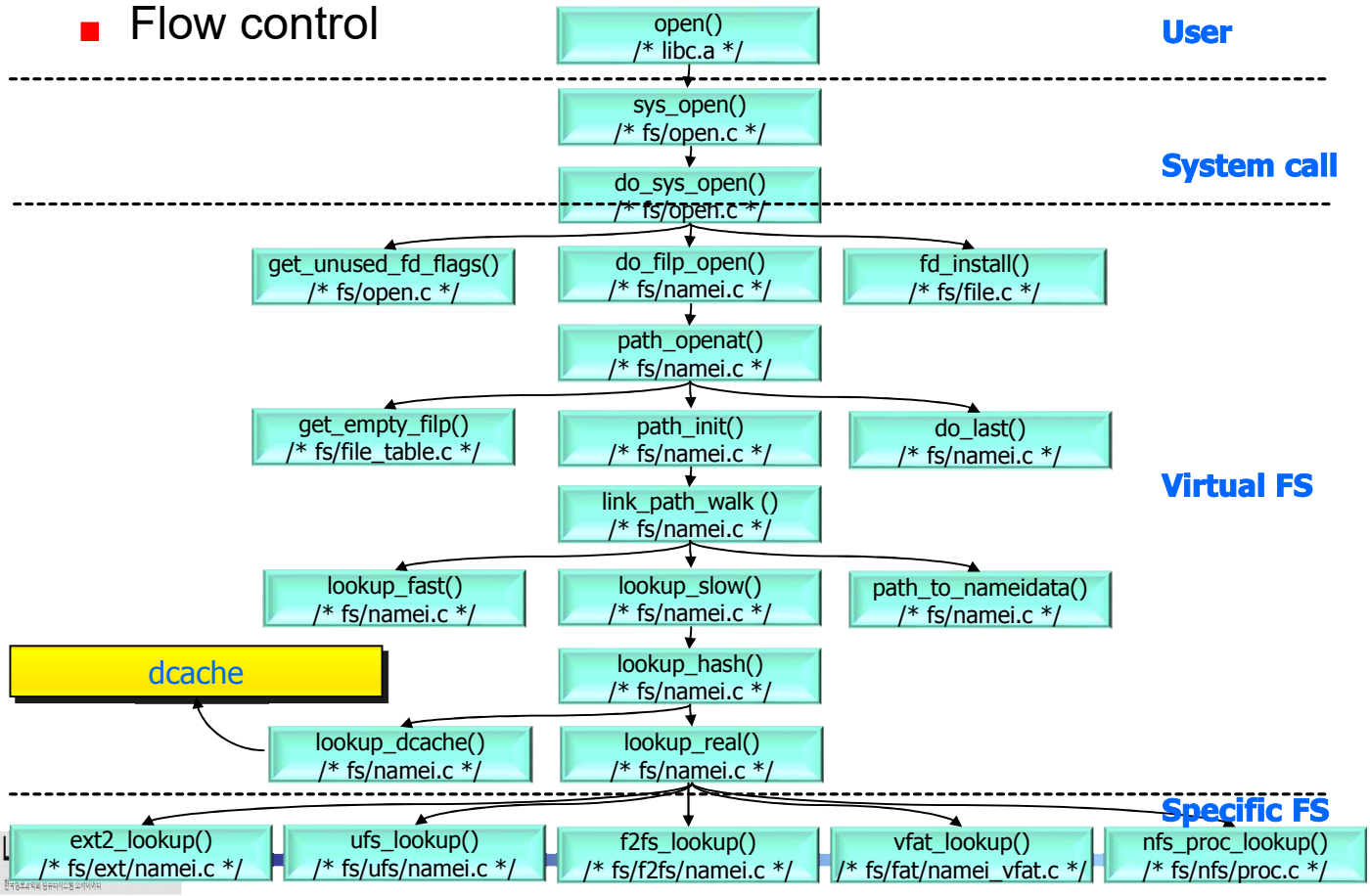
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs/ext2

```
static struct dentry *ext2_lookup(struct inode * dir, struct dentry *dentry,
ignned int flags)
{
        struct inode * inode;
        ino_t ino;

        if (dentry->d_name.len > EXT2_NAME_LEN)
                return ERR_PTR(-ENAMETOOLONG);

        ino = ext2_inode_by_name(dir, &dentry->d_name);
        inode = NULL;
        if (ino) {
                inode = ext2_iget(dir->i_sb, ino);
                if (inode == ERR_PTR(-ESTALE)) {
                        ext2_error(dir->i_sb, __func__,
                                "deleted inode referenced: %lu",
                                (unsigned long) ino);
                        return ERR_PTR(-EIO);
                }
        }
        return d_splice_alias(inode, dentry);
}
```

<fs/ext2/namei.c>

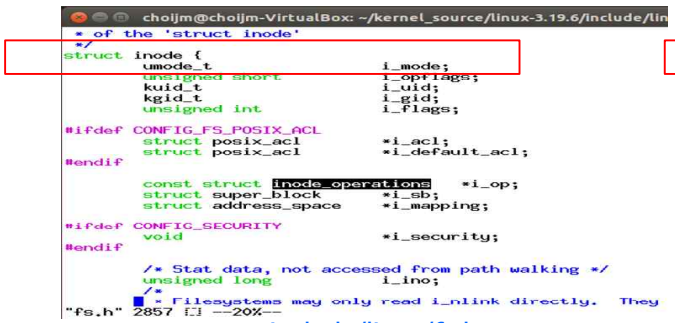
File open

- Flow control

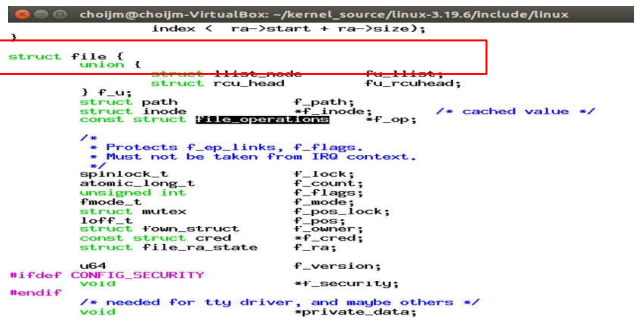


Key data structure

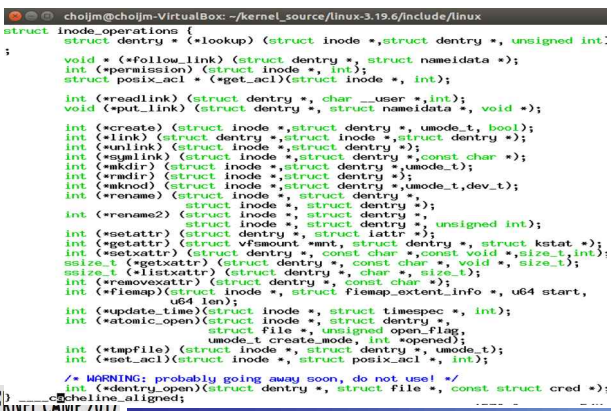
- Key data structures 1: inode and file operations
  - ✓ Definition



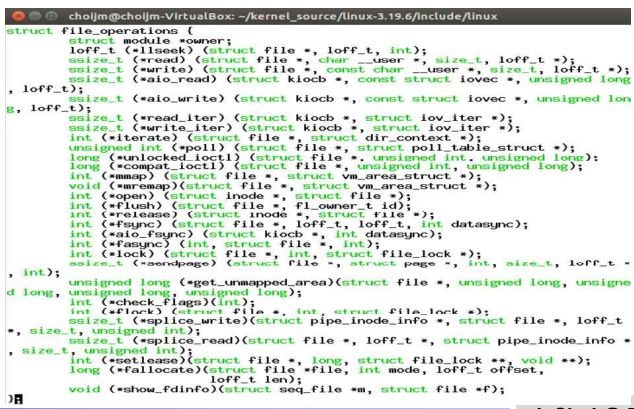
<include/linux/fs.h>



<include/linux/fs.h>



<include/linux/fs.h>



<include/linux/fs.h>

Key data structure

- Key data structures 1: inode and file operations
  - ✓ Various file system support using uniform interface

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
/*
 * We have mostly NULL's here: the current defaults are ok for
 * the ext2 filesystem.
 */
const struct file_operations ext2_file_operations = {
        .llseek         = generic_file_llseek,
        .read           = new_sync_read,
        .write          = new_sync_write,
        .read_iter      = generic_file_read_iter,
        .write_iter     = generic_file_write_iter,
        .unlocked_ioctl = ext2_ioctl,
#ifdef CONFIG_COMPAT
        .compat_ioctl   = ext2_compat_ioctl,
#endif
        .mmap           = generic_file_mmap,
        .open           = dquot_file_open,
        .release        = ext2_release_file,
        .fsync          = ext2_fsync,
        .splice_read    = generic_file_splice_read,
        .splice_write   = iter_file_splice_write,
};

#ifdef CONFIG_EXT2_FS_XIP
const struct file_operations ext2_xip_file_operations = {
        .llseek         = generic_file_llseek,
        .read           = xip_file_read,
```

<fs/ext2/file.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
const struct file_operations fat_file_operations = {
        .llseek         = generic_file_llseek,
        .read           = new_sync_read,
        .write          = new_sync_write,
        .read_iter      = generic_file_read_iter,
        .write_iter     = generic_file_write_iter,
        .mmap           = generic_file_mmap,
        .release        = fat_file_release,
        .unlocked_ioctl = fat_generic_ioctl,
#ifdef CONFIG_COMPAT
        .compat_ioctl   = fat_generic_compat_ioctl,
#endif
        .fsync          = fat_file_fsync,
        .splice_read    = generic_file_splice_read,
};

static int fat_cont_expand(struct inode *inode, loff_t size)
{
        struct address_space *mapping = inode->i_mapping;
        loff_t start = inode->i_size, count = size - inode->i_size;
        int err;

        err = generic_cont_expand_simple(inode, size);
```

<fs/fat/file.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
                break;
        default:
                return -ENOIOCTLCMD;
        }
        return f2fs_ioctl(file, cmd, (unsigned long) compat_ptr(arg));
}
#endif

const struct file_operations f2fs_file_operations = {
        .llseek         = f2fs_llseek,
        .read           = new_sync_read,
        .write          = new_sync_write,
        .read_iter      = generic_file_read_iter,
        .write_iter     = generic_file_write_iter,
        .open           = generic_file_open,
        .release        = f2fs_release_file,
        .mmap           = f2fs_file_mmap,
        .fsync          = f2fs_sync_file,
        .fallocate      = f2fs_fallocate,
        .unlocked_ioctl = f2fs_ioctl,
#ifdef CONFIG_COMPAT
        .compat_ioctl   = f2fs_compat_ioctl,
#endif
        .splice_read    = generic_file_splice_read,
        .splice_write   = iter_file_splice_write,
```

< fs/f2fs/file.c >

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
        /* We're simulating flock() locks using posix locks on the server
        if (fl->fl_type == F_UNLCK)
                return do_unlk(filp, cmd, fl, is_local);
        return do_setlk(filp, cmd, fl, is_local);
}
EXPORT_SYMBOL_GPL(nfs_flock);

const struct file_operations nfs_file_operations = {
        .llseek         = nfs_file_llseek,
        .read           = new_sync_read,
        .write          = new_sync_write,
        .read_iter      = nfs_file_read,
        .write_iter     = nfs_file_write,
        .mmap           = nfs_file_mmap,
        .open           = nfs_file_open,
        .flush          = nfs_file_flush,
        .release        = nfs_file_release,
        .fsync          = nfs_file_fsync,
        .lock           = nfs_lock,
        .flock          = nfs_flock,
        .splice_read    = nfs_file_splice_read,
        .splice_write   = iter_file_splice_write,
        .check_flags    = nfs_check_flags,
        .setlease       = simple_nosetlease,
};
EXPORT_SYMBOL_GPL(nfs_file_operations);
```

< fs/nfs/file.c >

Key data structure

- Key data structures 1: inode and file operations
  - ✓ Flexible for a regular file and directory

choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/ext2

```
        .compat_ioctl   = ext2_compat_ioctl,
#endif
        .mmap           = xip_file_mmap,
        .open           = dquot_file_open,
        .release        = ext2_release_file,
        .fsync          = ext2_fsync,
};
#endif

const struct inode_operations ext2_file_inode_operations = {
#ifdef CONFIG_EXT2_FS_XATTR
        .setxattr       = generic_setxattr,
        .getxattr       = generic_getxattr,
        .listxattr      = ext2_listxattr,
        .removexattr    = generic_removexattr,
#endif
        .setattr        = ext2_setattr,
        .get_acl        = ext2_get_acl,
        .set_acl        = ext2_set_acl,
        .fiemap         = ext2_fiemap,
}
"file.c" 108 [] --100%--                                108,2          [][]
```

<fs/ext2/file.c>

choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/ext2

```
 * the ext2 filesystem.
 */
const struct file_operations ext2_file_operations = {
        .llseek         = generic_file_llseek,
        .read           = new_sync_read,
        .write          = new_sync_write,
        .read_iter      = generic_file_read_iter,
        .write_iter     = generic_file_write_iter,
        .unlocked_ioctl = ext2_ioctl,
#ifdef CONFIG_COMPAT
        .compat_ioctl   = ext2_compat_ioctl,
#endif
        .mmap           = generic_file_mmap,
        .open           = dquot_file_open,
        .release        = ext2_release_file,
        .fsync          = ext2_fsync,
        .splice_read    = generic_file_splice_read,
        .splice_write   = iter_file_splice_write,
};

#ifdef CONFIG_EXT2_FS_XIP
"file.c" 108 [] --75%--                                 81,9           68%
```

<fs/ext2/file.c>

choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/ext2

```
const struct inode_operations ext2_dir_inode_operations = {
        .create         = ext2_create,
        .lookup         = ext2_lookup,
        .link           = ext2_link,
        .unlink         = ext2_unlink,
        .symlink        = ext2_symlink,
        .mkdir          = ext2_mkdir,
        .rmdir          = ext2_rmdir,
        .mknod          = ext2_mknod,
        .rename         = ext2_rename,
#ifdef CONFIG_EXT2_FS_XATTR
        .setxattr       = generic_setxattr,
        .getxattr       = generic_getxattr,
        .listxattr      = ext2_listxattr,
        .removexattr    = generic_removexattr,
#endif
        .setattr        = ext2_setattr,
        .get_acl        = ext2_get_acl,
        .set_acl        = ext2_set_acl,
        .tmpfile        = ext2_tmpfile,
}
"namei.c" 438 [] --97%--                                426,2          97%
```

<fs/ext2/namei.c>

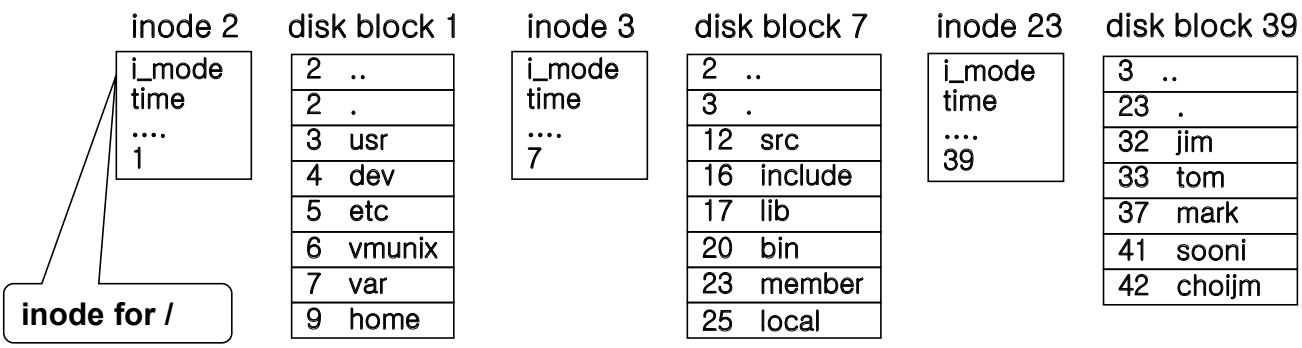
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/ext2

```
                        de = ext2_next_entry(de);
                }
                ext2_put_page(page);
        }
        return 1;

not_empty:
        ext2_put_page(page);
        return 0;
}

const struct file_operations ext2_dir_operations = {
        .llseek         = generic_file_llseek,
        .read           = generic_read_dir,
        .iterate        = ext2_readdir,
        .unlocked_ioctl = ext2_ioctl,
#ifdef CONFIG_COMPAT
        .compat_ioctl   = ext2_compat_ioctl,
#endif
        .fsync          = ext2_fsync,
~
"dir.c" [][] 729 [] --100%--                            729,2-9        [][]
```

<fs/ext2/dir.c>

File open

- Code example: name resolution (cont')
  - ✓ Specific file system layer: ext2 case
    - Read directory contents ➔ ext2_get_page()
    - Compare a file name with each entry ➔ ext2_match()
    - Read the inode matched with the requested file name ➔ ext2_iget()
    - E.g.) /usr/member/choijm



File open

- Code example: name resolution (cont')

```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/ext2
 * Methods themselves.
 */

static struct dentry *ext2_lookup(struct inode * dir, struct dentry *dentry,
unsigned int flags)
{
        struct inode * inode;
        ino_t ino;

        if (dentry->d_name.len > EXT2_NAME_LEN)
                return ERR_PTR(-ENAMETOOLONG);

        ino = ext2_inode_by_name(dir, &dentry->d_name);
        inode = NULL;
        if (ino) {
                inode = ext2_iget(dir->i_sb, ino);
                if (inode == ERR_PTR(-ESTALE)) {
                        ext2_error(dir->i_sb, __func__,
                                   "deleted inode referenced: %lu",
                                   (unsigned long) ino);
                        return ERR_PTR(-EIO);
                }
        }
        return d_splice_alias(inode, dentry);
}
"namei.c" 438 [] --18%--                                  79,0-1        13%
```

<fs/ext2/namei.c>

```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/ext2
        return de;
}

ino_t ext2_inode_by_name(struct inode *dir, struct qstr *child)
{
        ino_t res = 0;
        struct ext2_dir_entry_2 *de;
        struct page *page;

        de = ext2_find_entry (dir, child, &page);
        if (de) {
                res = le32_to_cpu(de->inode);
                ext2_put_page(page);
        }
        return res;
}

static int ext2_prepare_chunk(struct page *page, loff_t pos, unsigned len)
{
        return __block_write_begin(page, pos, len, ext2_get_block);
"dir.c" 730 [] --62%--                                    459,9-16      61%
```

<fs/ext2/dir.c>

```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/ext2
 *      ext2_find_entry():f
 *
 * finds an entry in the specified directory with the wanted name. It
 * returns the page in which the entry was found (as a parameter - res_page
),
 * and the entry itself. Page is returned mapped and unlocked.
 * Entry is guaranteed to be valid.
 */
struct ext2_dir_entry_2 *ext2_find_entry (struct inode * dir,
                        struct qstr *child, struct page ** res_page)
{
        const char *name = child->name;
        ...

        do {
                page = ext2_get_page(dir, n, dir_has_error);
                if (!IS_ERR(page)) {
                        kaddr = page_address(page);
                        de = (ext2_dirent *) kaddr;
                        kaddr += ext2_last_byte(dir, n) - reclen;
                        while ((char *) de <= kaddr) {
                                ...
                                if (ext2_match (namelen, name, de))
                                        goto found;
                                de = ext2_next_entry(de);
                        }
"dir.c" [[]] 707 [] --50%--                               360,2         52%
```

<fs/ext2/dir.c>

```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/ext2
static struct page * ext2_get_page(struct inode *dir, unsigned long n,
                                   int quiet)
{
        struct address_space *mapping = dir->i_mapping;
        struct page *page = read_mapping_page(mapping, n, NULL);
        if (!IS_ERR(page)) {
                kmap(page);
                if (!PageChecked(page))
                        ext2_check_page(page, quiet);
                if (PageError(page))
                        goto fail;
        }
        return page;

fail:
        ext2_put_page(page);
        return ERR_PTR(-EIO);
}

/*
"dir.c" [[]] 713 [] --30%--                               221,2
```

<fs/ext2/dir.c>

File open

- Code example: name resolution (cont')

```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/ext2
struct inode *ext2_iget (struct super_block *sb, unsigned long ino)
{
        struct ext2_inode_info *ei;
        struct buffer_head * bh;
        struct ext2_inode *raw_inode;
        struct inode *inode;
        long ret = -EIO;
        int n;
        uid_t i_uid;
        gid_t i_gid;

        inode = iget_locked(sb, ino);
        if (!inode)
                return ERR_PTR(-ENOMEM);
        if (!(inode->i_state & I_NEW))
                return inode;

        ei = EXT2_I(inode);
        ei->i_block_alloc_info = NULL;

        raw_inode = ext2_get_inode(inode->i_sb, ino, &bh);
        if (IS_ERR(raw_inode)) {
                ret = PTR_ERR(raw_inode);
                goto bad_inode;
        }
"inode.c" 1573 [] --84%--                                  1332,2-9        84%
```

<fs/ext2/inode.c>

```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/ext2
}

static struct ext2_inode *ext2_get_inode(struct super_block *sb, ino_t ino,
                                         struct buffer_head **p)
{
        struct buffer_head * bh;
        ...

        block_group = (ino - 1) / EXT2_INODES_PER_GROUP(sb);
        gdp = ext2_get_group_desc(sb, block_group, NULL);
        if (!gdp)
                goto Egdp;
        /*
         * Figure out the offset within the block group inode table
         */
        offset = ((ino - 1) % EXT2_INODES_PER_GROUP(sb)) * EXT2_INODE_SIZE(
sb);
        block = le32_to_cpu(gdp->bg_inode_table) +
                (offset >> EXT2_BLOCK_SIZE_BITS(sb));
        if (!(bh = sb_bread(sb, block)))
                goto Eio;

        *p = bh;
```

< fs/ext2/inode.c >

```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/ext2
#define in_range(b, first, len) ((b) >= (first) && (b) <= (first) + (len) -
 1)

struct ext2_group_desc * ext2_get_group_desc(struct super_block * sb,
                                             unsigned int block_group,
                                             struct buffer_head ** bh)
{
        unsigned long group_desc;
        unsigned long offset;
        struct ext2_group_desc * desc;
        struct ext2_sb_info *sbi = EXT2_SB(sb);

        if (block_group >= sbi->s_groups_count) {
                ext2_error (sb, "ext2_get_group_desc",
                            "block_group >= groups_count - "
                            "block_group = %d, groups_count = %lu",
                            block_group, sbi->s_groups_count);

                return NULL;
        }

        group_desc = block_group >> EXT2_DESC_PER_BLOCK_BITS(sb);
        offset = block_group & (EXT2_DESC_PER_BLOCK(sb) - 1);
        if (!sbi->s_group_desc[group_desc]) {
                ext2_error (sb, "ext2_get_group_desc",
                            "Group descriptor not loaded - "
"balloc.c" 1536 [] --3%--                                  61,10-31        2%
```

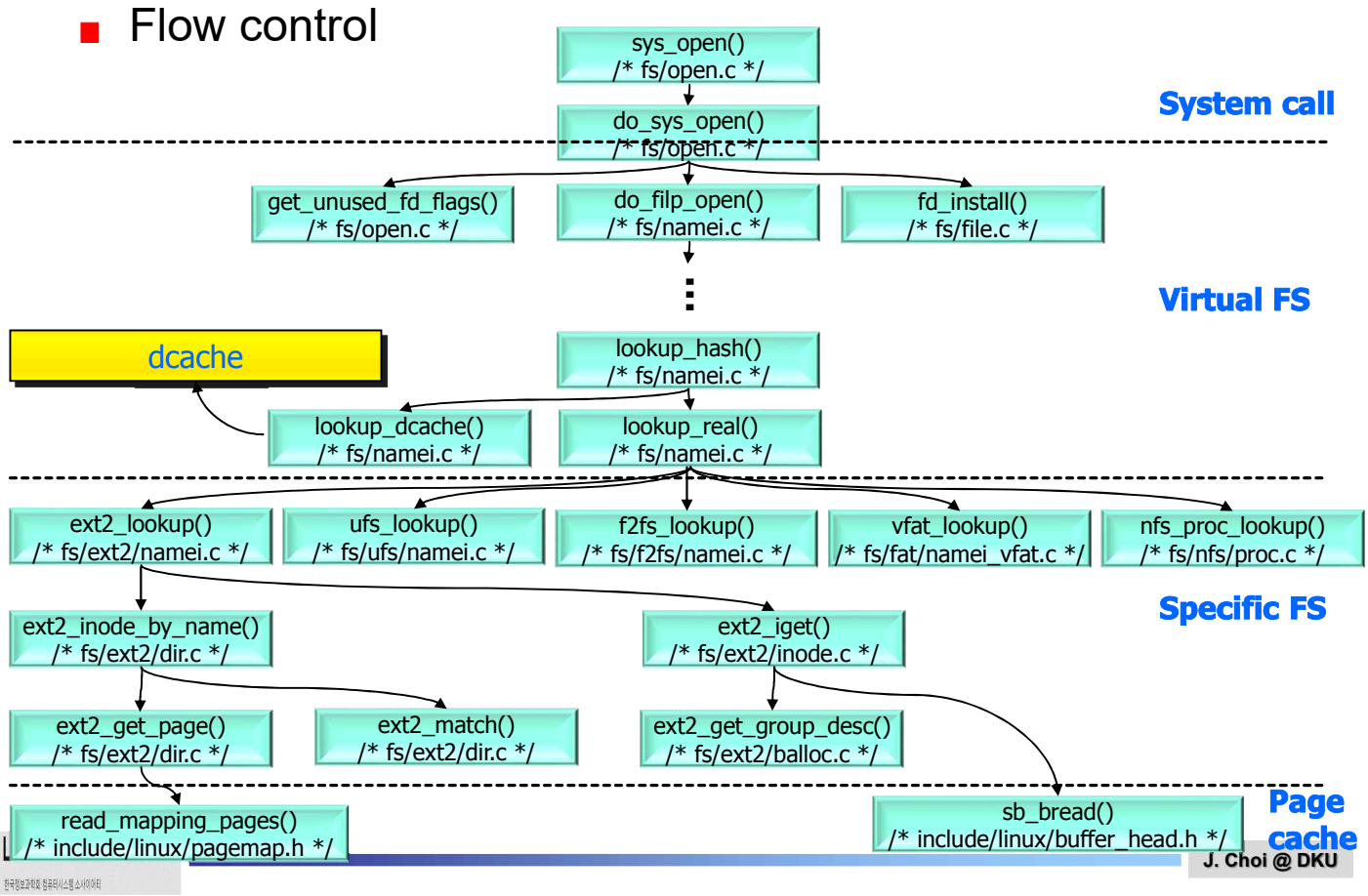
<fs/ext2/balloc.c>

```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/include/linux
                __bforget(bh);
}

static inline struct buffer_head *
sb_bread(struct super_block *sb, sector_t block)
{
        return __bread_gfp(sb->s_bdev, block, sb->s_blocksize, __G
FP_MOVABLE);
}

static inline struct buffer_head *
sb_bread_unmovable(struct super_block *sb, sector_t block)
{
        return __bread_gfp(sb->s_bdev, block, sb->s_blocksize, 0);
}

static inline void
sb_breadahead(struct super_block *sb, sector_t block)
{
"buffer_head.h" 396 [] --76%--                             301,0-1         77%
```

<include/linux/buffer_head.h>

File open

- Flow control



File open

- Code example: do_last

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
/*
 * Handle the last step of open()
 */
static int do_last(struct nameidata *nd, struct path *path,
                   struct file *file, const struct open_flags *op,
                   int *opened, struct filename *name)
{
        struct dentry *dir = nd->path.dentry;
        ...
finish_open:
        error = complete_walk(nd);
        audit_inode(name, nd->path.dentry, 0);
        ...
        if (will_truncate) {
                error = mnt_want_write(nd->path.mnt);
                if (error)
                        goto out;
                got_write = true;
        }
finish_open_created:
        error = may_open(&nd->path, acc_mode, open_flag);
        if (error)
                goto out;
        BUG_ON(*opened & FILE_OPENED); /* once it's opened, it's opened */
        error = vfs_open(&nd->path, file, current_cred());
        if (!error) {
                *opened |= FILE_OPENED;
        } else {
                if (error == -EOPENSTALE)
                        goto stale_open;
                goto out;
```

<fs/namei.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
/**
 * vfs_open - open the file at the given path
 * @path: path to open
 * @filp: newly allocated file with f_flag initialized
 * @cred: credentials to use
 */
int vfs_open(const struct path *path, struct file *filp,
             const struct cred *cred)
{
        struct inode *inode = path->dentry->d_inode;

        if (inode->i_op->dentry_open)
                return inode->i_op->dentry_open(path->dentry, filp, cred);
        else {
                filp->f_path = *path;
                return do_dentry_open(filp, NULL, cred);
        }
}
EXPORT_SYMBOL(vfs_open);

static inline int build_open_flags(int flags, umode_t mode, struct open_flags
p)
{
```

<fs/open.c>

```
static int do_dentry_open(struct file *f,
                          int (*open)(struct inode *, struct file *),
                          const struct cred *cred)
{
        ...
        inode = f->f_inode = f->f_path.dentry->d_inode;
        f->f_mapping = inode->i_mapping;

        f->f_op = fops_get(inode->i_fop);

        error = security_file_open(f, cred);

        if (!open)
                open = f->f_op->open;
        if (open) {
                error = open(inode, f);
                if (error)
                        goto cleanup_all;
        }
        if ((f->f_mode & FMODE_READ) &&
            likely(f->f_op->read || f->f_op->aio_read || f->f_op->read_iter))
                f->f_mode |= FMODE_CAN_READ;
        if ((f->f_mode & FMODE_WRITE) &&
            likely(f->f_op->write || f->f_op->aio_write || f->f_op->write_iter)
```

<fs/open.c>

```
const struct file_operations ext2_file_operations = {
        .llseek         = generic_file_llseek,
        .read           = new_sync_read,
        .write          = new_sync_write,
        .read_iter      = generic_file_read_iter,
        .write_iter     = generic_file_write_iter,
        .unlocked_ioctl = ext2_ioctl,
#ifdef CONFIG_COMPAT
        .compat_ioctl   = ext2_compat_ioctl,
#endif
        .mmap           = generic_file_mmap,
        .open           = dquot_file_open,
        .release        = ext2_release_file,
        .fsync          = ext2_fsync,
        .splice_read    = generic_file_splice_read,
        .splice_write   = iter_file_splice_write,
};
```

Standard(generic) functions supported by VFS

<fs/ext2/file.c>

File open

- Code example: file-specific open

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
/*
 * Generic helper for ->open on filesystems supporting disk quotas.
 */
int dquot_file_open(struct inode *inode, struct file *file)
{
        int error;

        error = generic_file_open(inode, file);
        if (!error && (file->f_mode & FMODE_WRITE))
                dquot_initialize(inode);
        return error;
}
EXPORT_SYMBOL(dquot_file_open);
```

<fs/quota/dquot.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
/*
 * Called when an inode is about to be open.
 * We use this to disallow opening large files on 32bit systems if
 * the caller didn't specify O_LARGEFILE.  On 64bit systems we force
 * on this flag in sys_open.
 */
int generic_file_open(struct inode * inode, struct file * filp)
{
        if (!(filp->f_flags & O_LARGEFILE) && i_size_read(inode) > MAX_NON_LFS)
                return -EOVERFLOW;
        return 0;
}

EXPORT_SYMBOL(generic_file_open);
```

<fs/open.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
                break;
        default:
                return -ENOIOCTLCMD;
        }
        return f2fs_ioctl(file, cmd, (unsigned long) compat_pt
}
#endif

const struct file_operations f2fs_file_operations = {
        .llseek         = f2fs_llseek,
        .read           = new_sync_read,
        .write          = new_sync_write,
        .read_iter      = generic_file_read_iter,
        .write_iter     = generic_file_write_iter,
        .open           = generic_file_open,
        .release        = f2fs_release_file,
        .mmap           = f2fs_file_mmap,
        .fsync          = f2fs_sync_file,
        .fallocate      = f2fs_fallocate,
        .unlocked_ioctl = f2fs_ioctl,
#ifdef CONFIG_COMPAT
        .compat_ioctl   = f2fs_compat_ioctl,
#endif
        .splice_read    = generic_file_splice_read,
        .splice_write   = iter_file_splice_write,
};
```

<fs/f2fs/file.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs/fat

```
const struct file_operations fat_file_operations = {
        .llseek         = generic_file_llseek,
        .read           = new_sync_read,
        .write          = new_sync_write,
        .read_iter      = generic_file_read_iter,
        .write_iter     = generic_file_write_iter,
        .mmap           = generic_file_mmap,
        .release        = fat_file_release,
        .unlocked_ioctl = fat_generic_ioctl,
#ifdef CONFIG_COMPAT
        .compat_ioctl   = fat_generic_compat_ioctl,
#endif
        .fsync          = fat_file_fsync,
        .splice_read    = generic_file_splice_read,
};

static int fat_cont_expand(struct inode *inode, loff_t size)
{
        struct address_space *mapping = inode->i_mapping;
        loff_t start = inode->i_size, count = size - inode->i_size;
```

<fs/fat/file.c>

File open

- Code example: file-specific open

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/drivers

```
/*
 * The File Operations structure for the serial/ioctl interface of the
 */
static const struct file_operations megadev_fops = {
        .owner          = THIS_MODULE,
        .unlocked_ioctl = megadev_unlocked_ioctl,
        .open           = megadev_open,
        .llseek         = noop_llseek,
};

/*
 * Array to structures for storing the information about the controller
 * information is sent to the user level applications, when they do an
 * for this information.
 */
static struct mcontroller mcontroller[MAX_CONTROLLERS];

/* The current driver version */
static u32 driver_ver = 0x02000000;

/* major number used by the device for character interface */
```

<drivers/scsi/megaraid.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/drivers

```
static const struct file_operations tty_fops = {
        .llseek         = no_llseek,
        .read           = tty_read,
        .write          = tty_write,
        .poll           = tty_poll,
        .unlocked_ioctl = tty_ioctl,
        .compat_ioctl   = tty_compat_ioctl,
        .open           = tty_open,
        .release        = tty_release,
        .fasync         = tty_fasync,
};

static const struct file_operations console_fops = {
        .llseek         = no_llseek,
        .read           = tty_read,
        .write          = redirected_tty_write,
        .poll           = tty_poll,
        .unlocked_ioctl = tty_ioctl,
        .compat_ioctl   = tty_compat_ioctl,
        .open           = tty_open,
        .release        = tty_release,
        .fasync         = tty_fasync,
};
```

<drivers/tty/tty_io.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/drivers

```
                        seq_printf(m, "Hop index        = %d\n",
                                   pfh->hop_index);
                        p += p[1] + 2;
                } else {
                        seq_puts(m,
                                 "Parse beacon failed at FH param element\
                        return 0;
                }
        } else {
                seq_puts(m, "No beacons received\n");
        }
        return 0;
}

static int ray_cs_proc_open(struct inode *inode, struct file *file)
{
        return single_open(file, ray_cs_proc_show, NULL);
}

static const struct file_operations ray_cs_proc_fops = {
        .owner = THIS_MODULE,
        .open = ray_cs_proc_open,
        .read = seq_read,
        .llseek = seq_lseek,
        .release = single_release,
};
```

<drivers/net/wireless/ray_cs.c>

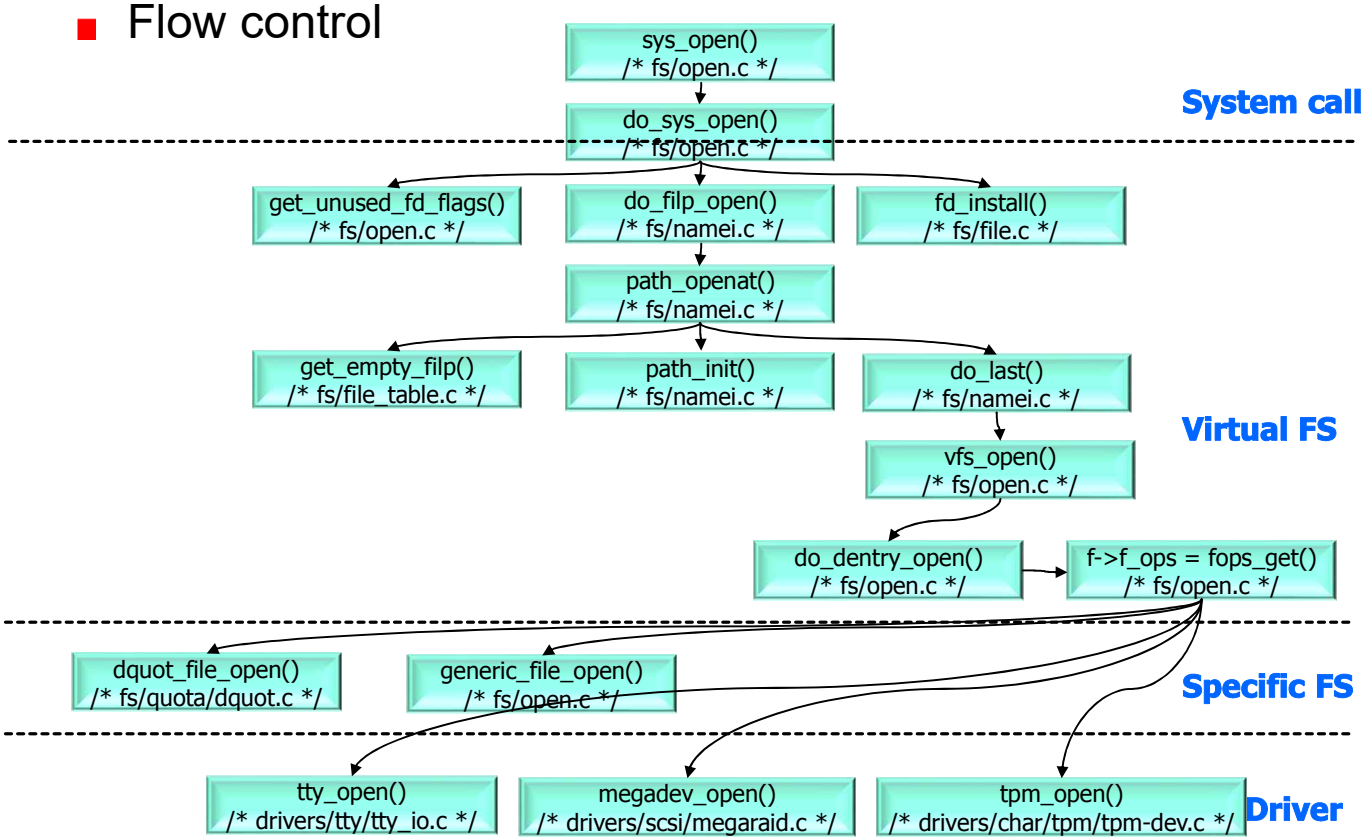
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/drivers

```
        clear_bit(0, &priv->chip->is_open);
        put_device(priv->chip->dev);
        kfree(priv);
        return 0;
}

static const struct file_operations tpm_fops = {
        .owner = THIS_MODULE,
        .llseek = no_llseek,
        .open = tpm_open,
        .read = tpm_read,
        .write = tpm_write,
        .release = tpm_release,
};

int tpm_dev_add_device(struct tpm_chip *chip)
{
        int rc;

        chip->vendor.miscdev.fops = &tpm_fops;
        if (chip->dev_num == 0)
                chip->vendor.miscdev.minor = TPM_MINOR;
```

<drivers/char/tpm/tpm-dev.c>

File open

- Flow control



File open: summary

- Comments from W. Mauerer
  - ✓ Name resolution code (e.g. link_path_walk() or namei()) is one of the most complicate codes in Linux
    - Need to traverse all components (/home/choijm/test.c)
    - Need to handle link, mounted point, .., .
    - Need to check the access permission
    - Need to take care of special files such as device file
    - Aggressively use dcache (since it is called frequently)
    - Manycore consideration: Using RCU (Read Copy Update) lock (A replacement for reader-writer locking in Linux)

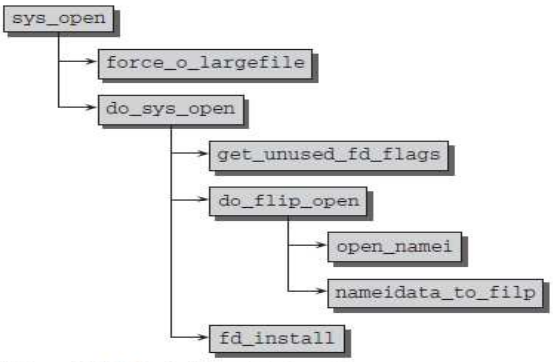


Figure 8-11: Code flow diagram for sys_open.

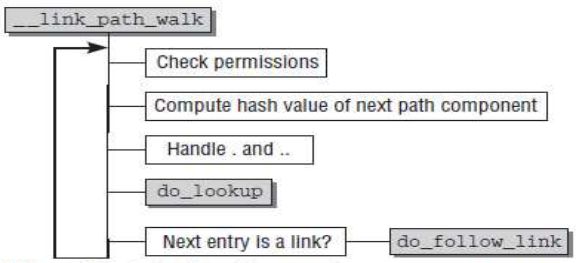


Figure 8-9: Code flow diagram for __link_path_walk.

(Source: Professional Linux Kernel Architecture, Wrox)

File open: summary

- Relation among data structures

struct task_struct
/* include/linux/sched.h */

```
....
struct files_struct *files;
struct fs_struct *fs;
....
```

struct file
/* include/linux/fs.h*/

```
struct path f_path (vfs_mount, dentry);
struct inode *f_inode;
struct file_operations *f_op;
spinlock_t f_lock;
struct address_space *f_mapping;
.......
```

struct super_block
/* include/linux/fs.h */

```
....
dev_t s_dev;
struct files_system_type s_type;
struct super_operations *s_op;
....
```

struct files_struct
/* include/linux/fdtable.h */

```
...
atomic_t count;
struct file *fd_array[] ;
long close_on_exec_init;
...........
```

struct dentry
/* include/linux/dcache.h */

```
int d_count, d_flags;
struct super_block *d_sb;
struct inode *d_inode;
struct dentry_operation *d_op;
....
```

struct address_space
/* include/linux/fs.h */

```
struct inode *host;
struct radix_tree_root page_tree;
....
struct address_space_operations *a_ops;
....
```

struct inode
/* include/linux/fs.h */

```
int i_ino
umode_t
....
struct inode_operations i_op
struct address_space *i_mapping;
....
i_dentry
```

File read and write

- Code example: syscall and VFS

```
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs
SYSCALL_DEFINE3(read, unsigned int, fd, char __user *, buf, size_t, count)
{
        struct fd f = fdget_pos(fd);
        ssize_t ret = -EBADF;

        if (f.file) {
                loff_t pos = file_pos_read(f.file);
                ret = vfs_read(f.file, buf, count, &pos);
                if (ret >= 0)
                        file_pos_write(f.file, pos);
                fdput_pos(f);
        }
        return ret;
}

SYSCALL_DEFINE3(write, unsigned int, fd, const char __user *, buf,
                size_t, count)
{
        struct fd f = fdget_pos(fd);
        ssize_t ret = -EBADF;

        if (f.file) {
                loff_t pos = file_pos_read(f.file);
                ret = vfs_write(f.file, buf, count, &pos);
                if (ret >= 0)
                        file_pos_write(f.file, pos);
                fdput_pos(f);
        }
        return ret;
}
                                                        574,9         42%
```

<fs/read_write.c>

```
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs
        }

        return ret;
}

EXPORT_SYMBOL(vfs_write);

static inline loff_t file_pos_read(struct file *file)
{
        return file->f_pos;
}

static inline void file_pos_write(struct file *file, loff_t pos)
{
        file->f_pos = pos;
}

SYSCALL_DEFINE3(read, unsigned int, fd, char __user *, buf, size_t, count)
{
        struct fd f = fdget_pos(fd);
        ssize_t ret = -EBADF;

        if (f.file) {
                loff_t pos = file_pos_read(f.file);
                ret = vfs_read(f.file, buf, count, &pos);
                if (ret >= 0)
                        file_pos_write(f.file, pos);
```

<fs/read_write.c>

```
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs
ssize_t __vfs_read(struct file *file, char __user *buf, size_t count,
                   loff_t *pos)
{
        ssize_t ret;

        if (file->f_op->read)
                ret = file->f_op->read(file, buf, count, pos);
        else if (file->f_op->aio_read)
                ret = do_sync_read(file, buf, count, pos);
        else if (file->f_op->read_iter)
                ret = new_sync_read(file, buf, count, pos);
        else
                ret = -EINVAL;

        return ret;
}

ssize_t vfs_read(struct file *file, char __user *buf, size_t count, loff_t *pos
{
        ssize_t ret;

        if (!(file->f_mode & FMODE_READ))
                return -EBADF;
        if (!(file->f_mode & FMODE_CAN_READ))
                return -EINVAL;
        if (unlikely(!access_ok(VERIFY_WRITE, buf, count)))
                return -EFAULT;

        ret = rw_verify_area(READ, file, pos, count);
        if (ret >= 0) {
                count = ret;
                ret = __vfs_read(file, buf, count, pos);
                if (ret > 0) {
                        fsnotify_access(file);
                                                        414,0-1       30%
```

<fs/read_write.c>

```
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs
EXPORT_SYMBOL(__kernel_write);

ssize_t vfs_write(struct file *file, const char __user *buf, size_t count, loff
t *pos)
{
        ssize_t ret;

        if (!(file->f_mode & FMODE_WRITE))
                return -EBADF;
        if (!(file->f_mode & FMODE_CAN_WRITE))
                return -EINVAL;
        if (unlikely(!access_ok(VERIFY_READ, buf, count)))
                return -EFAULT;

        ret = rw_verify_area(WRITE, file, pos, count);
        if (ret >= 0) {
                count = ret;
                file_start_write(file);
                if (file->f_op->write)
                        ret = file->f_op->write(file, buf, count, pos);
                else if (file->f_op->aio_write)
                        ret = do_sync_write(file, buf, count, pos);
                else
                        ret = new_sync_write(file, buf, count, pos);
                if (ret > 0) {
                        fsnotify_modify(file);
                        add_wchar(current, ret);
                }
                inc_syscw(current);
                file_end_write(file);
        }

        return ret;
}
```

<fs/read_write.c>

File read and write

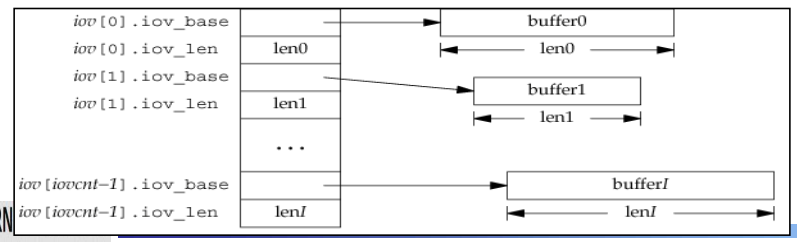
- Read/write interface in file_operations
  - ✓ read/write
    - normal read/write
  - ✓ aio_read/write (removable)
    - vectored (iovec), asynchronous
  - ✓ read_iter/write_iter
    - iov_iter
    - an iterator for several iovec

```
struct iovec
{
    void __user *iov_base;
    __kernel_size_t iov_len;
};
```

/* include/uapi/linux/sched.h */

```
struct iov_iter {
    int type;
    size_t iov_offset;
    size_t count;
    const struct iovec *iov; /* SIMPLIF
    unsigned long nr_segs;
};
```

/* include/linux/sched.h */




```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t
*);
        ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned l
ong, loff_t);
        ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned
long, loff_t);
        ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
        ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
        int (*iterate) (struct file *, struct dir_context *);
LibreOffice Impress int (*poll) (struct file *, struct poll_table_struct *);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        void (*mremap)(struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, loff_t, loff_t, int datasync);
        int (*aio_fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_
t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsi
gned long, unsigned long, unsigned long);
        int (*check_flags)(int);
        int (*flock) (struct file *, int, struct file_lock *);
        ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff
_t *, size_t, unsigned int);
        ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_inf
o *, size_t, unsigned int);
        int (*setlease)(struct file *, long, struct file_lock **, void **);
        long (*fallocate)(struct file *file, int mode, loff_t offset,
                          loff_t len);
        void (*show_fdinfo)(struct seq_file *m, struct file *f);
};
"../include/linux/fs.h" 2857 [] --52%--                1507,21       53%
```

File read and write

- Code example: file_operations for FSs

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
/*
 * We have mostly NULL's here: the current defaults are ok for
 * the ext2 filesystem.
 */
const struct file_operations ext2_file_operations = {
        .llseek         = generic_file_llseek,
        .read           = new_sync_read,
        .write          = new_sync_write,
        .read_iter      = generic_file_read_iter,
        .write_iter     = generic_file_write_iter,
        .unlocked_ioctl = ext2_ioctl,
#ifdef CONFIG_COMPAT
        .compat_ioctl   = ext2_compat_ioctl,
#endif
        .mmap           = generic_file_mmap,
        .open           = dquot_file_open,
        .release        = ext2_release_file,
        .fsync          = ext2_fsync,
        .splice_read    = generic_file_splice_read,
        .splice_write   = iter_file_splice_write,
};

#ifdef CONFIG_EXT2_FS_XIP
const struct file_operations ext2_xip_file_operations = {
        .llseek         = generic_file_llseek,
        .read           = xip_file_read,
```

<fs/ext2/file.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
const struct file_operations fat_file_operations = {
        .llseek         = generic_file_llseek,
        .read           = new_sync_read,
        .write          = new_sync_write,
        .read_iter      = generic_file_read_iter,
        .write_iter     = generic_file_write_iter,
        .mmap           = generic_file_mmap,
        .release        = fat_file_release,
        .unlocked_ioctl = fat_generic_ioctl,
#ifdef CONFIG_COMPAT
        .compat_ioctl   = fat_generic_compat_ioctl,
#endif
        .fsync          = fat_file_fsync,
        .splice_read    = generic_file_splice_read,
};

static int fat_cont_expand(struct inode *inode, loff_t size)
{
        struct address_space *mapping = inode->i_mapping;
        loff_t start = inode->i_size, count = size - inode->i_size;
        int err;

        err = generic_cont_expand_simple(inode, size);
```

Standard(generic) functions supported by VFS

<fs/fat/file.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
                break;
        default:
                return -ENOIOCTLCMD;
        }
        return f2fs_ioctl(file, cmd, (unsigned long) compat_ptr(arg));
}
#endif

const struct file_operations f2fs_file_operations = {
        .llseek         = f2fs_llseek,
        .read           = new_sync_read,
        .write          = new_sync_write,
        .read_iter      = generic_file_read_iter,
        .write_iter     = generic_file_write_iter,
        .open           = generic_file_open,
        .release        = f2fs_release_file,
        .mmap           = f2fs_file_mmap,
        .fsync          = f2fs_sync_file,
        .fallocate      = f2fs_fallocate,
        .unlocked_ioctl = f2fs_ioctl,
#ifdef CONFIG_COMPAT
        .compat_ioctl   = f2fs_compat_ioctl,
#endif
        .splice_read    = generic_file_splice_read,
        .splice_write   = iter_file_splice_write,
```

< fs/f2fs/file.c >

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
        /* We're simulating flock() locks using posix locks on the server
        if (fl->fl_type == F_UNLCK)
                return do_unlk(filp, cmd, fl, is_local);
        return do_setlk(filp, cmd, fl, is_local);
}
EXPORT_SYMBOL_GPL(nfs_flock);

const struct file_operations nfs_file_operations = {
        .llseek         = nfs_file_llseek,
        .read           = new_sync_read,
        .write          = new_sync_write,
        .read_iter      = nfs_file_read,
        .write_iter     = nfs_file_write,
        .mmap           = nfs_file_mmap,
        .open           = nfs_file_open,
        .flush          = nfs_file_flush,
        .release        = nfs_file_release,
        .fsync          = nfs_file_fsync,
        .lock           = nfs_lock,
        .flock          = nfs_flock,
        .splice_read    = nfs_file_splice_read,
        .splice_write   = iter_file_splice_write,
        .check_flags    = nfs_check_flags,
        .setlease       = simple_nosetlease,
};
EXPORT_SYMBOL_GPL(nfs_file_operations);
```

< fs/nfs/file.c >

File read and write

- Code example: generic file system routines

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
        *ppos = kiocb.ki_pos;
        return ret;
}
EXPORT_SYMBOL(do_sync_read);

ssize_t new_sync_read(struct file *filp, char __user *buf, size_t len, loff_t *p
pos)
{
        struct iovec iov = { .iov_base = buf, .iov_len = len };
        struct kiocb kiocb;
        struct iov_iter iter;
        ssize_t ret;

        init_sync_kiocb(&kiocb, filp);
        kiocb.ki_pos = *ppos;
        kiocb.ki_nbytes = len;
        iov_iter_init(&iter, READ, &iov, 1, len);

        ret = filp->f_op->read_iter(&kiocb, &iter);
        if (-EIOCBQUEUED == ret)
                ret = wait_on_sync_kiocb(&kiocb);
        *ppos = kiocb.ki_pos;
        return ret;
}

EXPORT_SYMBOL(new_sync_read);

ssize_t __vfs_read(struct file *file, char __user *buf, size_t count,
                   loff_t *pos)
{
        ssize_t ret;

        if (file->f_op->read)
```

<fs/read_write.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/mm

```
/**
 * generic_file_read_iter - generic filesystem read routine
 * @iocb:       kernel I/O control block
 * @iter:       destination for the data read
 *
 * This is the "read_iter()" routine for all filesystems
 * that can use the page cache directly.
 */
ssize_t
generic_file_read_iter(struct kiocb *iocb, struct iov_iter *iter)
{
        struct file *file = iocb->ki_filp;
        ssize_t retval = 0;
        loff_t *ppos = &iocb->ki_pos;
        loff_t pos = *ppos;

        /* coalesce the iovecs and go direct-to-BIO for O_DIRECT */
        if (file->f_flags & O_DIRECT) {
                ...
                retval = filemap_write_and_wait_range(mapping, pos,
                                                      pos + count - 1);
                if (!retval) {
                        struct iov_iter data = *iter;
                        retval = mapping->a_ops->direct_IO(READ, iocb, &data, pos);
                }
                ...
        }

        retval = do_generic_file_read(file, ppos, iter, retval);
out:
        return retval;
}
```

<mm/filemap.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/mm

```
static ssize_t do_generic_file_read(struct file *filp, loff_t *ppos,
                    struct iov_iter *iter, ssize_t written)
{
        struct address_space *mapping = filp->f_mapping;
        struct inode *inode = mapping->host;

        ...
        for (;;) {
                ...
find_page:
                page = find_get_page(mapping, index);
                if (!page) {
                        ...
                        goto no_cached_page;
                }
                /* handling readahead and Update */
                ...
page_ok:
                ret = copy_page_to_iter(page, offset, nr, iter);
                if (!iov_iter_count(iter))
                        goto out;
                continue;
                ...
readpage:
                /* Start the actual read. The read will unlock the page. */
                error = mapping->a_ops->readpage(filp, page);
                ...
                goto page_ok;

no_cached_page:
                page = page_cache_alloc_cold(mapping);
                error = add_to_page_cache_lru(page, mapping,
                goto readpage;
        }

out:
        file_accessed(filp);
        return written ? written : error;
```

<mm/filemap.c>

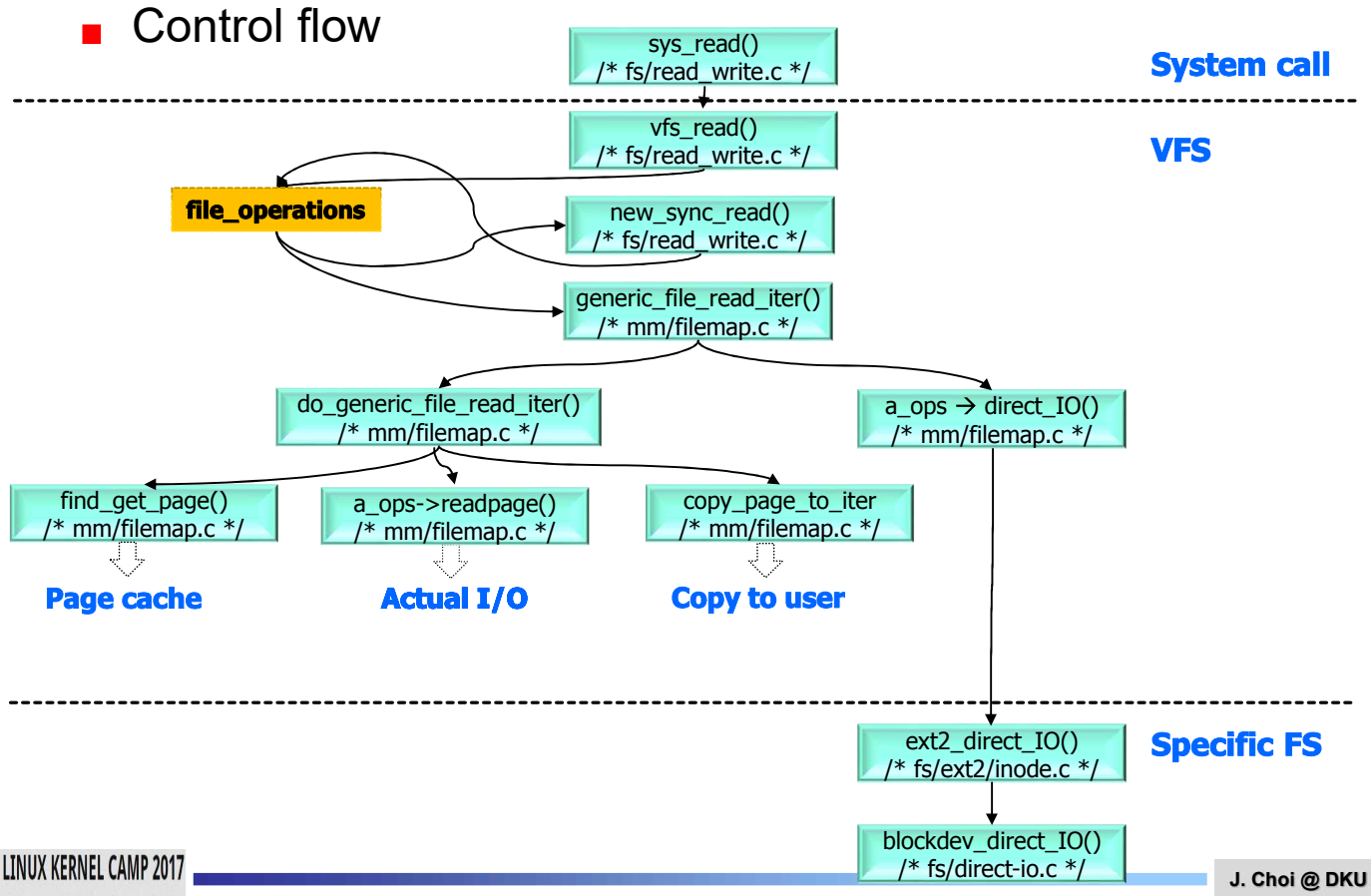
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/mm

```
        return generic_block_bmap(mapping,block,ext2_get_block);
}

static ssize_t
ext2_direct_IO(int rw, struct kiocb *iocb, struct iov_iter *iter,
               loff_t offset)
{
        struct file *file = iocb->ki_filp;
        struct address_space *mapping = file->f_mapping;
        struct inode *inode = mapping->host;
        size_t count = iov_iter_count(iter);
        ssize_t ret;

        ret = blockdev_direct_IO(rw, iocb, inode, iter, offset, ext2_get_bloc
k);
        if (ret < 0 && (rw & WRITE))
                ext2_write_failed(mapping, offset + count);
        return ret;
}
"../fs/ext2/inode.c" 1573 [] --55%--                    866,1          54%
```

<fs/ext2/inode.c>

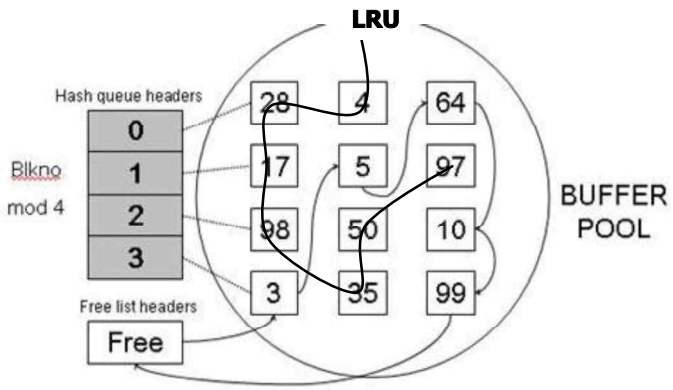
File read and write

- Control flow

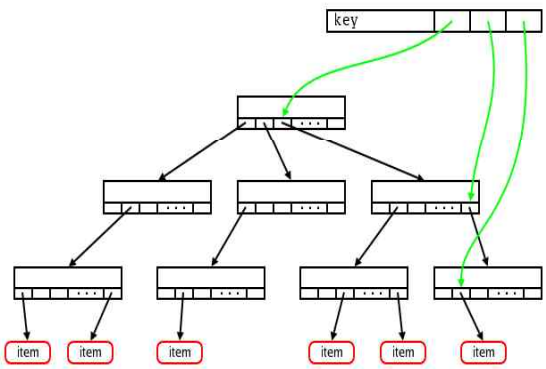


File read and write

- Page cache and Buffer cache
  - ✓ kernel features two general caching options for block devices
    - Buffer cache
      - Traditional, sector basis, managed by LRU and hash, currently used for small read operations
    - Page cache
      - Commonly used in recent Linux, page basis, managed by radix tree, employ the address_space object which is associated with a file
  - ✓ In many scenarios, page and buffer caches are used in combination
    - a cached page is divided into various buffers during write operations so that the modified parts of the page can be more finely grained



(LRU with hash)



(Radix tree)

File read and write

- Key data structure 2: address_space and its operation
  - ✓ address_space: link btw an object and its cached data

```
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/include/linux

struct backing_dev_info;
struct address_space {
        struct inode            *host;          /* owner: inode, block_device */
        struct radix_tree_root  page_tree;      /* radix tree of all pages */
        spinlock_t              tree_lock;      /* and lock protecting it */
        atomic_t                i_mmap_writable;/* count VM_SHARED mappings */
        struct rb_root          i_mmap;         /* tree of private and shared ma
ppings */
        struct list_head        i_mmap_nonlinear;/*list VM_NONLINEAR mappings */
        struct rw_semaphore     i_mmap_rwsem;   /* protect tree, count, list */
        /* Protected by tree_lock together with the radix tree */
        unsigned long           nrpages;        /* number of total pages */
        unsigned long           nrshadows;      /* number of shadow entries */
        pgoff_t                 writeback_index;/* writeback starts here */
        const struct address_space_operations *a_ops;   /* methods */
        unsigned long           flags;          /* error bits/gfp mask */
        struct backing_dev_info *backing_dev_info; /* device readahead, etc */
        spinlock_t              private_lock;   /* for use by the address_space
*/
        struct list_head        private_list;   /* ditto */
        void                    *private_data;  /* ditto */
} __attribute__((aligned(sizeof(long))));
        /*
         * On most architectures that alignment is already the case; but
         * must be enforced here for CRIS, to let the least significant bit
         * of struct page's "mapping" pointer be used for PAGE_MAPPING_ANON.
         */
struct request_queue;

struct block_device {
        dev_t                   bd_dev;  /* not a kdev_t - it's a search key */
        int                     bd_openers;
        struct inode *          bd_inode;       /* will die */
        struct super_block *    bd_super;
        struct mutex            bd_mutex;       /* open/close mutex */
        struct list_head        bd_inodes;
        void *                  bd_claiming;
        void *                  bd_holder;
        int                     bd_holders;
        bool                    bd_write_holder;
#ifdef CONFIG_SYSFS
        struct list_head        bd_holder_disks;
#endif
        struct block_device *   bd_contains;
```

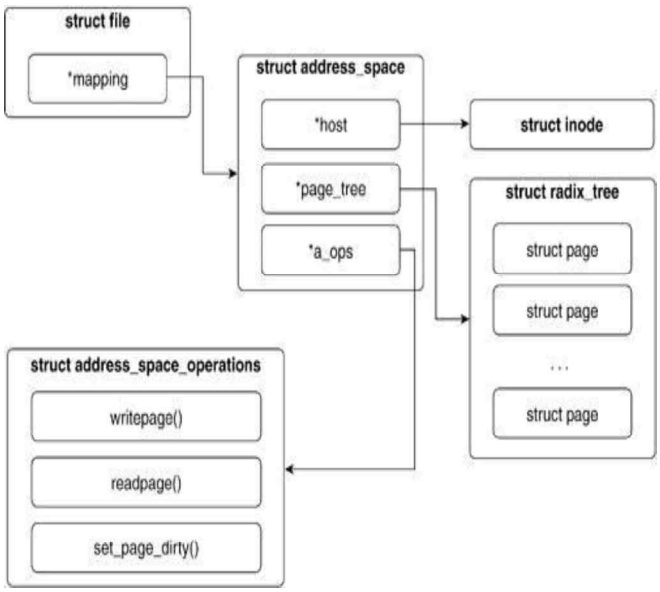
<include/fs.h>

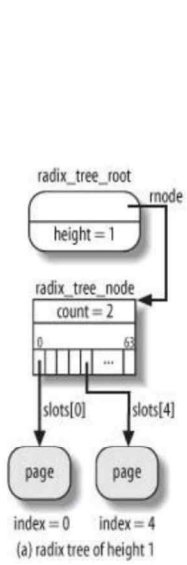
```
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/include/linux

struct address_space_operations {
        int (*writepage)(struct page *page, struct writeback_control *wbc);
        int (*readpage)(struct file *, struct page *);

        /* Write back some dirty pages from this mapping. */
        int (*writepages)(struct address_space *, struct writeback_control *);

        /* Set a page dirty.  Return true if this dirtied it */
        int (*set_page_dirty)(struct page *page);

        int (*readpages)(struct file *filp, struct address_space *mapping,
                         struct list_head *pages, unsigned nr_pages);

        int (*write_begin)(struct file *, struct address_space *mapping,
                           loff_t pos, unsigned len, unsigned flags,
                           struct page **pagep, void **fsdata);
        int (*write_end)(struct file *, struct address_space *mapping,
                         loff_t pos, unsigned len, unsigned copied,
                         struct page *page, void *fsdata);

        /* Unfortunately this kludge is needed for FIBMAP. Don't use it */
        sector_t (*bmap)(struct address_space *, sector_t);
        void (*invalidatepage) (struct page *, unsigned int, unsigned int);
        int (*releasepage) (struct page *, gfp_t);
        void (*freepage)(struct page *);
        ssize_t (*direct_IO)(int, struct kiocb *, struct iov_iter *iter, loff_t
offset);
        int (*get_xip_mem)(struct address_space *, pgoff_t, int,
                           void **, unsigned long *);
        /*
         * migrate the contents of a page to the specified target. If
         * migrate_mode is MIGRATE_ASYNC, it must not block.
         */
        int (*migratepage) (struct address_space *,
                            struct page *, struct page *, enum migrate_mode);
        int (*launder_page) (struct page *);
        int (*is_partially_uptodate) (struct page *, unsigned long,
                                      unsigned long);
        void (*is_dirty_writeback) (struct page *, bool *, bool *);
        int (*error_remove_page)(struct address_space *, struct page *);

        /* swapfile support */
        int (*swap_activate)(struct swap_info_struct *sis, struct file *file,
                             sector_t *span);
        void (*swap_deactivate)(struct file *file);
};
                                                        337,7          11%
```

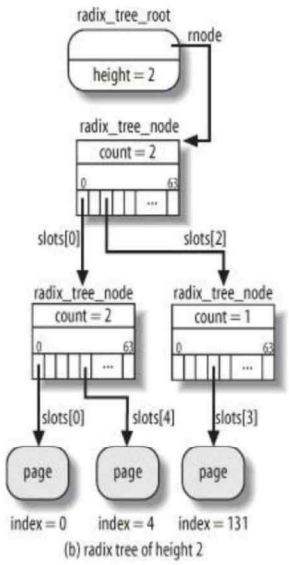
<include/fs.h>

File read and write

- Relation between address_space and radix tree







(Source: R. Love, :"Linux Kernel Development")

File read and write

- Code example: page cache

choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/include/linux

```
 *
 * Otherwise, %NULL is returned.
 */
static inline struct page *find_get_page(struct address_space *mapping,
                                         pgoff_t offset)
{
        return pagecache_get_page(mapping, offset, 0, 0);
}

static inline struct page *find_get_page_flags(struct address_space *mapp
ing,
                                               pgoff_t offset, int fgp_flags)
{
        return pagecache_get_page(mapping, offset, fgp_flags, 0);
}

/**
 * find_lock_page - locate, pin and lock a pagecache page
"pagemap.h" 673 [] --41%--                                  279,4          39%
```

<include/linux/pagemap.h>

choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/mm

```
 */
struct page *pagecache_get_page(struct address_space *mapping, pgoff_t of
fset,
        int fgp_flags, gfp_t gfp_mask)
{
        struct page *page;

repeat:
        page = find_get_entry(mapping, offset);
        if (radix_tree_exceptional_entry(page))
                page = NULL;
        if (!page)
                goto no_page;

        if (fgp_flags & FGP_LOCK) {
                if (fgp_flags & FGP_NOWAIT) {
                        if (!trylock_page(page)) {
                                page_cache_release(page);
                                                                1082,1-8       39%
```

<mm/filemap.c>

choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/mm

```
 * Otherwise, %NULL is returned.
 */
struct page *find_get_entry(struct address_space *mapping, pgoff_t offset
)
{
        void **pagep;
        struct page *page;

        rcu_read_lock();
repeat:
        page = NULL;
        pagep = radix_tree_lookup_slot(&mapping->page_tree, offset);
        if (pagep) {
                page = radix_tree_deref_slot(pagep);
                if (unlikely(!page))
                        goto out;
                if (radix_tree_exception(page)) {
                        if (radix_tree_deref_retry(page))
                                goto repeat;
                        /*
                         * A shadow entry of a recently evicted page.
"filemap.c" 2703 [] --35%--                                 966,1          35%
```

<mm/filemap.c>

choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/lib

```
/**
 *      __radix_tree_lookup     -       lookup an item in a radix tree
 *      @root:          radix tree root
 *      @index:         index key
 *      @nodep:         returns node
 *      @slotp:         returns slot
 *
 *      Lookup and return the item at position @index in the radix
 *      tree @root.
 *
 *      Until there is more than one item in the tree, no nodes are
 *      allocated and @root->rnode is used as a direct slot instead of
 *      pointing to a node, in which case *@nodep will be NULL.
 */
void *__radix_tree_lookup(struct radix_tree_root *root, unsigned long ind
ex,
                          struct radix_tree_node **nodep, void ***slotp)
{
        struct radix_tree_node *node, *parent;
        unsigned int height, shift;
        void **slot;

        node = rcu_dereference_raw(root->rnode);
        if (node == NULL)
                return NULL;

        if (!radix_tree_is_indirect_ptr(node)) {
                if (index > 0)
                        return NULL;

                if (nodep)
                        *nodep = NULL;
                if (slotp)
                        *slotp = (void **)&root->rnode;
"radix-tree.c" 1488 [] --32%--                              484,14-19      32%
```

<lib/radix-tree.c>

File read and write

- Code example: Specific FS

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
const struct address_space_operations ext2_aops = {
        .readpage               = ext2_readpage,
        .readpages              = ext2_readpages,
        .writepage              = ext2_writepage,
        .write_begin            = ext2_write_begin,
        .write_end              = ext2_write_end,
        .bmap                   = ext2_bmap,
        .direct_IO              = ext2_direct_IO,
        .writepages             = ext2_writepages,
        .migratepage            = buffer_migrate_page,
        .is_partially_uptodate  = block_is_partially_uptodate,
        .error_remove_page      = generic_error_remove_page,
};

const struct address_space_operations ext2_aops_xip = {
        .bmap                   = ext2_bmap,
        .get_xip_mem            = ext2_get_xip_mem,
};

const struct address_space_operations ext2_nobh_aops = {
                                                                873,0-1        5
```

<fs/ext2/inode.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
int fat_block_truncate_page(struct inode *inode, loff_t from)
{
        return block_truncate_page(inode->i_mapping, from, fat_get_block);
}

static const struct address_space_operations fat_aops = {
        .readpage       = fat_readpage,
        .readpages      = fat_readpages,
        .writepage      = fat_writepage,
        .writepages     = fat_writepages,
        .write_begin    = fat_write_begin,
        .write_end      = fat_write_end,
        .direct_IO      = fat_direct_IO,
        .bmap           = _fat_bmap
};

/*
 * New FAT inode stuff. We do the following:
 *      a) i_ino is constant and has nothing with on-disk location.
 *      b) FAT manages its own cache of directory entries.
 *      c) *This* cache is indexed by on-disk location.
```

<fs/fat/inode.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs/f2fs

```
        if (f2fs_has_inline_data(inode)) {
                int err = f2fs_convert_inline_inode(inode);
                if (err)
                        return err;
        }
        return generic_block_bmap(mapping, block, get_data_block);
}

const struct address_space_operations f2fs_dblock_aops = {
        .readpage       = f2fs_read_data_page,
        .readpages      = f2fs_read_data_pages,
        .writepage      = f2fs_write_data_page,
        .writepages     = f2fs_write_data_pages,
        .write_begin    = f2fs_write_begin,
        .write_end      = f2fs_write_end,
        .set_page_dirty = f2fs_set_data_page_dirty,
        .invalidatepage = f2fs_invalidate_data_page,
        .releasepage    = f2fs_release_data_page,
        .direct_IO      = f2fs_direct_IO,
        .bmap           = f2fs_bmap,
}
                                                                1175,2
```

<fs/f2fs/data.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs/nfs

```
const struct address_space_operations nfs_file_aops = {
        .readpage = nfs_readpage,
        .readpages = nfs_readpages,
        .set_page_dirty = __set_page_dirty_nobuffers,
        .writepage = nfs_writepage,
        .writepages = nfs_writepages,
        .write_begin = nfs_write_begin,
        .write_end = nfs_write_end,
        .invalidatepage = nfs_invalidate_page,
        .releasepage = nfs_release_page,
        .direct_IO = nfs_direct_IO,
        .migratepage = nfs_migrate_page,
        .launder_page = nfs_launder_page,
        .is_dirty_writeback = nfs_check_dirty_writeback,
        .error_remove_page = generic_error_remove_page,
#ifdef CONFIG_NFS_SWAP
        .swap_activate = nfs_swap_activate,
        .swap_deactivate = nfs_swap_deactivate,
#endif
}
                                                                599,2          6
```

<fs/nfs/file.c>

File read and write

- Code example: Specific FS and cache
  - ✓ mpage: preparing and submitting BIOs which contain multiple pagecache pages

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs/ext2

```
int ext2_fiemap(struct inode *inode, struct fiemap_extent_info *fieinfo,
                u64 start, u64 len)
{
        return generic_block_fiemap(inode, fieinfo, start, len,
                                    ext2_get_block);
}

static int ext2_writepage(struct page *page, struct writeback_control *wbc)
{
        return block_write_full_page(page, ext2_get_block, wbc);
}

static int ext2_readpage(struct file *file, struct page *page)
{
        return mpage_readpage(page, ext2_get_block);
}

static int
ext2_readpages(struct file *file, struct address_space *mapping,
               struct list_head *pages, unsigned nr_pages)
"inode.c" 1573 [] --49%--                                   777,16          49%
```

<fs/ext2/inode.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
int mpage_readpage(struct page *page, get_block_t get_block)
{
        struct bio *bio = NULL;
        sector_t last_block_in_bio = 0;
        struct buffer_head map_bh;
        unsigned long first_logical_block = 0;

        map_bh.b_state = 0;
        map_bh.b_size = 0;
        bio = do_mpage_readpage(bio, page, 1, &last_block_in_bio,
                                &map_bh, &first_logical_block, get_block);
        if (bio)
                mpage_bio_submit(READ, bio);
        return 0;
}
EXPORT_SYMBOL(mpage_readpage);

/*
 * Writing is not so simple.
 *
 * If the page has buffers then they will be used for obtaining the disk
 * mapping.  We only support pages which are fully mapped-and-dirty, with a
 * special case for pages which are unmapped at the end: end-of-file.
 *
```

<fs/mpage.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
/*
 * This is the worker routine which does all the work of mapping the disk
 * blocks and constructs largest possible bios, submits them for IO if the
 * blocks are not contiguous on the disk.
 *
 * We pass a buffer_head back and forth and use its buffer_mapped() flag to
 * represent the validity of its disk mapping and to decide when to do the next
 * get_block() call.
 */
static struct bio *
do_mpage_readpage(struct bio *bio, struct page *page, unsigned nr_pages,
                sector_t *last_block_in_bio, struct buffer_head *map_bh,
                unsigned long *first_logical_block, get_block_t get_block)
{
        ...
        /* Map blocks using the result from the previous get_blocks call.  */
        nblocks = map_bh->b_size >> blkbits;
        if (buffer_mapped(map_bh) && block_in_file > *first_logical_block &&
                        block_in_file < (*first_logical_block + nblocks)) {
                ...
                bdev = map_bh->b_bdev;
        }

        /* Then do more get_blocks calls until we are done with this page.  */
        map_bh->b_page = page;
        while (page_block < blocks_per_page) {
                get_block(inode, block_in_file, map_bh, 0);
        }
alloc_new:
        if (bio == NULL) {
                bio = mpage_alloc(bdev, blocks[0] << (blkbits - 9), min_t(int, n
r_pages, bio_get_nr_vecs(bdev)), GFP_KERNEL);
        }
out:
        return bio;
```

< fs/mpage.c >

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
static struct bio *mpage_bio_submit(int rw, struct bio *bio)
{
        bio->bi_end_io = mpage_end_io;
        guard_bio_eod(rw, bio);
        submit_bio(rw, bio);
        return NULL;
}

static struct bio *
mpage_alloc(struct block_device *bdev,
                sector_t first_sector, int nr_vecs,
                gfp_t gfp_flags)
{
        struct bio *bio;

        bio = bio_alloc(gfp_flags, nr_vecs);

        if (bio == NULL && (current->flags & PF_MEMALLOC)) {
                while (!bio && (nr_vecs /= 2))
                        bio = bio_alloc(gfp_flags, nr_vecs);
        }

        if (bio) {
                bio->bi_bdev = bdev;
                bio->bi_iter.bi_sector = first_sector;
        }
        return bio;
}
```

< fs/mpage.c >

File read and write

- Code example: Driver

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/block

```
/*
 * submit_bio - submit a bio to the block device layer for I/O
 * @rw: whether to %READ or %WRITE, or maybe to %READA (read ahead)
 * @bio: The &struct bio which describes the I/O
 *
 * submit_bio() is very similar in purpose to generic_make_request(), and
 * uses that function to do most of the work. Both are fairly rough
 * interfaces; @bio must be presetup and ready for I/O.
 *
 */
void submit_bio(int rw, struct bio *bio)
{
        bio->bi_rw |= rw;

        if (bio_has_data(bio)) {
                if (rw & WRITE) {
                        count_vm_events(PGPGOUT, count);
                } else {
                        task_io_account_read(bio->bi_iter.bi_size);
                        count_vm_events(PGPGIN, count);
                }
                ...
        }
        generic_make_request(bio);
}
```

<block/blk-core.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/block

```
 */
void generic_make_request(struct bio *bio)
{
        struct bio_list bio_list_on_stack;

        if (!generic_make_request_checks(bio))
                return;

        if (current->bio_list) {
                bio_list_add(current->bio_list, bio);
                return;
        }

        bio_list_init(&bio_list_on_stack);
        current->bio_list = &bio_list_on_stack;
        do {
                struct request_queue *q = bdev_get_queue(bio->bi_bdev);

                q->make_request_fn(q, bio);

                bio = bio_list_pop(current->bio_list);
        } while (bio);
        current->bio_list = NULL; /* deactivate */
}
```

< block/blk-core.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/include/linux

```
 * Try to put the fields that are referenced together in the same cacheline.
 *
 * If you modify this structure, make sure to update blk_rq_init() and
 * especially blk_mq_rq_ctx_init() to take care of the added fields.
 */
struct request {
        struct list_head queuelist;
        union {
                struct call_single_data csd;
                unsigned long fifo_time;
        };

        struct request_queue *q;
        struct blk_mq_ctx *mq_ctx;

        u64 cmd_flags;
        enum rq_cmd_type_bits cmd_type;
        unsigned long atomic_flags;

        int cpu;

        /* the following two fields are internal, NEVER access directly */
        unsigned int __data_len;        /* total data len */
        sector_t __sector;              /* sector cursor */

        struct bio *bio;
        struct bio *biotail;
```

< include/linux/blkdev.h>

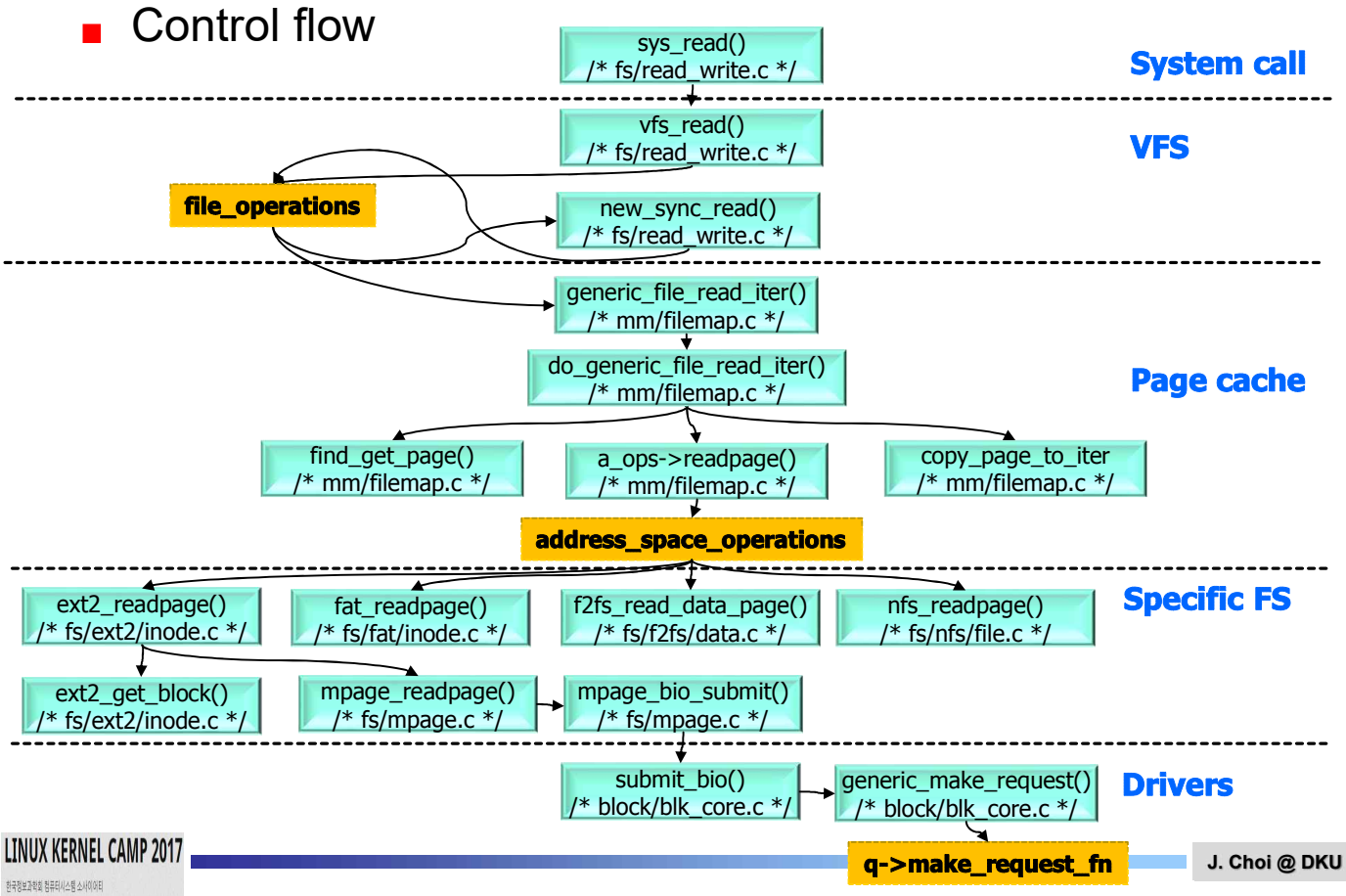
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/drivers

```
        return IRQ_HANDLED;
}

static const struct block_device_operations hd_fops = {
        .getgeo =       hd_getgeo,
};

static int __init hd_init(void)
{
        int drive;

        if (register_blkdev(HD_MAJOR, "hd"))
                return -1;

        hd_queue = blk_init_queue(do_hd_request, &hd_lock);
        if (!hd_queue) {
                unregister_blkdev(HD_MAJOR, "hd");
                return -ENOMEM;
        }

        blk_queue_max_hw_sectors(hd_queue, 255);
        init_timer(&device_timer);
        device_timer.function = hd_times_out;
        blk_queue_logical_block_size(hd_queue, 512);

        if (!NR_HD) {
                /*
                 * We don't know anything about the drive.  This means
```

< drivers/block/hd.c >

File read and write

- Control flow



File read and write

- Logical view
  - ✓ File system and block device driver

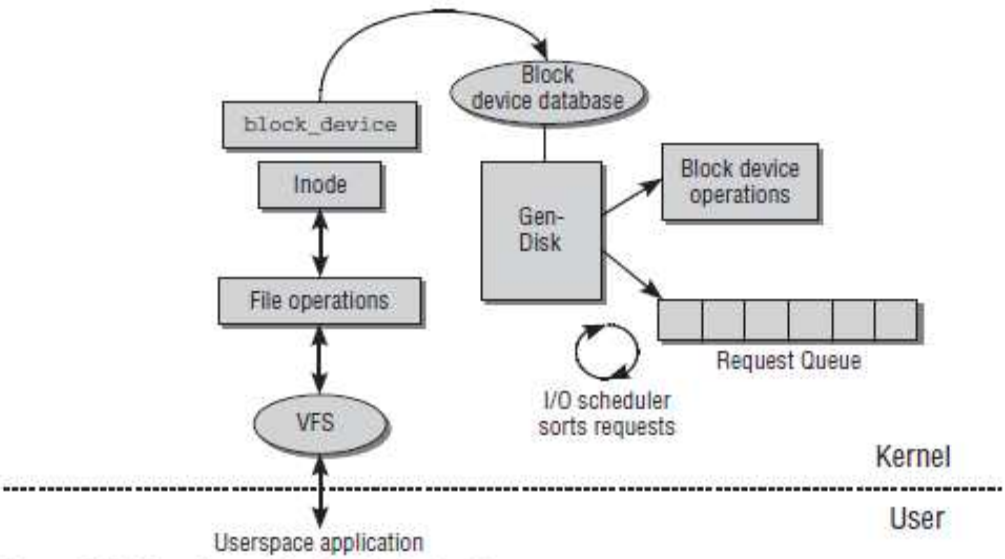


Figure 6-9: Overview of the block device layer.

(Source: Professional Linux Kernel Architecture, Wrox)

File system initialization

- Code examples: fs_init

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs/ext2

```
static int __init init_ext2_fs(void)
{
        int err = init_ext2_xattr();
        err = init_inodecache();
        err = register_filesystem(&ext2_fs_type);
        if (err)
                goto out;
        return 0;
}

static void __exit exit_ext2_fs(void)
{
        unregister_filesystem(&ext2_fs_type);
        destroy_inodecache();
        exit_ext2_xattr();
}

MODULE_AUTHOR("Remy Card and others");
MODULE_DESCRIPTION("Second Extended Filesystem");
MODULE_LICENSE("GPL");
module_init(init_ext2_fs)
module_exit(exit_ext2_fs)
```

<fs/ext2/super.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs/ext2

```
        inode->i_version++;
        inode->i_mtime = inode->i_ctime = CURRENT_TIME;
        mark_inode_dirty(inode);
        return len - towrite;
}

#endif

static struct file_system_type ext2_fs_type = {
        .owner          = THIS_MODULE,
        .name           = "ext2",
        .mount          = ext2_mount,
        .kill_sb        = kill_block_super,
        .fs_flags       = FS_REQUIRES_DEV,
};
MODULE_ALIAS_FS("ext2");

static int __init init_ext2_fs(void)
{
        int err = init_ext2_xattr();
        err = init_inodecache();
        err = register_filesystem(&ext2_fs_type);
        if (err)
                goto out;
```

<fs/ext2/super.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/include/linux

```
int sync_inode(struct inode *inode, struct writeback_control *wbc);
int sync_inode_metadata(struct inode *inode, int wait);

struct file_system_type {
        const char *name;
        int fs_flags;
#define FS_REQUIRES_DEV         1
#define FS_BINARY_MOUNTDATA     2
#define FS_HAS_SUBTYPE          4
#define FS_USERNS_MOUNT         8       /* Can be mounted by userns root */
#define FS_USERNS_DEV_MOUNT     16 /* A userns mount does not imply MNT_NODEV */
#define FS_RENAME_DOES_D_MOVE   32768   /* FS will handle d_move() during rename
() internally. */
        struct dentry *(*mount) (struct file_system_type *, int,
                        const char *, void *);
        void (*kill_sb) (struct super_block *);
        struct module *owner;
        struct file_system_type * next;
        struct hlist_head fs_supers;

        struct lock_class_key s_lock_key;
        struct lock_class_key s_umount_key;
        struct lock_class_key s_vfs_rename_key;
        struct lock_class_key s_writers_key[SB_FREEZE_LEVELS];
```

<include/linux/fs.h>

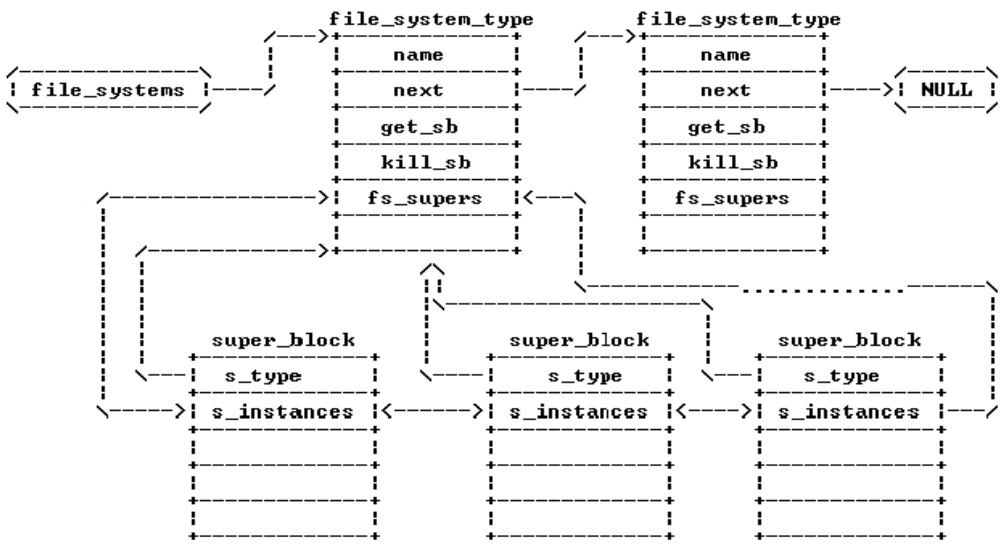
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
 *      The &struct file_system_type that is passed is linked into the kernel
 *      structures and must not be freed until the file system has been
 *      unregistered.
 */

int register_filesystem(struct file_system_type * fs)
{
        int res = 0;
        struct file_system_type ** p;

        BUG_ON(strchr(fs->name, '.'));
        if (fs->next)
                return -EBUSY;
        write_lock(&file_systems_lock);
        p = find_filesystem(fs->name, strlen(fs->name));
        if (*p)
                res = -EBUSY;
        else
                *p = fs;
        write_unlock(&file_systems_lock);
        return res;
}
```

<fs/filesystems.c>

File system initialization

- Logical view



(Source: http://asfadsfe7477.fotopages.com/?entry=3773395)

File system mount

- Code examples: mount

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
SYSCALL_DEFINE5(mount, char __user *, dev_name, char __user *, dir_name,
                char __user *, type, unsigned long, flags, void __user *, data)
{
        int ret;

        kernel_type = copy_mount_string(type);
        ret = PTR_ERR(kernel_type);
        if (IS_ERR(kernel_type))
                goto out_type;

        kernel_dev = copy_mount_string(dev_name);
        ret = PTR_ERR(kernel_dev);
        if (IS_ERR(kernel_dev))
                goto out_dev;

        ret = copy_mount_options(data, &data_page);
        if (ret < 0)
                goto out_data;

        ret = do_mount(kernel_dev, dir_name, kernel_type, flags,
                       (void *) data_page);

        free_page(data_page);
out_data:
                                                        2811,0-1      88%
```

<fs/namespace.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
*/
long do_mount(const char *dev_name, const char __user *dir_name,
              const char *type_page, unsigned long flags, void *data_page)
{
        ...
        retval = user_path(dir_name, &path);

        retval = security_sb_mount(dev_name, &path,
                                   type_page, flags, data_page);

        ...
        if (flags & MS_REMOUNT)
                retval = do_remount(&path, flags & ~MS_REMOUNT, mnt_flags,
                                    data_page);
        else if (flags & MS_BIND)
                retval = do_loopback(&path, dev_name, flags & MS_REC);
        else if (flags & (MS_SHARED | MS_PRIVATE | MS_SLAVE | MS_UNBINDABLE))
                retval = do_change_type(&path, flags);
        else if (flags & MS_MOVE)
                retval = do_move_mount(&path, dev_name);
        else
                retval = do_new_mount(&path, type_page, flags, mnt_flags,
                                      dev_name, data_page);
dput_out:
        path_put(&path);
```

<fs/namespace.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
static int do_new_mount(struct path *path, const char *fstype, int flags,
                        int mnt_flags, const char *name, void *data)
{
        struct user_namespace *user_ns = current->nsproxy->mnt_ns->user_ns;
        struct vfsmount *mnt;
        int err;

        type = get_fs_type(fstype);
        if (!type)
                return -ENODEV;

        ...

        mnt = vfs_kern_mount(type, flags, name, data);
        if (!IS_ERR(mnt) && (type->fs_flags & FS_HAS_SUBTYPE) &&
            !mnt->mnt_sb->s_subtype)
                mnt = fs_set_subtype(mnt, fstype);

        put_filesystem(type);

        err = do_add_mount(real_mount(mnt), path, mnt_flags);
        if (err)
                mntput(mnt);
        return err;
```

<fs/namespace.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
static struct file_system_type *__get_fs_type(const char *name, int len)
{
        struct file_system_type *fs;

        read_lock(&file_systems_lock);
        fs = *(find_filesystem(name, len));
        if (fs && !try_module_get(fs->owner))
                fs = NULL;
        read_unlock(&file_systems_lock);
        return fs;
}

struct file_system_type *get_fs_type(const char *name)
{
        struct file_system_type *fs;
        const char *dot = strchr(name, '.');
        int len = dot ? dot - name : strlen(name);

        fs = __get_fs_type(name, len);
        if (!fs && (request_module("fs-%.*s", len, name) == 0))
                fs = __get_fs_type(name, len);

        return fs;
}
```

<fs/filesystems.c>

File system mount

- Code examples

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
struct vfsmount *
vfs_kern_mount(struct file_system_type *type, int flags, const char *name, void
*data)
{
        struct mount *mnt;
        struct dentry *root;

        if (!type)
                return ERR_PTR(-ENODEV);

        mnt = alloc_vfsmnt(name);
        if (!mnt)
                return ERR_PTR(-ENOMEM);

        if (flags & MS_KERNMOUNT)
                mnt->mnt.mnt_flags = MNT_INTERNAL;

        root = mount_fs(type, flags, name, data);
        if (IS_ERR(root)) {
                mnt_free_id(mnt);
                free_vfsmnt(mnt);
                return ERR_CAST(root);
        }
```

<fs/namespace.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
EXPORT_SYMBOL(mount_single);

struct dentry *
mount_fs(struct file_system_type *type, int flags, const char *name, void *data)
{
        struct dentry *root;
        struct super_block *sb;
        char *secdata = NULL;
        int error = -ENOMEM;

        if (data && !(type->fs_flags & FS_BINARY_MOUNTDATA)) {
                secdata = alloc_secdata();
                if (!secdata)
                        goto out;

                error = security_sb_copy_data(data, secdata);
                if (error)
                        goto out_free_secdata;
        }

        root = type->mount(type, flags, name, data);
        if (IS_ERR(root)) {
                error = PTR_ERR(root);
                goto out_free_secdata;
```

<fs/super.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs/ext2

```
                le64_to_cpup((void *)es->s_uuid + sizeof(u64));
        buf->f_fsid.val[0] = fsid & 0xFFFFFFFFUL;
        buf->f_fsid.val[1] = (fsid >> 32) & 0xFFFFFFFFUL;
        spin_unlock(&sbi->s_lock);
        return 0;
}

static struct dentry *ext2_mount(struct file_system_type *fs_type,
        int flags, const char *dev_name, void *data)
{
        return mount_bdev(fs_type, flags, dev_name, data, ext2_fill_super);
}

#ifdef CONFIG_QUOTA

/* Read data from quotafile - avoid pagecache and such because we cannot afford
 * acquiring the locks... As quota files are never truncated and quota code
 * itself serializes the operations (and no one else should touch the files)
 * we don't have to be afraid of races */
static ssize_t ext2_quota_read(struct super_block *sb, int type, char *data,
                               size_t len, loff_t off)
{
        struct inode *inode = sb_dqopt(sb)->files[type];
        sector_t blk = off >> EXT2_BLOCK_SIZE_BITS(sb);
```

<fs/ext2/super.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs

```
struct dentry *mount_bdev(struct file_system_type *fs_type,
        int flags, const char *dev_name, void *data,
        int (*fill_super)(struct super_block *, void *, int))
{
        ...
        s = sget(fs_type, test_bdev_super, set_bdev_super, flags | MS_NOSEC,
                 bdev);

        if (s->s_root) {
                up_write(&s->s_umount);
                blkdev_put(bdev, mode);
                down_write(&s->s_umount);
        } else {
                s->s_mode = mode;
                strlcpy(s->s_id, bdevname(bdev, b), sizeof(s->s_id));
                sb_set_blocksize(s, block_size(bdev));
                error = fill_super(s, data, flags & MS_SILENT ? 1 : 0);
                if (error) {
                        deactivate_locked_super(s);
                        goto error;
                }

                s->s_flags |= MS_ACTIVE;
```

<fs/super.c>

File system mount

- Key data structure 3: super block and its operations

choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/ext2

```
static int ext2_fill_super(struct super_block *sb, void *data, int silent
)
{
        struct buffer_head * bh;
        ...
        sb->s_fs_info = sbi;
        sb->s_magic = le16_to_cpu(es->s_magic);
        sbi->s_group_desc = kmalloc (db_count * sizeof (struct buffer_hea
d *), GFP_KERNEL);
        sb->s_op = &ext2_sops;
        ...
        root = ext2_iget(sb, EXT2_ROOT_INO);
        ...
        sb->s_root = d_make_root(root);
        return 0;

cantfind_ext2:
                                                        770,0-1       62%
```

<fs/ext2/super.c>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/include/linux

```
struct super_block {
        struct list_head        s_list;         /* Keep this first */
        dev_t                   s_dev;          /* search index; _not_ kdev_t */
        unsigned char           s_blocksize_bits;
        unsigned long           s_blocksize;
        loff_t                  s_maxbytes;     /* Max file size */
        struct file_system_type *s_type;
        const struct super_operations   *s_op;
        const struct dquot_operations   *dq_op;
        const struct quotactl_ops       *s_qcop;
        const struct export_operations *s_export_op;
        unsigned long           s_flags;
        unsigned long           s_magic;
        struct dentry           *s_root;
        struct rw_semaphore     s_umount;
        int                     s_count;
        atomic_t                s_active;
#ifdef CONFIG_SECURITY
        void                    *s_security;
#endif
        const struct xattr_handler **s_xattr;

        struct list_head        s_inodes;       /* all inodes */
                                                        1214,0-1      42%
```

<include/linux/fs.h>

choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/include/linux

```
                unsigned long, loff_t *);

struct super_operations {
        struct inode *(*alloc_inode)(struct super_block *sb);
        void (*destroy_inode)(struct inode *);

        void (*dirty_inode) (struct inode *, int flags);
        int (*write_inode) (struct inode *, struct writeback_control *wbc);
        int (*drop_inode) (struct inode *);
        void (*evict_inode) (struct inode *);
        void (*put_super) (struct super_block *);
        int (*sync_fs)(struct super_block *sb, int wait);
        int (*freeze_super) (struct super_block *);
        int (*freeze_fs) (struct super_block *);
        int (*thaw_super) (struct super_block *);
        int (*unfreeze_fs) (struct super_block *);
        int (*statfs) (struct dentry *, struct kstatfs *);
        int (*remount_fs) (struct super_block *, int *, char *);
        void (*umount_begin) (struct super_block *);

        int (*show_options)(struct seq_file *, struct dentry *);
        int (*show_devname)(struct seq_file *, struct dentry *);
        int (*show_path)(struct seq_file *, struct dentry *);
        int (*show_stats)(struct seq_file *, struct dentry *);
                                                        1593,8        56%
```

<include/linux/fs.h>

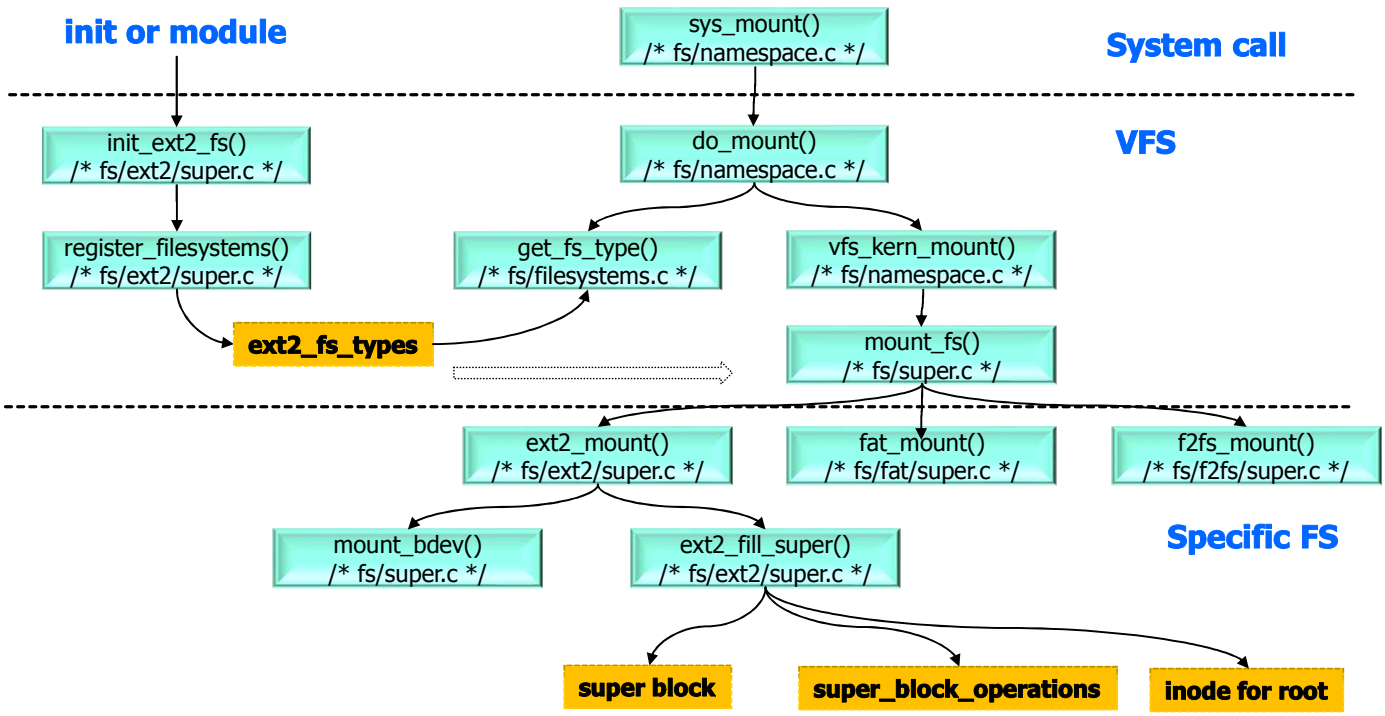
choijm@choijm-VirtualBox: ~/kernel_source/linux-3.19.6/fs/ext2

```
static const struct super_operations ext2_sops = {
        .alloc_inode    = ext2_alloc_inode,
        .destroy_inode  = ext2_destroy_inode,
        .write_inode    = ext2_write_inode,
        .evict_inode    = ext2_evict_inode,
        .put_super      = ext2_put_super,
        .sync_fs        = ext2_sync_fs,
        .freeze_fs      = ext2_freeze,
        .unfreeze_fs    = ext2_unfreeze,
        .statfs         = ext2_statfs,
        .remount_fs     = ext2_remount,
        .show_options   = ext2_show_options,
#ifdef CONFIG_QUOTA
        .quota_read     = ext2_quota_read,
        .quota_write    = ext2_quota_write,
        .get_dquots     = ext2_get_dquots,
#endif
};

static struct inode *ext2_nfs_get_inode(struct super_block *sb,
                u64 ino, u32 generation)
{
        struct inode *inode;
```

<fs/ext2/super.c>

File system mount

- Control flow



File system mount

- Revisit: code-level view
  - ✓ Code analysis: sys_mount

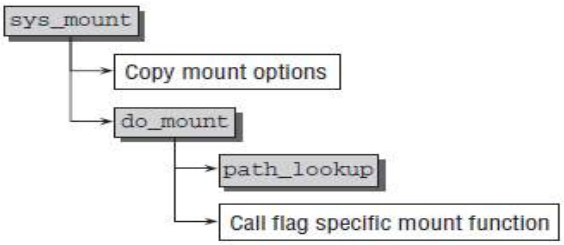


Figure 8-5: Code flow diagram for **sys_mount**.

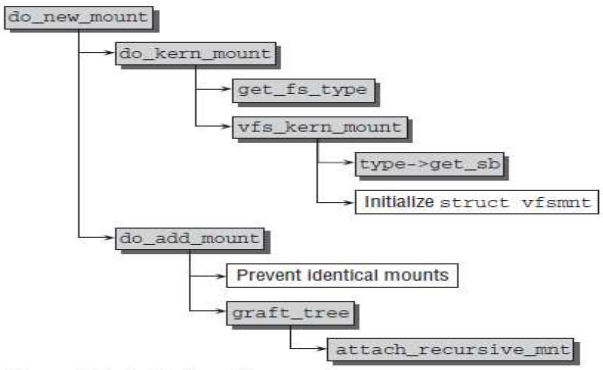


Figure 8-6: Code flow diagram for **do_new_mount**.

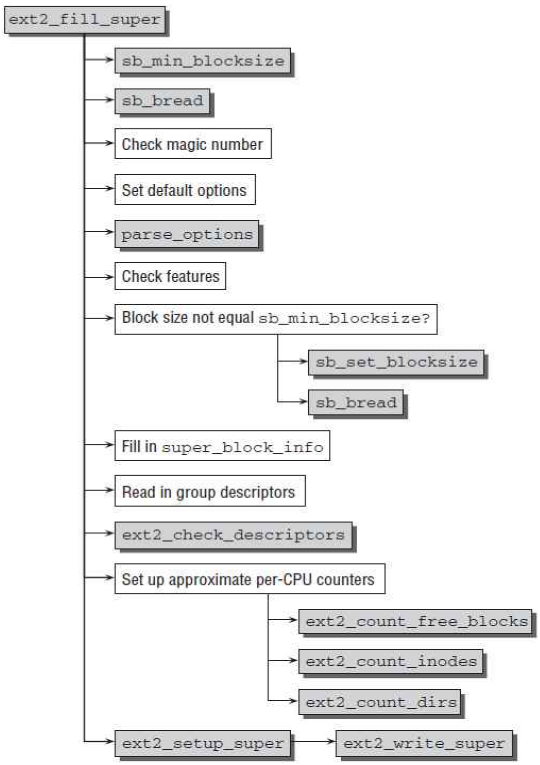
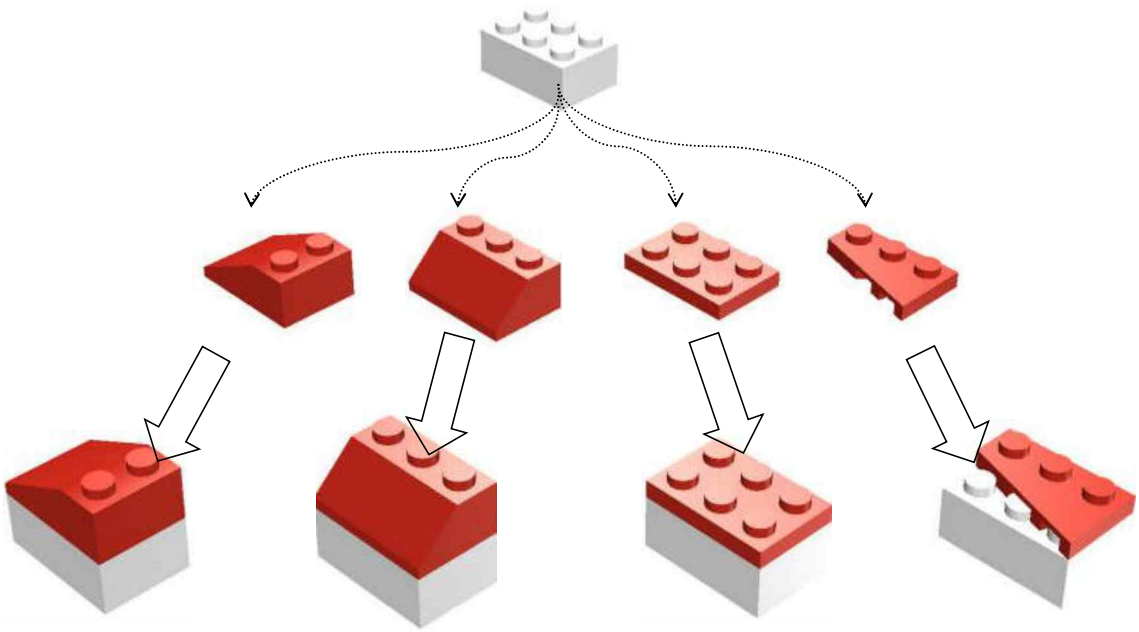


Figure 9-8: Code flow diagram for **ext2_fill_super**.

(Source: W. Mauerer, Professional Linux Kernel Architecture, Wrox)

How to make a new file system?

- Take out lessons
  - ✓ Layout: Relatively free
  - ✓ Interface: Need to satisfy the Linux kernel's expectation

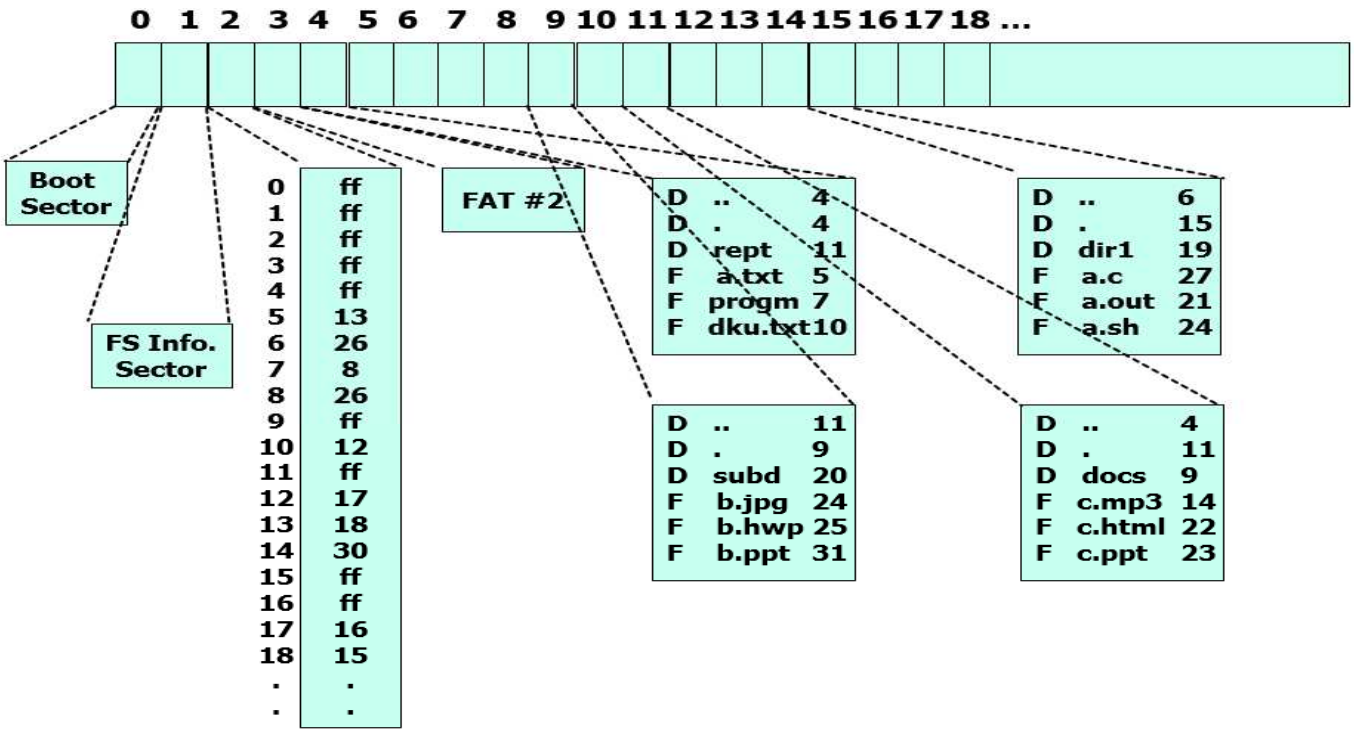


How to make a new file system?

- Interface summary
  - ✓ struct file_operations
    - General file interfaces
    - lseek(), read(), write(), read_iter(), write_iter(), open(), close(), ioctl(), …
  - ✓ struct inode_operations
    - Inode related interfaces
    - lookup(), link(), unlink(), mknod(), mkdir(), setattr(), …
  - ✓ struct address_space_operations
    - Used to map parts of files into pages in Linux
    - writepage(), readpage(), set_page_dirty(), bmap(), …
  - ✓ struct super_operations
    - File system management interface
    - alloc_inode(), write_inode(), sync_fs(), remount_fs(), …
  - ✓ struct file_system_type
    - Used for register_filesystem()
    - name, mount(), …

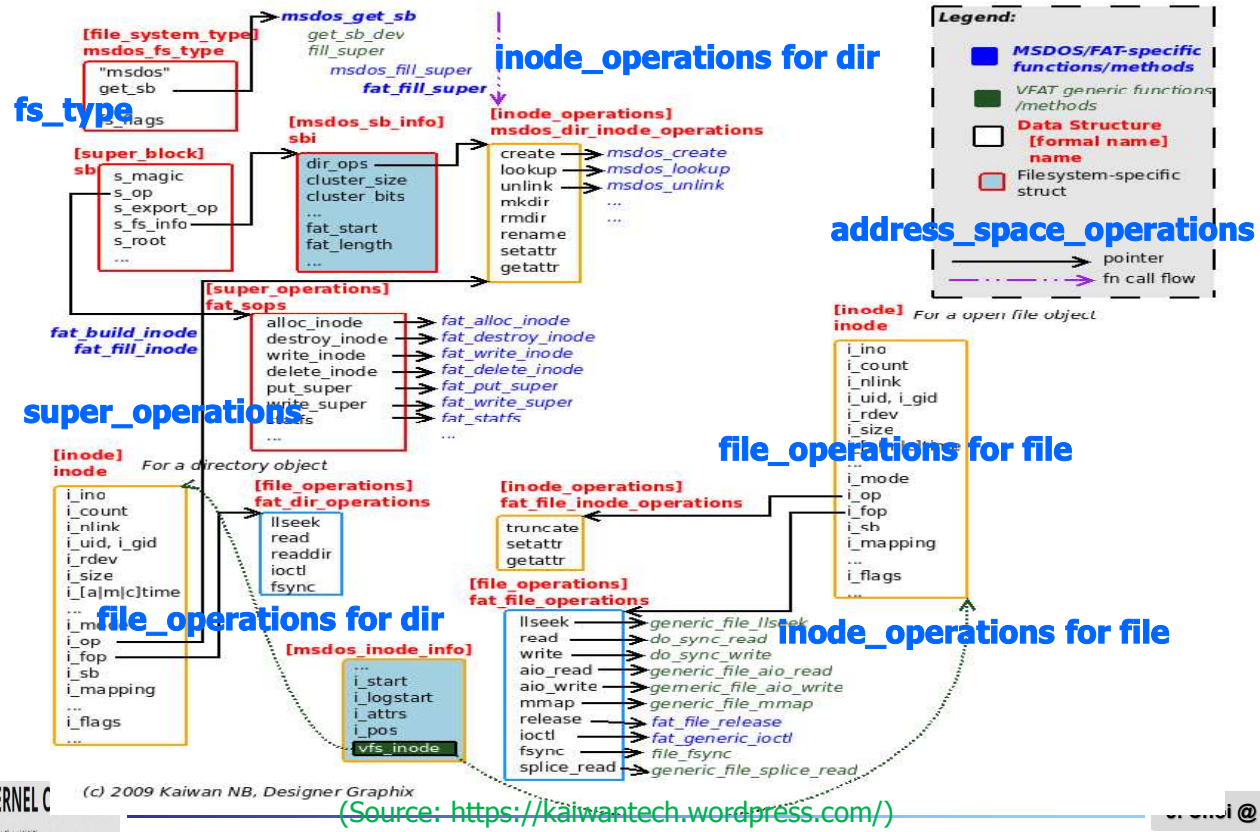
How to make a new file system?

- FAT layout



How to make a new file system?

- FAT interface: overall



(Source: https://kaiwantech.wordpress.com/)

How to make a new file system?

- F2FS layout

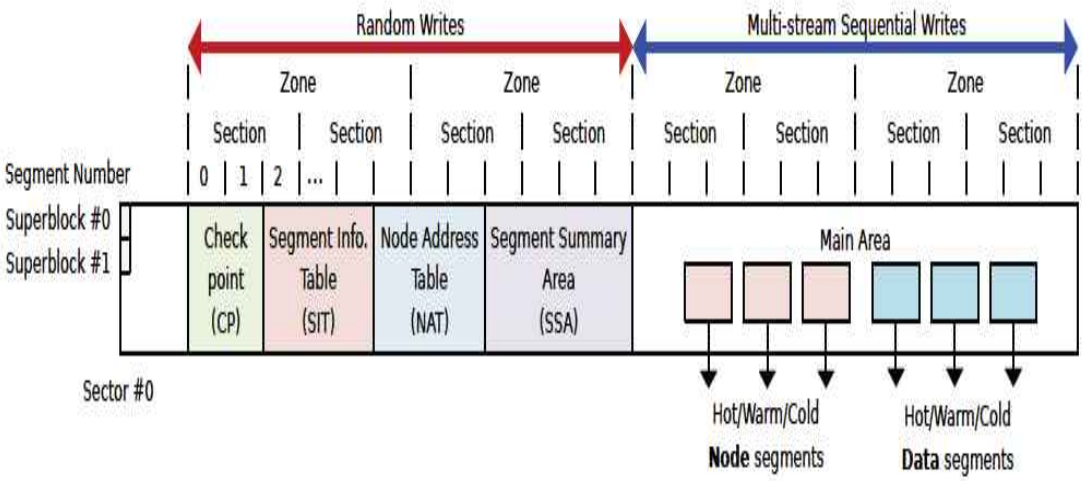


Figure 1: On-disk layout of F2FS.

How to make a new file system?

F2FS interface

```
const struct file_operations f2fs_file_operations = {
        .llseek         = f2fs_llseek,
        .read_iter      = generic_file_read_iter,
        .write_iter     = f2fs_file_write_iter,
        .open           = f2fs_file_open,
        .release        = f2fs_release_file,
        .mmap           = f2fs_file_mmap,
        .fsync          = f2fs_sync_file,
        .fallocate      = f2fs_fallocate,
        .unlocked_ioctl = f2fs_ioctl,
        .splice_read    = generic_file_splice_read,
        .splice_write   = iter_file_splice_write,};
/* fs/f2fs/file.c */
```

```
const struct inode_operations f2fs_file_inode_operations = {
        .getattr        = f2fs_getattr,
        .setattr        = f2fs_setattr,
        .get_acl        = f2fs_get_acl,
        .set_acl        = f2fs_set_acl,
        ...
        .fiemap         = f2fs_fiemap,
};   /* fs/f2fs/file.c */
```

```
const struct file_operations f2fs_dir_operations = {
        .llseek         = generic_file_llseek,
        .read           = generic_read_dir,
        .iterate        = f2fs_readdir,
        .fsync          = f2fs_sync_file,
        .unlocked_ioctl = f2fs_ioctl,
};   /* fs/f2fs/dir.c */
```

```
const struct inode_operations f2fs_dir_inode_operations = {
        .create         = f2fs_create,
        .lookup         = f2fs_lookup,
        .link           = f2fs_link,
        .unlink         = f2fs_unlink,
        .symlink        = f2fs_symlink,
        .mkdir          = f2fs_mkdir,
        .rmdir          = f2fs_rmdir,
        .mknod          = f2fs_mknod,
        .rename2        = f2fs_rename2,
        .tmpfile        = f2fs_tmpfile,
        .getattr        = f2fs_getattr,
        .setattr        = f2fs_setattr,
        .get_acl        = f2fs_get_acl,
        .set_acl        = f2fs_set_acl,
};   /* fs/f2fs/namei.c */
```


How to make a new file system?

F2FS interface

```
static struct super_operations f2fs_sops = {
 .alloc_inode    = f2fs_alloc_inode,
        .drop_inode     = f2fs_drop_inode,
        .destroy_inode  = f2fs_destroy_inode,
        .write_inode    = f2fs_write_inode,
        .dirty_inode    = f2fs_dirty_inode,
        .show_options   = f2fs_show_options,
        .evict_inode    = f2fs_evict_inode,
        .put_super      = f2fs_put_super,
        .sync_fs        = f2fs_sync_fs,
        .freeze_fs      = f2fs_freeze,
        .unfreeze_fs    = f2fs_unfreeze,
        .statfs         = f2fs_statfs,
        .remount_fs     = f2fs_remount,
};   /* fs/f2fs/super.c */
```

```
static struct file_system_type f2fs_fs_type = {
        .owner          = THIS_MODULE,
        .name           = "f2fs",
        .mount          = f2fs_mount,
        .kill_sb        = kill_f2fs_super,
        .fs_flags       = FS_REQUIRES_DEV,
}; /* fs/f2fs/super.c */
```

```
const struct address_space_operations f2fs_dblock_aops = {
        .readpage       = f2fs_read_data_page,
        .readpages      = f2fs_read_data_pages,
        .writepage      = f2fs_write_data_page,
        .writepages     = f2fs_write_data_pages,
        .write_begin    = f2fs_write_begin,
        .write_end      = f2fs_write_end,
        .set_page_dirty = f2fs_set_data_page_dirty,
        .invalidatepage = f2fs_invalidate_page,
        .releasepage    = f2fs_release_page,
        .direct_IO      = f2fs_direct_IO,
        .bmap           = f2fs_bmap,
};  /* fs/f2fs/data.c */
```

```
const struct address_space_operations f2fs_node_aops = {
        .writepage      = f2fs_write_node_page,
        .writepages     = f2fs_write_node_pages,
        .set_page_dirty = f2fs_set_node_page_dirty,
        .invalidatepage = f2fs_invalidate_page,
        .releasepage    = f2fs_release_page,
};/* fs/f2fs/node.c */
```

F2FS

- How to materialize? (revisit)

```
choijm@choijm-VirtualBox: /usr/src/linux-3.19.6/fs/f2fs
choijm@choijm-VirtualBox:/usr/src/linux-3.19.6/fs/f2fs$
choijm@choijm-VirtualBox:/usr/src/linux-3.19.6/fs/f2fs$ pwd
/usr/src/linux-3.19.6/fs/f2fs
choijm@choijm-VirtualBox:/usr/src/linux-3.19.6/fs/f2fs$ ls -l
합계 5232
-rw-rw-r-- 1 choijm choijm    2499  4월 29  2015 Kconfig
-rw-rw-r-- 1 choijm choijm     282  4월 29  2015 Makefile
-rw-rw-r-- 1 choijm choijm    9068  4월 29  2015 acl.c
-rw-rw-r-- 1 choijm choijm    1251  4월 29  2015 acl.h
-rw-rw-r-- 1 choijm choijm   27778  4월 29  2015 checkpoint.c
-rw-rw-r-- 1 choijm choijm   28256  4월 29  2015 data.c
-rw-rw-r-- 1 choijm choijm   11981  4월 29  2015 debug.c
-rw-rw-r-- 1 choijm choijm   19400  4월 29  2015 dir.c
-rw-rw-r-- 1 choijm choijm   50554  4월 29  2015 f2fs.h
-rw-r--r-- 1 root   root   4834153  5월  4  2015 f2fs.ko
-rw-r--r-- 1 root   root     14611  5월  4  2015 f2fs.mod.c
-rw-rw-r-- 1 choijm choijm   24632  4월 29  2015 file.c
-rw-rw-r-- 1 choijm choijm   18810  4월 29  2015 gc.c
-rw-rw-r-- 1 choijm choijm    3181  4월 29  2015 gc.h
-rw-rw-r-- 1 choijm choijm    2142  4월 29  2015 hash.c
-rw-rw-r-- 1 choijm choijm   12586  4월 29  2015 inline.c
-rw-rw-r-- 1 choijm choijm   10042  4월 29  2015 inode.c
-rw-r--r-- 1 root   root        23  5월  4  2015 modules.order
-rw-rw-r-- 1 choijm choijm   17560  4월 29  2015 namei.c
-rw-rw-r-- 1 choijm choijm   51226  4월 29  2015 node.c
-rw-rw-r-- 1 choijm choijm   10922  4월 29  2015 node.h
-rw-rw-r-- 1 choijm choijm   13705  4월 29  2015 recovery.c
-rw-rw-r-- 1 choijm choijm   59747  4월 29  2015 segment.c
-rw-rw-r-- 1 choijm choijm   22682  4월 29  2015 segment.h
-rw-rw-r-- 1 choijm choijm   32344  4월 29  2015 super.c
-rw-rw-r-- 1 choijm choijm   14988  4월 29  2015 xattr.c
-rw-rw-r-- 1 choijm choijm    4600  4월 29  2015 xattr.h
choijm@choijm-VirtualBox:/usr/src/linux-3.19.6/fs/f2fs$
```

For configuration

For build

Access control list

Make checkpoint

Data management
address_space_operations

Directory management
file_operations, inode_operations

f2fs main header

File interface, file_operations

Garbage collection management

inode management inode_operations

Directory traverse, dentry_operations

Node management

Rollback/forward recovery

Segment management

Super block management, super_operations, fs_type

Discussion

- Two key aspects for understanding file system
  - ✓ Layout
    - File system structure
    - Metadata format
  - ✓ Access method
    - open(), creat(), mount(), …
    - read(), write(), lseek(), stat(), …
- Importance of mental model

ASIDE: MENTAL MODELS OF FILE SYSTEMS

As we've discussed before, mental models are what you are really trying to develop when learning about systems. For file systems, your mental model should eventually include answers to questions like: what on-disk structures store the file system's data and metadata? What happens when a process opens a file? Which on-disk structures are accessed during a read or write? By working on and improving your mental model, you develop an abstract understanding of what is going on, instead of just trying to understand the specifics of some file-system code (though that is also useful, of course!).

(Source: Remzi, "Operating Systems: Three Easy Pieces")

Discussion

- Q&A

Appendix: File read and write

- Control flow (from Professional Linux Kernel Architecture)
  - ✓ do_generic_mapping_read() /* Standard functions */
    - A large loop to read all necessary pages
    - Find_get_page(): checks if the page is already contained in the page cache

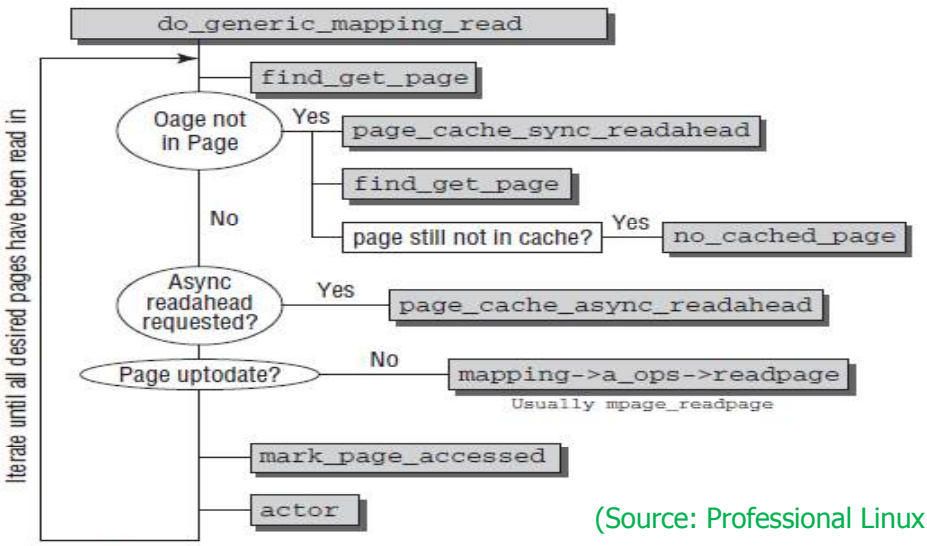


Figure 8-14: Code flow diagram for do_generic_mapping_read.

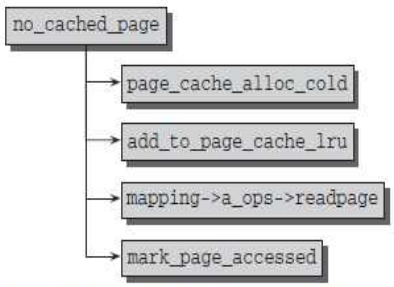


Figure 8-15: Code flow diagram for no_cached_page.

(Source: Professional Linux Kernel Architecture, Wrox)

Appendix: Timeline of journaling

- Data journaling vs. Metadata Journaling
  - ✓ Horizontal dashed line is “write barrier”
  - ✓ Note that, in this figure, the order btw Data and Journaling is not guaranteed in the metadata journaling timeline (writeback mode in the ext3.)

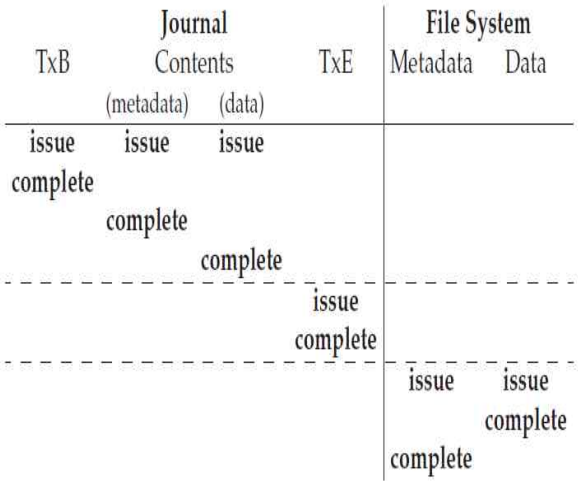


Figure 42.1: Data Journaling Timeline

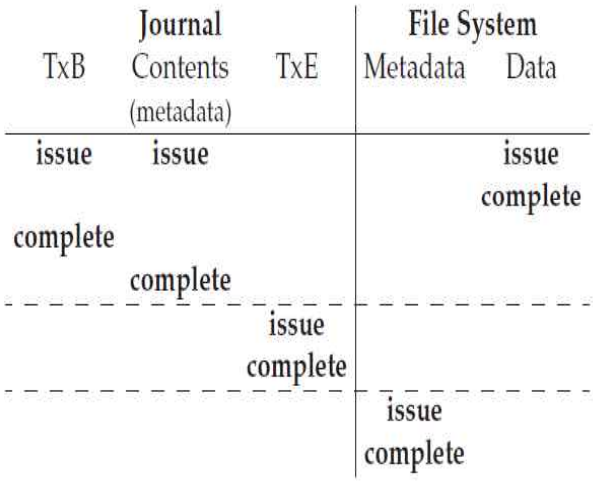


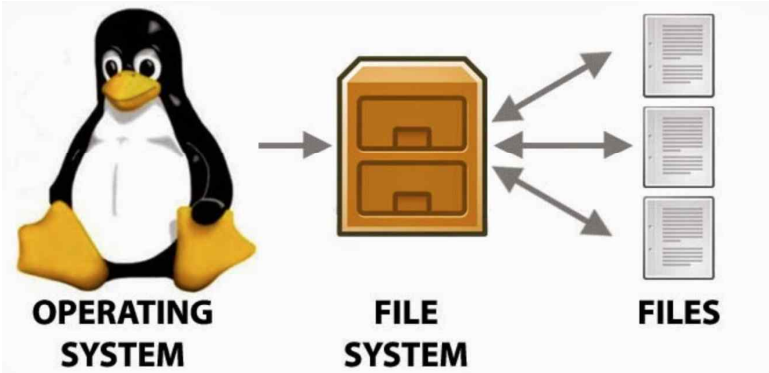
Figure 42.2: Metadata Journaling Timeline

Linux Kernel Camp 2017
- Part III: File system practice -

August, 23, 2017.
Dankook University
JuHyung Son

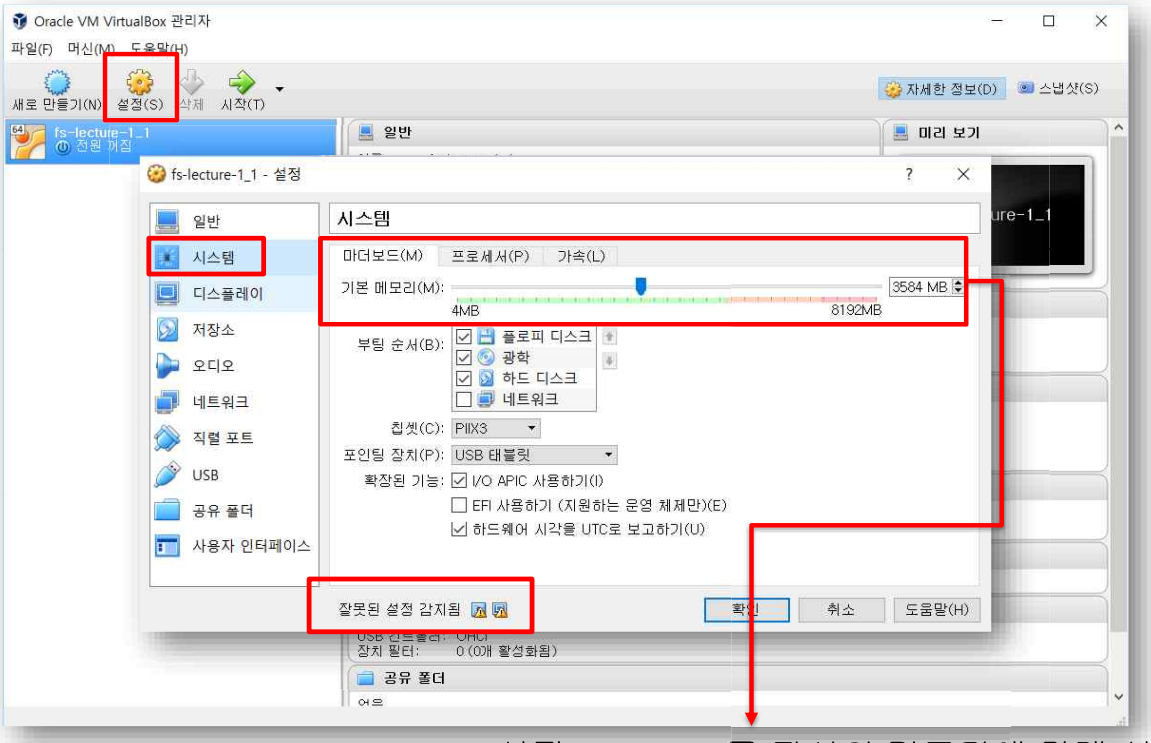
Contents

- FAT file system
  - ✓ FAT file system의 구조
  - ✓ FAT file system 실습
    - 환경 구축
    - file recovery
    - logging file access
- NAND FLASH & F2FS
  - ✓ NAND FLASH 개요
  - ✓ F2FS의 구조
  - ✓ F2FS 실습
    - 환경 구축
    - F2FS kernel source code 분석 & 수정 실습
- Performance evaluation
  - ✓ Benchmark 개요
  - ✓ 다양한 Benchmark들의 사용법 & 실습



Linux Kernel Camp 2017

- 실습 이미지 가져오기



✓ cpu 설정, memory 를 자신의 컴퓨터에 맞게 설정

1. FAT file system

FAT file system 실습 1

- FAT file system Recovery 실습의 목표
  - ✓ Ram disk/file system 모듈을 통해 모듈의 사용법과 file system 마운트 방법을 숙지한다.
  - ✓ FAT file system의 전체적인 구조 및 원리를 이해한다.
  - ✓ 향후 file system에 필요한 tool작성 능력을 배양한다.
- FAT file system Recovery 실습 내용
  - ✓ FAT file system 실습을 위해 ramdisk를 생성하고 이를 vfat 형식으로 포맷 후 마운트한다.
  - ✓ FAT file system에서 파일을 삭제하는 경우 삭제하고자 하는 파일 엔트리의 첫 바이트를 “0xe5”로 변경을 시켜놓고 향후에 덮어쓰기를 통해 완전히 삭제된다.
  - ✓ 만약 삭제한 파일이 덮어쓰기 되지 않았다면 이 파일은 복구가 가능할 것이다. 또한 파일의 데이터를 담고있는 클러스터 덮어쓰기 되지 않았다면 데이터도 복구가 가능할 것 이다.
  - ✓ 이를 이용해 새로운 tool을 만들고 삭제한 파일을 복구한다.

FAT file system 실습 2

- FAT file system logging 실습의 목표
  - ✓ FAT file system 이 실제 기록되는 Hex값을 통해 관련 자료구조를 이해한다.
  - ✓ FAT file system 을 code level 에서 분석을 통해 커널 코드의 이해도를 높인다.
  - ✓ 향후 필요에 따라 file system 을 수정 할 수 있는 능력을 배양한다.
- FAT file system logging 실습 내용
  - ✓ FAT file system 실습을 위해 ramdisk를 생성하고 이를 vfat 형식으로 포맷 후 마운트한다.
  - ✓ FAT file system 의 기존 구조에 logging을 위한 새로운 영역을 확보한다.
  - ✓ 확보한 logging 영역에 기록될 자료구조 작성 및 이를 기존의 자료구조들의 정보로 초기화 한다.
  - ✓ 파일에 대한 접근이 수행되는 code 부분을 수정하여 logging 함수를 작성한다.
  - ✓ 이를 이용해 파일에 대한 접근을 기록할 수 있도록 한다.

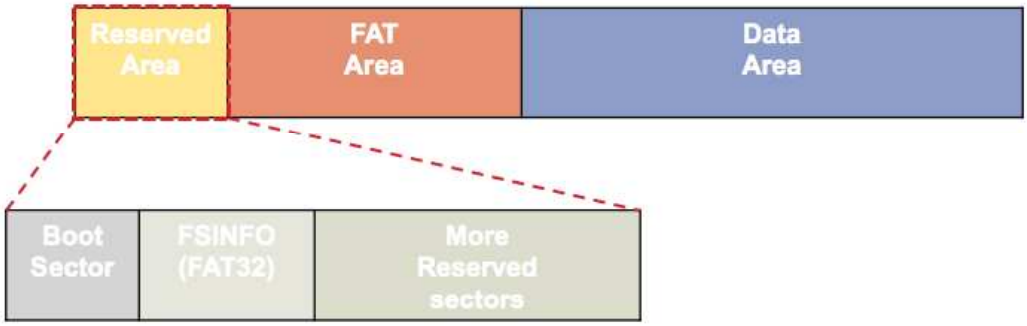
FAT file system이란?

- FAT file system
  - ✓ FAT(File Allocation Table)은 이름 그대로 파일의 할당 정보를 표현한 테이블이다.
  - ✓ FAT file system은 가장 범용적인 파일시스템이다. 구조가 간단하다는 장점을 기반으로 다양한 OS에서 지원하며 메모리 카드, 디지털 카메라, 플래시 메모리 등에 널리 사용되고있다.
  - ✓ FAT은 크게 FAT12, FAT16, FAT32로 나눌 수 있으며 FAT 뒤의 숫자는 비트 수로, 최대 표현 가능한 클러스터의 수를 의미한다.

FAT 형식	최대 표현 가능한 클러스터 수
FAT12	4084(2^12 – 12)
FAT 16	65,524
FAT 32	268,435,444

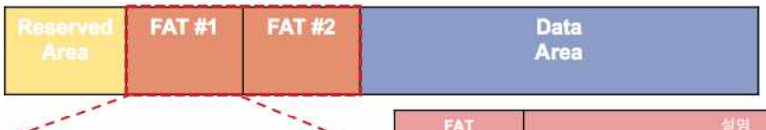
FAT file system의 구조

- 예약된 영역(Reserved Area)
  - ✓ 예약된 영억은 FAT file system에서 가장 앞 부분에 위치하는 구조로서 여러개의 sector로 구성된다.
  - ✓ 예약된 영역의 크기는 기본적으로 FAT12/16에서는 1sector, FAT32에서는 32개의 sector로 구성된다.
  - ✓ 예약된 영역은 다시 boot sector, FSINFO(File System Information), 추가적인 sector로 구분된다.



FAT file system의 구조

- FAT(File Allocation Table) 영역
  - ✓ FAT 영역은 파일 할당 테이블이다. 다시 말해, 데이터 영역에 저장된 파일들의 할당 관계를 표시해주는 테이블이다.
  - ✓ FAT 영역의 목전은 데이터 영역에 저장된 파일의 클러스터 할당 관계를 테이블 형태로 보여주는 것이다.
  - ✓ 데이터 영역에 저장된 파일은 연속적인 클러스터에 저장될 수도 있지만, 조각나 저장될 수도 있다. FAT 영역은 이러한 조각난 클러스터의 연결 상태까지도 표현해 준다.



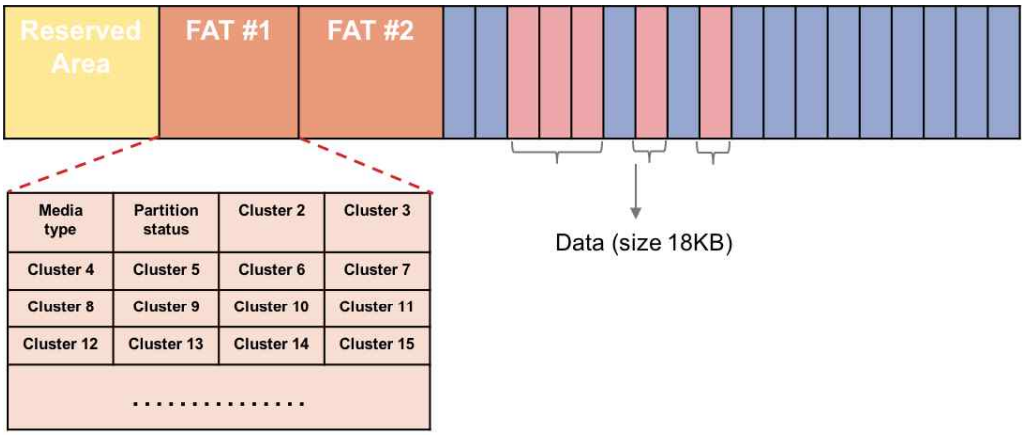
Media type	Partition status	Cluster 2	Cluster 3
Cluster 4	Cluster 5	Cluster 6	Cluster 7
Cluster 8	Cluster 9	Cluster 10	Cluster 11
Cluster 12	Cluster 13	Cluster 14	Cluster 15
.....			

FAT	설명
0x00	해당 클러스터가 사용이 가능하다는 의미로, 비할당 상태를 나타낸다. 파일 시스템을 새롭게 포맷하거나 파일을 삭제할 경우 나타나는 값으로, 파일을 할당하고자 할 경우 FAT 영역을 검색하여 해당 값인 클러스터에 할당한다.
0xff8	파일의 끝(End-Of-Marker)를 나타낸다. 즉, 파일에 할당된 마지막 클러스터를 의미한다.
0xff7	배드 클러스터를 나타낸다. 배트 섹터가 발생하여 해당 클러스터에 데이터를 저장할 수 없음을 나타낸다.

- Data 영역
  - ✓ 데이터 영역에 저장되는 데이터는 크게 파일의 메타 정보 (파일 이름, 크기, 타입, 시간 정보, 시작 클러스터 위치)를 지정하고 있는 디렉터리 엔트리 구조와 실제 파일 데이터로 나뉜다.

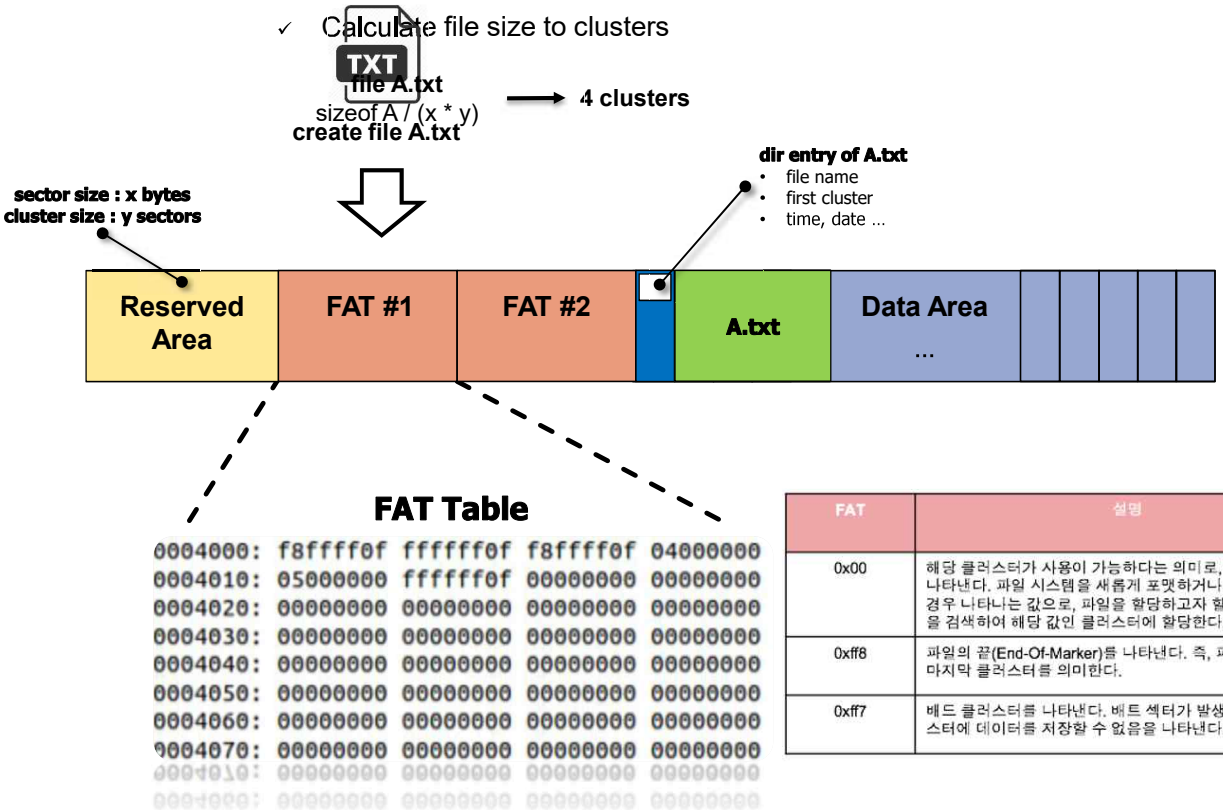
FAT file system Mechanism

- FAT table
  - ✓ 루트 디렉토리에 위치한 특정 파일이나 폴더에 접근하고자 하는 경우 루트 디렉터리의 디렉터리 엔트리를 검색하면서 해당 파일 이름을 찾는다.
  - ✓ 파일 이름을 가진 디렉터리 엔트리를 찾았다면 시작 클러스터 정보를 얻어서 해당 위치부터 파일 크기 만큼 파일을 획득하면 될 것이다.
  - ✓ 하지만 파일은 항상 연속된 클러스터로 표현되지 않고 조각 날 수 있기 때문에 이러한 파일 정보는 FAT영역에서 시작 클러스터부터 따라가면서 원하는 클러스터 만큼 획득하면 된다.



FAT file system Mechanism

- ✓ Calculate file size to clusters

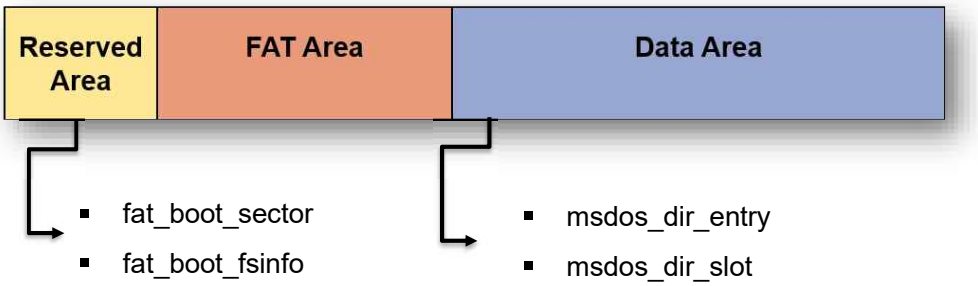


FAT	설명
0x00	해당 클러스터가 사용이 가능하다는 의미로, 비할당 상태를 나타낸다. 파일 시스템을 새롭게 포맷하거나 파일을 삭제할 경우 나타나는 값으로, 파일을 할당하고자 할 경우 FAT 영역을 검색하여 해당 값인 클러스터에 할당한다.
0xff8	파일의 끝(End-Of-Marker)를 나타낸다. 즉, 파일에 할당된 마지막 클러스터를 의미한다.
0xff7	배드 클러스터를 나타낸다. 배드 섹터가 발생하여 해당 클러스터에 데이터를 저장할 수 없음을 나타낸다.

FAT file system의 자료구조 분석

On disk FAT structure

- ✓ File System format 의 정보를 보관하여 관련 연산에 활용한다.
- ✓ Data 영역에 저장된 정보에 관한 meta data 정보를 보관 하여 관련 연산에 활용한다.



FAT file system 자료구조 분석

On Disk FAT structure

- ✓ fat_boot_sector

```
root@son:/usr/src/linux-4.6.4/fs/fat# xxd -g 1 -l 256 -s+0 /dev/rdk
0000000: eb 58 90 6d 6b 66 73 2e 66 61 74 00 02 08 20 00  .X.mkfs.fat... .
0000010: 02 00 00 00 00 f8 00 00 10 00 04 00 00 00 00 00  ................
0000020: 00 00 80 00 f1 1f 00 00 00 00 00 00 02 00 00 00  ................
0000030: 01 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000040: 80 01 29 62 8f 5c 9d 4e 4f 20 4e 41 4d 45 20 20  ..)b.\.NO NAME
0000050: 20 20 46 41 54 33 32 20 20 20 0e 1f be 77 7c ac    FAT32   ...w|.
0000060: 22 c0 74 0b 56 b4 0e bb 07 00 cd 10 5e eb f0 32  ".t.V.......^..2
```

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Jump to Boot code			OEM NAME in ASCII								Bytes per sector		S p Clust	Size of reserved area	
FATs	Max num of files in root dir		Sectors in File System(16 bit)		Media	Sectors in FAT area(12,16)		Sectors per track		Number of heads		Number of sectors before the start of partition			
Sectors in File System(32 bit, if 19-20byte area can't afford)															

- FAT File System 의 첫번째 섹터에 위치한다.
- 전체 FAT File System의 섹터의 수, 섹터의 크기, FAT영역의 섹터 크기, FAT영역의 개수 등 Boot Record 관련 정보를 가지고 있는 구조체 이다.
- FAT12/FAT16 과 FAT32가 가지는 구조체의 정보는 다르다 . 이중 다음은 fat_boot_sector struct중 공통적인 첫 36byte 부분의 모습이다.
- 나머지 정보들은 공용체로 존재하여 FAT File System들간 서로 다른 정보를 가진다.

```
struct fat_boot_sector {
    __u8    ignored[3]; /* Boot strap short or near jump */
    __u8    system_id[8];   /* Name - can be used to special case
                partition manager volumes */
    __u8    sector_size[2]; /* bytes per logical sector */
    __u8    sec_per_clus;   /* sectors/cluster */
    __le16  reserved;   /* reserved sectors */
    __u8    fats;       /* number of FATs */
    __u8    dir_entries[2]; /* root directory entries */
    __u8    sectors[2]; /* number of sectors */
    __u8    media;      /* media code */
    __le16  fat_length; /* sectors/FAT */
    __le16  secs_track; /* sectors per track */
    __le16  heads;      /* number of heads */
    __le32  hidden;     /* hidden sectors (unused) */
    __le32  total_sect; /* number of sectors (if sectors == 0) */
```

FAT file system 자료구조 분석

- On Disk FAT structure
  - ✓ fat_boot_sector
    - fat12/fat16

```
root@son:/usr/src/linux-4.6.4/fs/fat# xxd -g 1 -l 256 -s+0 /dev/rdk
0000000: eb 58 90 6d 6b 66 73 2e 66 61 74 00 02 08 20 00  .X.mkfs.fat... .
0000010: 02 00 00 00 00 f8 00 00 10 00 04 00 00 00 00 00  ................
0000020: 00 00 80 00 f1 1f 00 00 00 00 00 00 02 00 00 00  ................
0000030: 01 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000040: 80 01 29 62 8f 5c 9d 4e 4f 20 4e 41 4d 45 20 20  ..)b.\.NO NAME
0000050: 20 20 46 41 54 33 32 20 20 20 0e 1f be 77 7c ac    FAT32   ...w|.
0000060: 22 c0 74 0b 56 b4 0e bb 07 00 cd 10 5e eb f0 32  ".t.V.......^..2
0000070: e4 cd 16 cd 19 eb fe 54 68 69 73 20 69 73 20 6e  .......This is n
```

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Jump to Boot code			OEM NAME in ASCII								Bytes per sector		S p Clust	Size of reserved area	
FATs	Max num of files in root dir		Sectors in File System(16 bit)		Media	Sectors in FAT area(12,16)		Sectors per track		Number of heads	Number of sectors before the start of partition				
Sectors in File System(32 bit, if 19-20byte area can't afford)				Drive num	Not used	Sign ature	Volume serial number				File System label				
						File System type									

- BIOS INT drive number 와 확장 BIOS signature 등을 가진다. BIOS signature 는 0x29 로 채워진다.
- File System Label 에는 File System 의 이름인 "FAT", "FAT12", "FAT16" 등으로 채워진다.

```
struct {
    /*  Extended BPB Fields for FAT16 */
    __u8    drive_number;   /* Physical drive number */
    __u8    state;      /* undocumented, but used
                for mount state. */
    __u8    signature;  /* extended boot signature */
    __u8    vol_id[4];  /* volume ID */
    __u8    vol_label[11];  /* volume label */
    __u8    fs_type[8];     /* file system type */
    /* other fields are not added here */
} fat16;
```


FAT file system 자료구조 분석

- On Disk FAT structure
  - ✓ fat_boot_sector
    - fat32

```
root@son:/usr/src/linux-4.6.4/fs/fat# xxd -g 1 -l 256 -s+0 /dev/rdk
0000000: eb 58 90 6d 6b 66 73 2e 66 61 74 00 02 08 20 00  .X.mkfs.fat... .
0000010: 02 00 00 00 00 f8 00 00 10 00 04 00 00 00 00 00  ................
0000020: 00 00 80 00 f1 1f 00 00 00 00 00 00 02 00 00 00  ................
0000030: 01 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000040: 80 01 29 62 8f 5c 9d 4e 4f 20 4e 41 4d 45 20 20  ..)b.\.NO NAME
0000050: 20 20 46 41 54 33 32 20 20 20 0e 1f be 77 7c ac    FAT32   ...w|.
0000060: 22 c0 74 0b 56 b4 0e bb 07 00 cd 10 5e eb f0 32  ".t.V.......^..2
0000070: e4 cd 16 cd 19 eb fe 54 68 69 73 20 69 73 20 6e  .......This is n
```

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Jump to Boot code			OEM NAME in ASCII								Bytes per sector		S p Clust	Size of reserved area	
FATs	Max num of files in root dir		Sectors in File System(16 bit)		Media	Sectors in FAT area(12,16)		Sectors per track		Number of heads	Number of sectors before the start of partition				
Sectors in File System(32 bit, if 19-20byte area can't afford)				Sectors in FAT area				Define flags		Major version number	Root directory cluster				
FSINFO sector		Boot Sector copy sector		Reserved											
Drive num	None	singn ature	Volume Serial Number				Volume label in ASCII								
		File System Type													

- Define flag를 통해 두번째 FAT영역의 사용여부를 결정한다.
- FSINFO struct 의 위치를 나타내는 정보를 가진다.
- File System Type에 "FAT32" 가 저장된다.

```
struct {
    /* only used by FAT32 */
    __le32  length;     /* sectors/FAT */
    __le16  flags;      /* bit 8: fat mirroring,
                low 4: active fat */
    __u8    version[2]; /* major, minor filesystem
                version */
    __le32  root_cluster;   /* first cluster in
                root directory */
    __le16  info_sector;    /* filesystem info sector */
    __le16  backup_boot;    /* backup boot sector */
    __le16  reserved2[6];   /* Unused */
    /* Extended BPB Fields for FAT32 */
    __u8    drive_number;   /* Physical drive number */
    __u8    state;          /* undocumented, but used
                for mount state. */
    __u8    signature;  /* extended boot signature */
    __u8    vol_id[4];  /* volume ID */
    __u8    vol_label[11];  /* volume label */
    __u8    fs_type[8];     /* file system type */
    /* other fields are not added here */
} fat32;
```

FAT file system 자료구조 분석

- On Disk FAT structure
  - ✓ fat_fs_info

```
root@son:/usr/src/linux-4.6.4/fs/fat# xxd -g 1 -l 512 -s+0x200 /dev/rdk
0000200: 52 52 61 41 00 00 00 00 00 00 00 00 00 00 00 00  RRaA............
0000210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00003b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00003c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00003d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00003e0: 00 00 00 00 72 72 41 61 fe f7 0f 00 06 00 00 00  ....rrAa........
00003f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa  ..............U.
```

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Fixed Signature			Not used												
Not used															
Not used				Fixed Signature				Number of Free Cluster				Next Free Cluster			
Not used															

- 필수적인 On-Disk 구조체는 아니며 OS 에게 다음 클러스터 할당 정보를 쉽게 제공하기 위한 구조체이다.
- 두 Signature값은 0x41615252 와 0x61417272의 고정 값을 갖는다.

```
struct fat_boot_fsinfo {
    __le32   signature1;     /* 0x41615252L */
    __le32   reserved1[120];     /* Nothing as far as I can tell */
    __le32   signature2;     /* 0x61417272L */
    __le32   free_clusters; /* Free cluster count.  -1 if unknown */
    __le32   next_cluster;  /* Most recently allocated cluster */
    __le32   reserved2[4];
};
```


FAT file system 자료구조 분석

- On Disk FAT structure
  - ✓ msdos_dir_entry

```
root@son:/usr/src/linux-4.6.4/fs/fat# xxd -g 1 -l 128 -s+0xBFE600 /dev/rdk
0bfe600: 41 66 00 69 00 6c 00 65 00 2e 00 0f 00 85 63 00  Af.i.l.e......c.
0bfe610: 00 00 ff ff ff ff ff ff ff ff 00 00 ff ff ff ff  ................
0bfe620: 46 49 4c 45 20 20 20 20 43 20 20 20 00 00 34 4c  FILE    C   ..4L
0bfe630: 04 49 04 49 00 00 34 4c 04 49 00 40 00 00 00 00  .I.I..4L.I......
0bfe640: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0x00 or 0xe5 or name	Characters 2 to 11 of file name in ASCII										Attribute	Reserved	Ctime(sec)	Ctime(h,m,s)	
Created day		Accessed day		First cluster(High 2byte)		Written time		Written day		First cluster(Low 2byte)		Size of file			

- data 들이 실제로 저장되는 영역이다.
- 파일의 할당 상태, 속성, 생성 시간, 접근 시간, 첫번째 클러스터 위치, 크기 등의 meta data 관련 정보를 저장한다.

File Attribute	Description
0x01	Read only
0x02	Hidden file
0x04	System file
0x08	Volume label
0x0f	Long file name
0x10	Directory
0x20	Regular file

```
struct msdos_dir_entry {
    __u8    name[MSDOS_NAME];/* name and extension */
    __u8    attr;        /* attribute bits */
    __u8    lcase;       /* Case for base and extension */
    __u8    ctime_cs;    /* Creation time, centiseconds (0-199) */
    __le16  ctime;       /* Creation time */
    __le16  cdate;       /* Creation date */
    __le16  adate;       /* Last access date */
    __le16  starthi;     /* High 16 bits of cluster in FAT32 */
    __le16  time,date,start;/* time, date and first cluster */
    __le32  size;        /* file size (in bytes) */
};
```

FAT file system 자료구조 분석

- On Disk FAT structure
  - ✓ msdos_dir_entry
    - 파일의 생성 시간, 접근 시간 등의 정보를 다음과 같이 16bit로 저장하여 관리한다.

```
root@son:/usr/src/linux-4.6.4/fs/fat# xxd -g 1 -l 256 -s+0xBFE600 /dev/rdk
0bfe600: e5 2e 00 74 00 65 00 73 00 74 00 0f 00 a1 2e 00  ...t.e.s.t......
0bfe610: 74 00 78 00 74 00 2e 00 73 00 00 00 77 00 70 00  t.x.t...s...w.p.
0bfe620: e5 45 53 54 54 58 7e 31 53 57 50 20 00 00 41 52  .ESTTX~1SWP ..AR
0bfe630: 04 49 04 49 00 00 41 52 04 49 00 00 00 00 00 00  .I.I..AR.I......
0bfe640: 41 74 00 65 00 73 00 74 00 2e 00 0f 00 8f 74 00  At.e.s.t......t.
0bfe650: 78 00 74 00 00 00 ff ff ff ff 00 00 ff ff ff ff  x.t.............
0bfe660: 54 45 53 54 20 20 20 20 54 58 54 20 00 00 47 52  TEST    TXT ..GR
0bfe670: 04 49 04 49 00 00 47 52 04 49 04 00 28 42 00 00  .I.I..GR.I..(B..
0bfe680: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

지워진 파일의 경우 이름의 첫글자가 0xe5

생성 시간 생성 날짜

일반 파일

    - date
      - Day : 하위 0~4 위치의 bit 를 사용하여 1~31 범위의 날짜를 표현한다.
      - Month : 5~8 위치의 bit 를 사용하여 1~12 범위의 달을 표현한다.
      - Year : 9~15 위치의 bit 를 사용하여 1980 ~ 2107 범위의 년도를 표현한다.

0x0449 -> 0x4904 (little endian) ➡

Year	Month	Day
0 1 0 0 1 0 0	1 0 0 0	0 0 1 0 0
7	4	5

➡ **2016년 8월 4일**

    - time
      - Second : 하위 0~4 위치의 bit 를 사용하여 0~29의 값으로 2초 간격으로 초를 표현한다.
      - Minute : 5 ~ 10 위치의 bit를 사용하여 0~59 범위의 분을 표현한다.
      - Hour : 11~15 위치의 bit를 사용하여 0~23 범위의 시간을 표현한다.

0x4752 -> 0x5247 (little endian) ➡

Hour	Minute	Seconds
0 1 0 1 0	0 1 0 0 1 0	0 0 1 1 1
5	6	5

➡ **10시 18분 14초**

FAT file system 자료구조 분석

- On Disk FAT structure
  - ✓ msdos_dir_slot

```
0bfe600: 42 76 00 65 00 72 00 73 00 65 00 0f 00 e9 2e 00  Bv.e.r.s.e......
0bfe610: 73 00 68 00 00 00 ff ff ff ff 00 00 ff ff ff ff  s.h.............
0bfe620: 01 69 00 6e 00 73 00 6d 00 6f 00 0f 00 e9 64 00  .i.n.s.m.o....d.
0bfe630: 5f 00 66 00 61 00 74 00 5f 00 00 00 69 00 6e 00  _.f.a.t._...i.n.
0bfe640: 49 4e 53 4d 4f 44 7e 31 53 48 20 20 00 00 8b 26  INSMOD~1SH  ...&
0bfe650: 05 49 05 49 00 00 8b 26 05 49 03 00 30 00 00 00  .I.I...&.I..0...
0bfe660: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0bfe670: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Sequence number, Alloc status	File name characters 1-5 (Unicode)										Attribute	Reserved	checksum	File name Characters 6-11(Unicode)	
File name Characters 6-11(Unicode)										Reserved		File name Characters 12-13 (Unicode)			

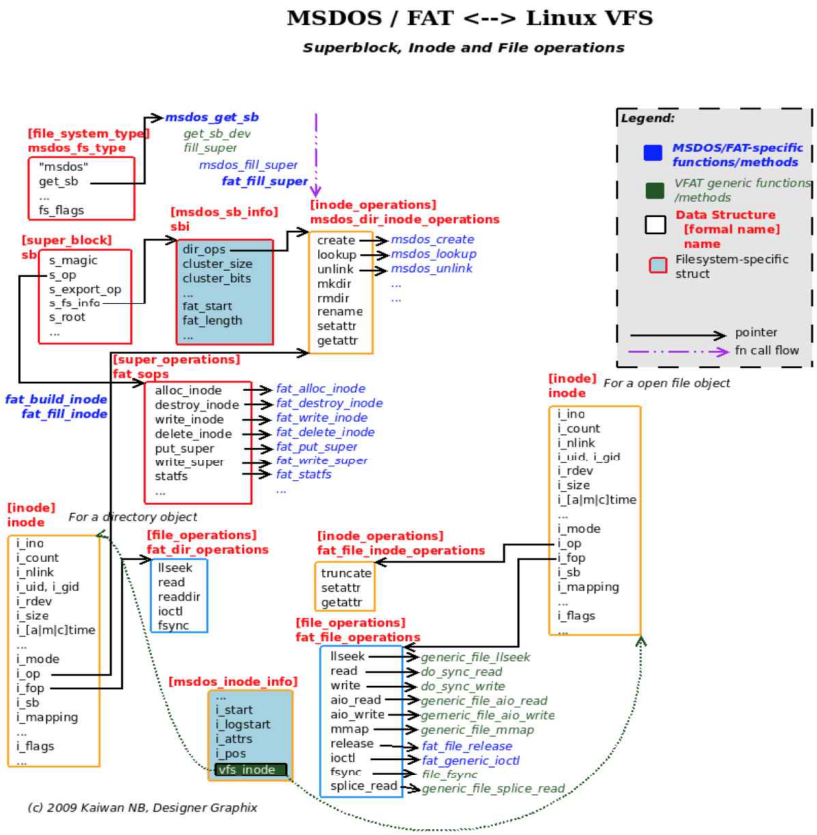
- 파일의 이름이 MSDOS_NAME의 길이보다 길 때 생성된다.
- msdos_dir_entry 보다 먼저 저장된다.
- 0x0f 의 Attribute를 가지고 있다.
- Sequence Number는 위로 갈수록 증가하며 마지막 entry에는 0x40 과 OR연산 값을 저장한다.
- 첫번째 Byte는 Sequence numbe이며 이를 통해 msdos_dir_slot(Long File Name) 의 순서를 알 수 있다. msdos_dir_entry의 바로 위가 첫번째 LFN이다.

```
/* Up to 13 characters of the name */
struct msdos_dir_slot {
    __u8    id;     /* sequence number for slot */
    __u8    name0_4[10];    /* first 5 characters in name */
    __u8    attr;       /* attribute byte */
    __u8    reserved;   /* always 0 */
    __u8    alias_checksum; /* checksum for 8.3 alias */
    __u8    name5_10[12];   /* 6 more characters in name */
    __le16   start;     /* starting cluster number, 0 in long slots */
    __u8    name11_12[4];   /* last 2 characters in name */
};
```

FAT file system의 자료구조 분석

- Off disk FAT structure
  - ✓ File System 관련 연산들의 수행 및 on disk structure의 정보를 보관하기 위한 정보로 구성된다.
    - kernel dependent structure
      - super_block, inode, dentry, buffer_head …
    - mount structure
      - fat_mount_options, vfat_fs_type(file_system_type)
    - super structure
      - msdos_sb_info, fat_bios_param_block, fat_sops(super_operation)
    - inode structure
      - msdos_inode_info
    - directory information
      - fat_dir_operations(file_operations), fat_ioctl_filldir_callback, vfat_ci_dentry_ops / vfat_dentry_ops(dentry_operations), vfat_dir_inode_operations(inode_operations)
    - file structure
      - fat_slot_info, fat_file_operations(file_operations), fat_file_inode_opeartions(inode_operations)
    - fat area structure
      - fat12_ops / fat16_ops / fat32_ops(fatent_operations), fat_entry
    - page cache structure
      - fat_cache , fat_cache_id, fat_aops(address_space_operations)
    - …

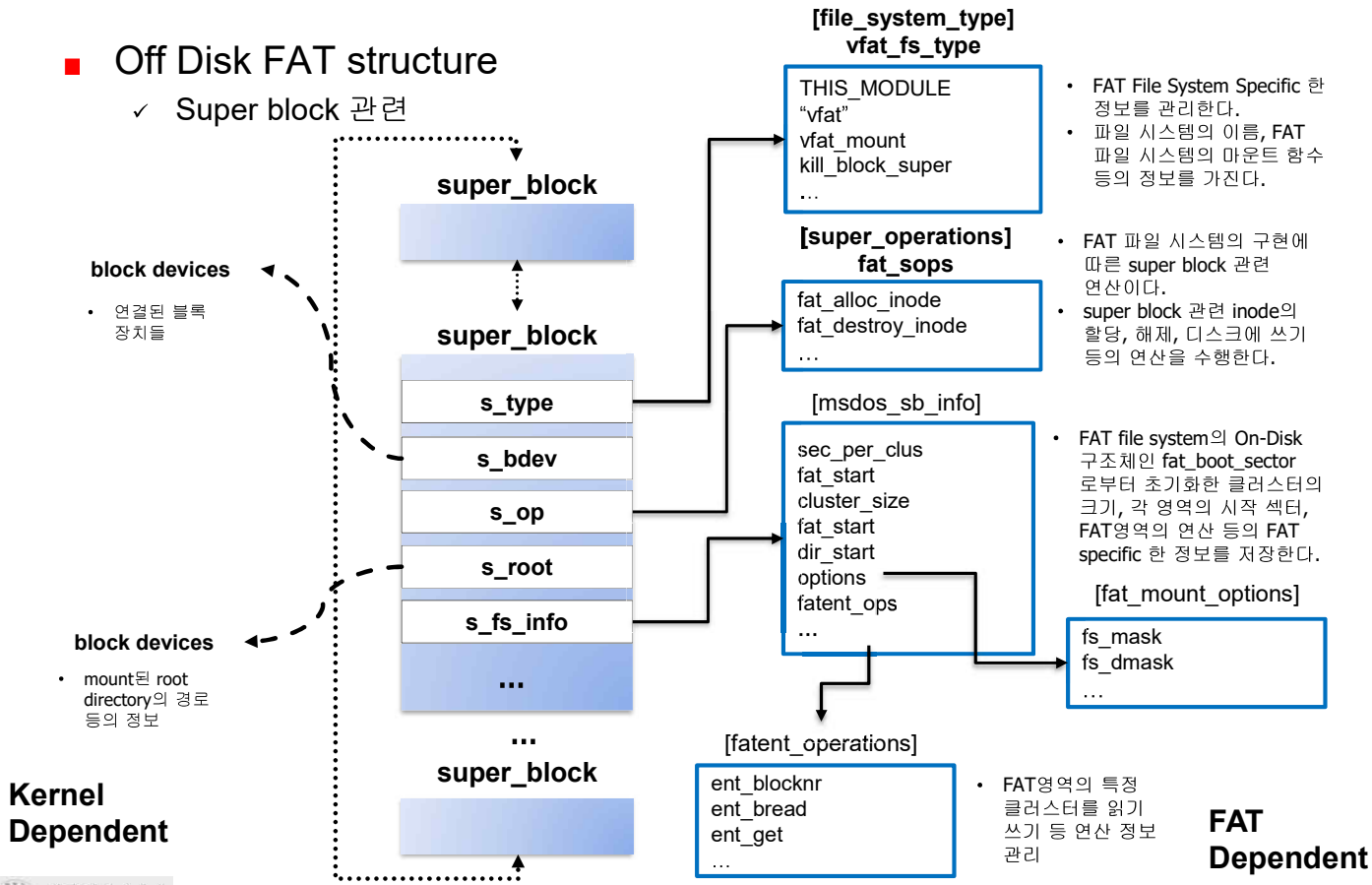
FAT file system 의 자료구조 분석



<https://kaiwantech.wordpress.com/2009/08/17/inside-the-msdos-fat-linux-vfs-implementation/>

FAT file system 자료구조 분석

- Off Disk FAT structure
  - ✓ Super block 관련



FAT file system 구조

- superblock
  - ✓ superblock의 초기화는 file system이 mount되는 시점에 수행된다.

```
static struct file_system_type vfat_fs_type = {
    .owner      = THIS_MODULE,
    .name       = "vfat",
    .mount      = vfat_mount,
    .kill_sb    = kill_block_super,
    .fs_flags   = FS_REQUIRES_DEV,
};
MODULE_ALIAS_FS("vfat");
```

</usr/src/linux/fs/fat/naemi_vfat.c>

```
static struct dentry *vfat_mount(struct file_system_type *fs_type,
                int flags, const char *dev_name,
                void *data)
{
    return mount_bdev(fs_type, flags, dev_name, data, vfat_fill_super);
}
```

```
static int vfat_fill_super(struct super_block *sb, void *data, int silent)
{
    return fat_fill_super(sb, data, silent, 1, setup);
}
```

FAT file system 구조

- superblock
  - ✓ vfat_fill_super 함수에서는 disk의 superblock을 읽어들여서 fat_boot_sector(boot record), superblock(msdos_sb_info), root directory(inode) 등의 정보를 초기화 한다.

</usr/src/linux/fs/fat/inode.c>

```
sb->s_flags |= MS_NODIRATIME;
sb->s_magic = MSDOS_SUPER_MAGIC;
sb->s_op = &fat_sops;
sb->s_export_op = &fat_export_ops;
```

superblock 초기화

```
bh = sb_bread(sb, 0);
if (bh == NULL) {
    fat_msg(sb, KERN_ERR, "unable to read boot sector");
    goto out_fail;
}

b = (struct fat_boot_sector *) bh->b_data;
```

boot sector

```
sbi->cluster_size = sb->s_blocksize * sbi->sec_per_clus;
sbi->cluster_bits = ffs(sbi->cluster_size) - 1;
sbi->fats = b->fats;
sbi->fat_bits = 0;      /* Don't know yet */
sbi->fat_start = le16_to_cpu(b->reserved);
sbi->fat_length = le16_to_cpu(b->fat_length);
sbi->root_cluster = 0;
sbi->free_clusters = -1;    /* Don't know yet */
sbi->free_clus_valid = 0;
sbi->prev_free = FAT_START_ENT;
sb->s_maxbytes = 0xffffffff;
```

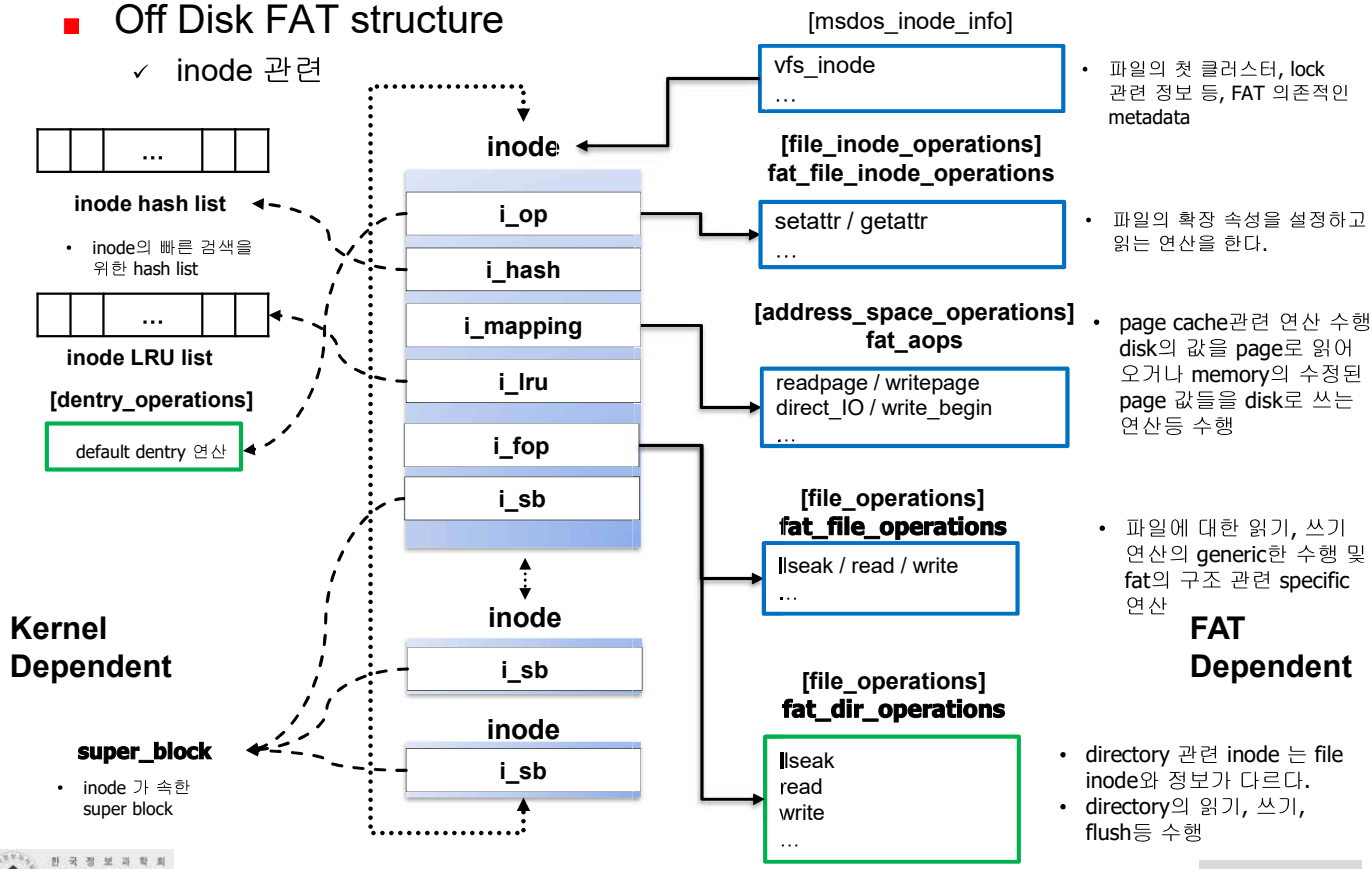
msdos_sb_info 초기화

```
root_inode = new_inode(sb);
if (!root_inode)
    goto out_fail;
root_inode->i_ino = MSDOS_ROOT_INO;
root_inode->i_version = 1;
error = fat_read_root(root_inode);
if (error < 0) {
    iput(root_inode);
    goto out_fail;
}
error = -ENOMEM;
insert_inode_hash(root_inode);
fat_attach(root_inode, 0);
sb->s_root = d_make_root(root_inode);
```

root inode 생성

FAT file system 자료구조 분석

- Off Disk FAT structure
  - ✓ inode 관련



FAT file system 구조

- Inode
  - ✓ 새로운 파일이 생성되는 경우 file system에서는 항상 inode를 할당해주어야한다.
  - ✓ Inode의 생성, 삭제등의 연산은 struct super_operations에서 관리한다.

```
static const struct super_operations fat_sops = {
    .alloc_inode    = fat_alloc_inode,
    .destroy_inode  = fat_destroy_inode,
    .write_inode    = fat_write_inode,
    .evict_inode    = fat_evict_inode,
    .put_super  = fat_put_super,
    .statfs     = fat_statfs,
    .remount_fs = fat_remount,

    .show_options   = fat_show_options,
};
```

```
static struct inode *fat_alloc_inode(struct super_block *sb)
{
    struct msdos_inode_info *ei;
    ei = kmem_cache_alloc(fat_inode_cachep, GFP_NOFS);
    if (!ei)
        return NULL;

    init_rwsem(&ei->truncate_lock);
    return &ei->vfs_inode;
}
```

</usr/src/linux/fs/fat/inode.c>

FAT file system 구조

- Inode
  - ✓ Inode의 초기화를 담당하는 함수는 fat_fill_inode이다.

directory inode 인 경우

```
int fat_fill_inode(struct inode *inode, struct msdos_dir_entry *de)
{
    struct msdos_sb_info *sbi = MSDOS_SB(inode->i_sb);
    int error;

    MSDOS_I(inode)->i_pos = 0;
    inode->i_uid = sbi->options.fs_uid;
    inode->i_gid = sbi->options.fs_gid;
    inode->i_version++;
    inode->i_generation = get_seconds();

    if ((de->attr & ATTR_DIR) && !IS_FREE(de->name)) {
        inode->i_generation &= ~1;
        inode->i_mode = fat_make_mode(sbi, de->attr, S_IRWXUGO);
        inode->i_op = sbi->dir_ops;
        inode->i_fop = &fat_dir_operations;

        MSDOS_I(inode)->i_start = fat_get_start(sbi, de);
        MSDOS_I(inode)->i_logstart = MSDOS_I(inode)->i_start;
        error = fat_calc_dir_size(inode);
        if (error < 0)
            return error;
        MSDOS_I(inode)->mmu_private = inode->i_size;

        set_nlink(inode, fat_subdirs(inode));
```

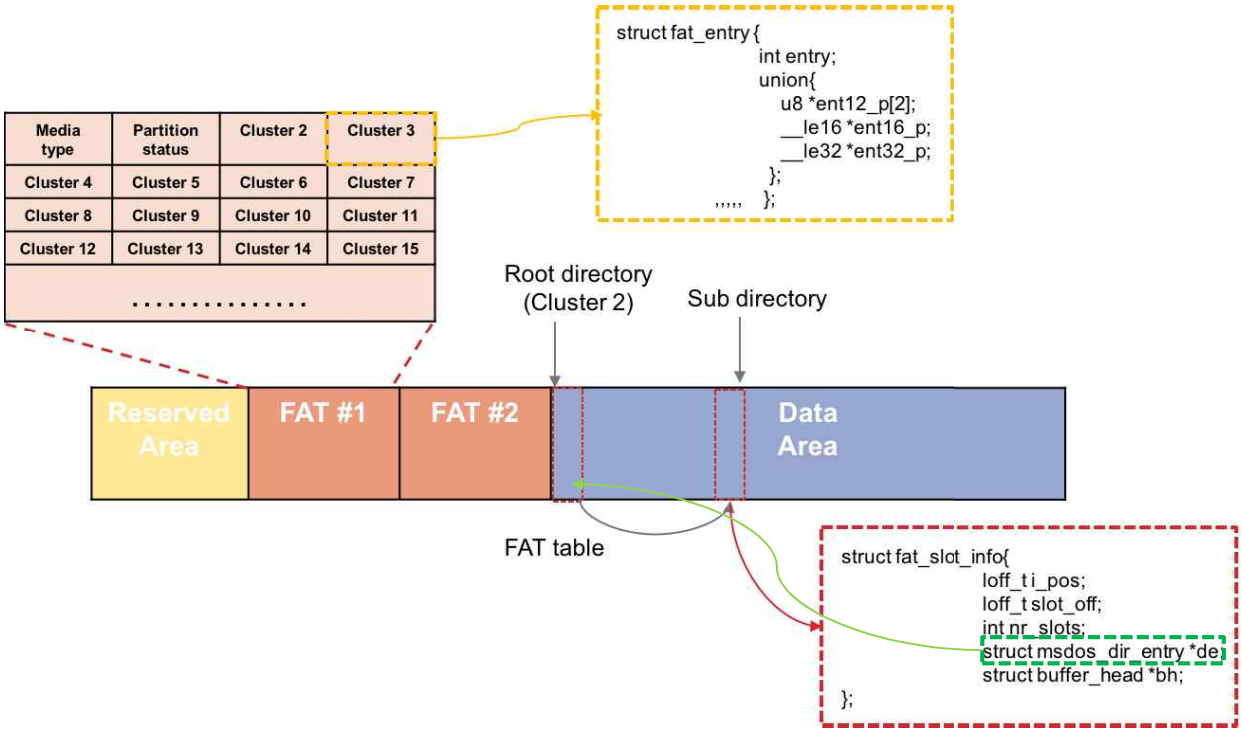
file inode 인 경우

```
} else { /* not a directory */
    inode->i_generation |= 1;
    inode->i_mode = fat_make_mode(sbi, de->attr,
        ((sbi->options.showexec && !is_exec(de->name + 8))
         ? S_IRUGO|S_IWUGO : S_IRWXUGO));
    MSDOS_I(inode)->i_start = fat_get_start(sbi, de);

    MSDOS_I(inode)->i_logstart = MSDOS_I(inode)->i_start;
    inode->i_size = le32_to_cpu(de->size);
    inode->i_op = &fat_file_inode_operations;
    inode->i_fop = &fat_file_operations;
    inode->i_mapping->a_ops = &fat_aops;
    MSDOS_I(inode)->mmu_private = inode->i_size;
}
```

</usr/src/linux/fs/fat/inode.c>

FAT file system 자료구조 분석

- Off Disk FAT structure
  - ✓ FAT 영역 및 Data 관련



FAT file system 실습

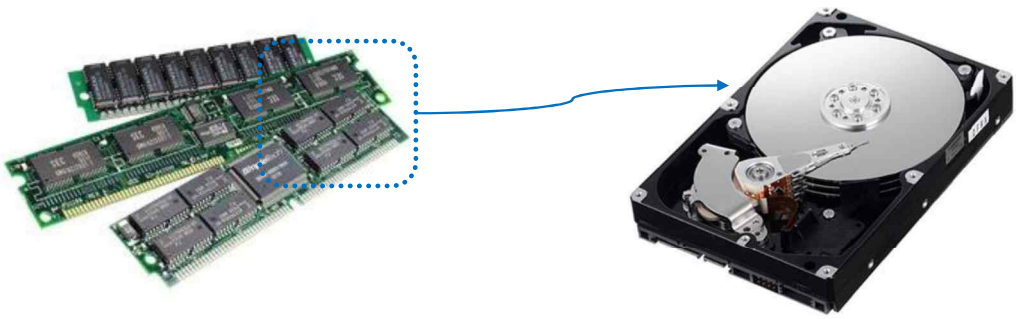
환경 구축

실습환경

- 실습환경
  - ✓ 본 실습은 virtual box를 통해 진행되며 os는 ubuntu 12.04, linux kernel version은 3.19.6을사용한다.
  - ✓ Ubuntu machine name : fs-lecture / password : 1234
  - ✓ ***모든 실습은 root권한으로 진행한다.***
    - **sudo su**
  - ✓ 실습 자료의 위치는 /home/fs-lecture/lecture에 의 진행은 이 디렉토리에서 진행한다.
  - ✓ 실습에 사용되는 자료들은 다음과 같다.
    - f2fs-tools_1.4.1
    - rxdsk-3.0
    - fat_module
    - practice/fat_recovery_practice
    - practice/fat_log_practice
    - mtd_module
    - f2fs_module

실습 환경 구축 – ramdisk 생성

- RAM disk란?
  - ✓ RAM disk또는 주 기억 장치 활용 저장법은 디스크가 아닌 RAM(DRAM)을 이용하여 디스크 드라이브를 구현하는 방식이다.
  - ✓ RAM disk를 “RapidDisk”라는 open source tool을 이용해서 생성한 후, 이를 임의의 디렉토리에 mount한다.
  - ✓ 이 강의의 실습은 실제 컴퓨터상의 storage를 사용하지 않고 ram disk를 생성하여 이를 storage로 이용한다.



실습 환경 구축 – ramdisk 생성

- RAM disk 생성
  - ✓ rxdsk-3.0 디렉토리 이동한다. - "# cd rxdsk-3.0"
  - ✓ 관련 파일들을 설치한다. - "# apt-get install zlib1g", "# apt-get install zlib1g-dev"
  - ✓ 컴파일한다. - "# make , make install , make tools-install"
  - ✓ 컴파일 후에 명령어가 생성되었는지 확인한다. - "# ls –al /sbin | grep rapiddisk"

```
-rwxr-xr-x  1 root root    10448  7월  26  2014 partprobe
-rwxr-xr-x  1 root root    18792  3월  22  2012 pccardctl
-rwxr-xr-x  1 root root     6200  6월  18  2014 pivot_root
-rwxr-xr-x  1 root root    10400  3월  31  2012 plipconfig
-rwxr-xr-x  1 root root    64752  5월  17  2013 plymouthd
lrwxrwxrwx  1 root root        6  3월  12 07:54 poweroff -> reboot
-rwxr-xr-x  1 root root    32570  7월  13 15:03 rapiddisk
-rwxr-xr-x  1 root root    29976  3월  31  2012 rarp
-rwxr-xr-x  1 root root    10384  6월  18  2014 raw
-rwxr-xr-x  1 root root    14728  3월  12 07:54 reboot
lrwxrwxrwx  1 root root        7  3월  12 07:54 reload -> initctl
-rwxr-xr-x  1 root root    43952  2월  17 04:26 resize2fs
-rwxr-xr-x  1 root root     5637  7월  19  2012 resolvconf
lrwxrwxrwx  1 root root        7  3월  12 07:54 restart -> initctl
```

  - ✓ 컴파일을 통해 생성된 모듈을 적재 시킨다.
    - "# modprobe rxdsk "
    - # modprobe rxcache "

실습 환경 구축 – ramdisk 생성

- RAM disk 생성
  - ✓ module이 적재되었는지 확인 - "# lsmod | grep rx"

```
Module                  Size  Used by
rxcache                18168  0
rxdsk                  17555  0
nls_iso8859_1          12713  0
bnep                   19625  2
rfcomm                 69563  0
bluetooth             480834  10 bnep,rfcomm
radeon               1669236  2
ttm                    93349  1 radeon
drm_kms_helper        117991  1 radeon
```

  - ✓ 원하는 크기만큼 ram disk를 생성한다. - "# rapiddisk --attach (size)"
    - size에는 원하는 size를 숫자로 입력한다.
    - Ex) rappidisk --attach 2048 : 2048mb 크기의 ram disk를 생성한다.
      - FAT32로 실습을 진행하기 위해 램디스크 용량을 최소 1G이상으로 설정해야한다.
      - 두번째 실습의 편의를 위해 1024mb 또는 2048mb로 설정한다. (다른 값 설정 시 부트 레코드 확인 필요)
  - ✓ ram disk 디바이스 파일(rxd0)이 생성되었는지 확인한다. - "# ls –al /dev | grep rxd0"

```
lrwxrwxrwx   1 root root          4  7월   9 21:42 rtc -> rtc0
crw-------   1 root root    254,  0  7월   9 21:42 rtc0
brw-rw----   1 root disk     31,  0  7월  13 15:12 rxd0
brw-rw----   1 root disk      8,  0  7월   9 21:42 sda
brw-rw----   1 root disk      8,  1  7월   9 21:42 sda1
brw-rw----   1 root disk      8,  2  7월   9 21:42 sda2
brw-rw----   1 root disk      8,  5  7월   9 21:42 sda5
crw-rw----   1 root disk     21,  0  7월   9 21:42 sg0
```

실습 환경 구축 - module compile

- module compile
  - ✓ 본 실습은 커널 내에서 기본적으로 제공하는 ram disk나 built-in 되어있는 file system들을 사용하지 않고 실습 자료로 제공된 module을 이용한다.
  - ✓ module을 적재 시키고 사용하기에 앞서 compile을 해주어야 *.ko 형식의 module 이 생성된다.
  - ✓ compile을 하기 위해 앞으로의 실습에서 해당 디렉터리내로 이동하여 “make” 명령어를 입력하여야 한다.
  - ✓ compile을 다시 할 경우 위의 명령어를 통해 다시 compile 하기 전에, 이전 compile 에서 생성된 것을 “make clean” 명령어를 통해 모두 clean 시켜주어야 한다.

실습 환경 구축 - module compile

- module compile
  - ✓ 실습 자료로 받은 fat_module, f2fs_module, mtd_module에서 “make” 명령어를 통해 compile을 할 것이다. 다음은 컴파일 된 결과의 모습이다.

< before compile >

```
drwxr-xr-x  2 root     root     4096  7월  14 15:41 .
drwxr-xr-x 43 donghwa donghwa  4096  7월  14 15:31 ..
-rw-r--r--  1 root     root     4164  7월   9 22:47 Kconfig
-rw-r--r--  1 root     root      428  7월  13 16:35 Makefile
-rw-r--r--  1 root     root        0  7월  13 22:13 Module.symvers
-rw-r--r--  1 root     root     8767  7월   9 22:47 cache.c
-rw-r--r--  1 root     root    35629  7월  13 22:12 dir.c
-rw-r--r--  1 root     root    14228  7월   9 22:47 fat.h
-rw-r--r--  1 root     root    16942  7월  13 22:12 fatent.c
-rw-r--r--  1 root     root    11735  7월  13 22:13 file.c
-rw-r--r--  1 root     root    48946  7월  13 22:13 inode.c
-rw-r--r--  1 root     root     7811  7월  13 22:13 misc.c
-rw-r--r--  1 root     root       87  7월  13 22:13 modules.order
-rw-r--r--  1 root     root    17034  7월   9 22:47 namei_msdos.c
-rw-r--r--  1 root     root    26141  7월  13 16:26 namei_vfat.c
-rw-r--r--  1 root     root     8016  7월   9 22:47 nfs.c
-rw-r--r--  1 root     root    49017  7월   9 22:47 tags
```

< after compile >

```
-rw-r--r--  1 root     root     4164  7월   9 22:47 Kconfig
-rw-r--r--  1 root     root      428  7월  13 16:35 Makefile
-rw-r--r--  1 root     root        0  7월  13 22:13 Module.symvers
-rw-r--r--  1 root     root     8767  7월   9 22:47 cache.c
-rw-r--r--  1 root     root   140816  7월  14 15:42 cache.o
-rw-r--r--  1 root     root    35629  7월  13 22:12 dir.c
-rw-r--r--  1 root     root   254288  7월  14 15:42 dir.o
-rw-r--r--  1 root     root    14228  7월   9 22:47 fat.h
-rw-r--r--  1 root     root  1523862  7월  14 15:42 fat.ko
-rw-r--r--  1 root     root     7730  7월  14 15:42 fat.mod.c
-rw-r--r--  1 root     root    74152  7월  14 15:42 fat.mod.o
-rw-r--r--  1 root     root  1451034  7월  14 15:42 fat.o
-rw-r--r--  1 root     root    16942  7월  13 22:12 fatent.c
-rw-r--r--  1 root     root   247656  7월  14 15:42 fatent.o
-rw-r--r--  1 root     root    11735  7월  13 22:13 file.c
-rw-r--r--  1 root     root   226064  7월  14 15:42 file.o
-rw-r--r--  1 root     root    48946  7월  13 22:13 inode.c
-rw-r--r--  1 root     root   337120  7월  14 15:42 inode.o
-rw-r--r--  1 root     root     7811  7월  13 22:13 misc.c
-rw-r--r--  1 root     root   149368  7월  14 15:42 misc.o
-rw-r--r--  1 root     root       87  7월  14 15:42 modules.order
-rw-r--r--  1 root     root    17034  7월   9 22:47 namei_msdos.c
-rw-r--r--  1 root     root    26141  7월  13 16:26 namei_vfat.c
-rw-r--r--  1 root     root   206200  7월  14 15:42 namei_vfat.o
-rw-r--r--  1 root     root     8016  7월   9 22:47 nfs.c
-rw-r--r--  1 root     root   131920  7월  14 15:42 nfs.o
-rw-r--r--  1 root     root    49017  7월   9 22:47 tags
-rw-r--r--  1 root     root   271653  7월  14 15:42 vfat_lecture.ko
-rw-r--r--  1 root     root     3607  7월  14 15:42 vfat_lecture.mod.c
-rw-r--r--  1 root     root    69456  7월  14 15:42 vfat_lecture.mod.o
-rw-r--r--  1 root     root   203574  7월  14 15:42 vfat_lecture.o
```

실습 환경 구축 – FAT file system mount

- FAT file system mount
  - ✓ /home/fs-lecture/lecture/fat_module 위치의 FAT file system 모듈 디렉토리로 이동한다.
    - "# cd /home/fs-lecture/lecture/fat_module"
  - ✓ 컴파일한다
    - "# make"
  - ✓ FAT file system모듈을 적재 시킨다.
    - "# insmod fat.ko",
    - "# insmod vfat_lecture.ko"

```
Module                  Size  Used by
vfat_lecture           17411  0
fat                    65951  1 vfat_lecture
rxcache                18168  0
rxdsk                  17555  0
nls_iso8859_1          12713  0
bnep                   19625  2
rfcomm                 69563  0
```

  - ✓ 적재된 FAT file system 모듈을 확인한다.
    - "# lsmod | grep fat"
  - ✓ /dev/rxd0의 ram disk를 FAT file system의 형태로 포맷시켜준다.
    - "# mkfs.vfat /dev/rxd0"
      - Unable to get drive geometry … 는 error 가 아닙니다.
  - ✓ 추후 본 실습 시 모듈로 적재한 ram disk 를 mount 한다.

실습 환경 구축 – FAT file system mount

- FAT file system mount
  - ✓ file system이 제대로 mount되었는지 확인은 "# df –Th", "# mount" 의 다음 두 가지 명령어를 이용할 수 있다.

```
Filesystem      Type          Size  Used Avail Use% Mounted on
/dev/sda1       ext4          222G   87G  124G  42% /
udev            devtmpfs      3.8G  4.0K  3.8G   1% /dev
tmpfs           tmpfs         774M  832K  773M   1% /run
none            tmpfs         5.0M     0  5.0M   0% /run/lock
none            tmpfs         3.8G  144K  3.8G   1% /run/shm
/dev/rxd0       vfat_lecture   10G  8.0K   10G   1% /mnt/fat_test
```

```
none on /run/shm type tmpfs (rw,nosuid,nodev)
gvfs-fuse-daemon on /var/lib/lightdm/.gvfs type fuse.gvfs-fuse-daemon (rw,nosuid,nodev,user=lightdm)
/dev/rxd0 on /mnt/test type vfat_lecture (rw)
```

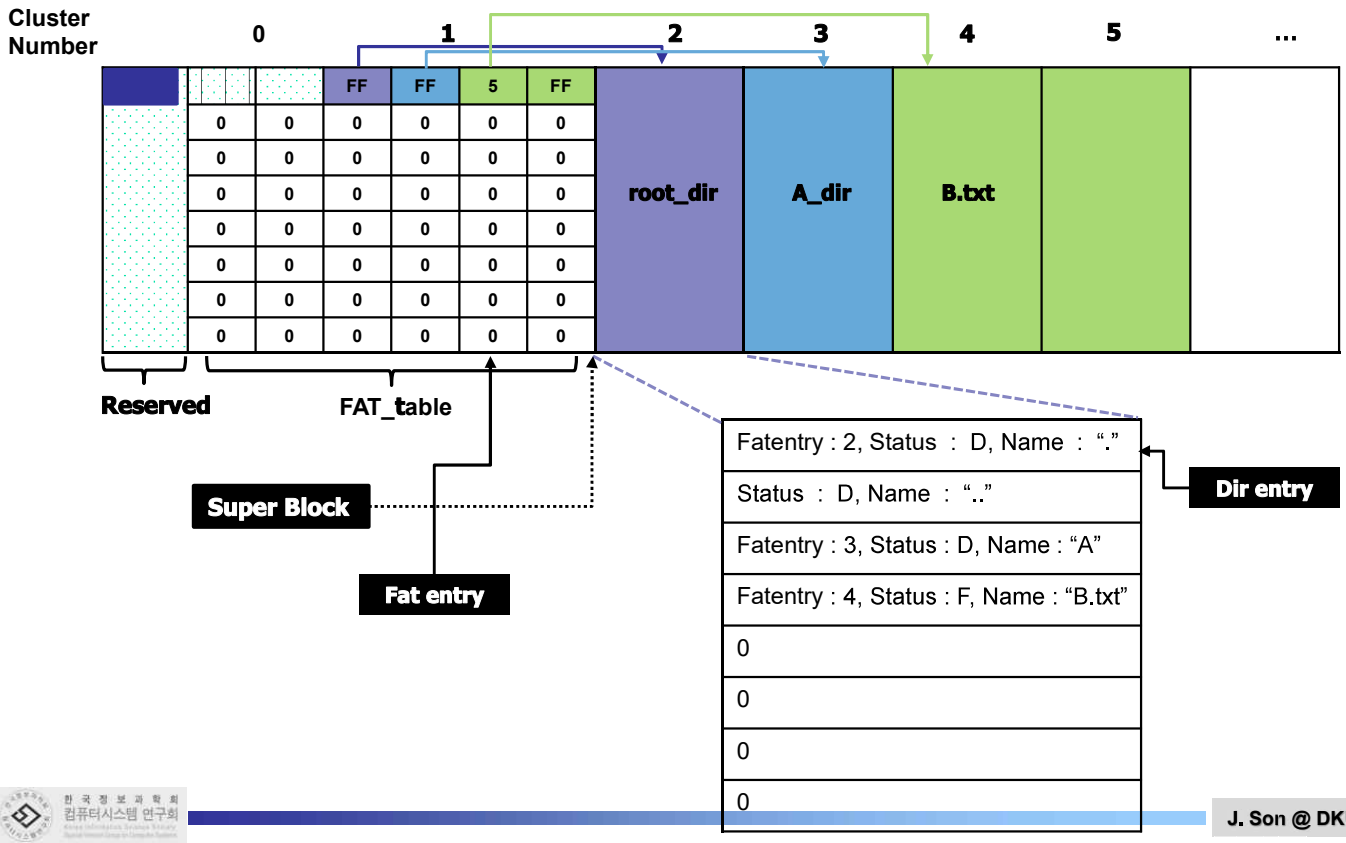
  - ✓ 실습이 끝난 후 mount를 해제 시키기 위해"# umount 마운트위치"를 입력한다.
  - ✓ "# rapiddisk --detach rxd0"를 통해 ram disk를 해제 시켜주어야 한다.

FAT file system 실습 1

FAT Recovery

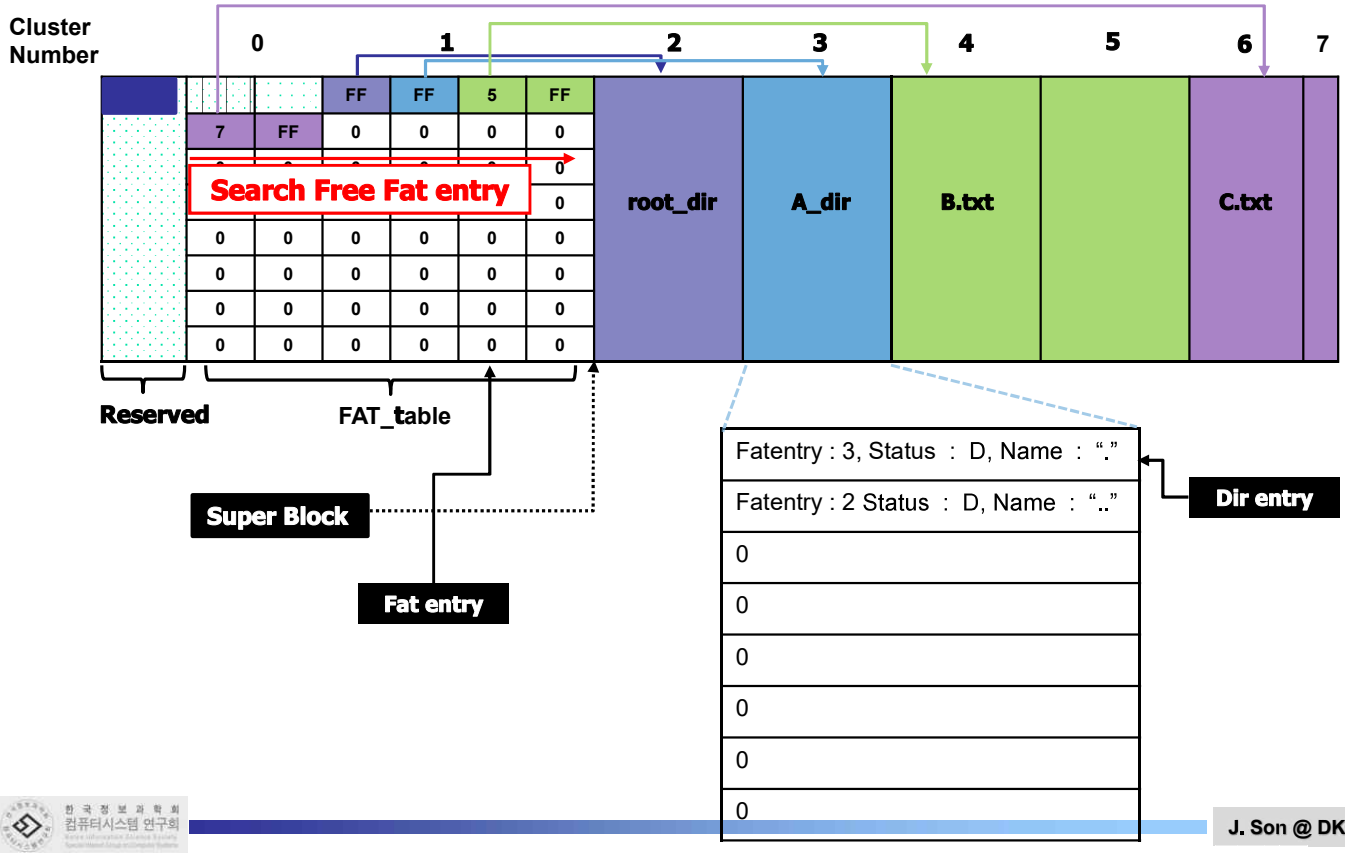
FAT file system Recovery실습

- FAT file system Recovery 시나리오 - Root directory



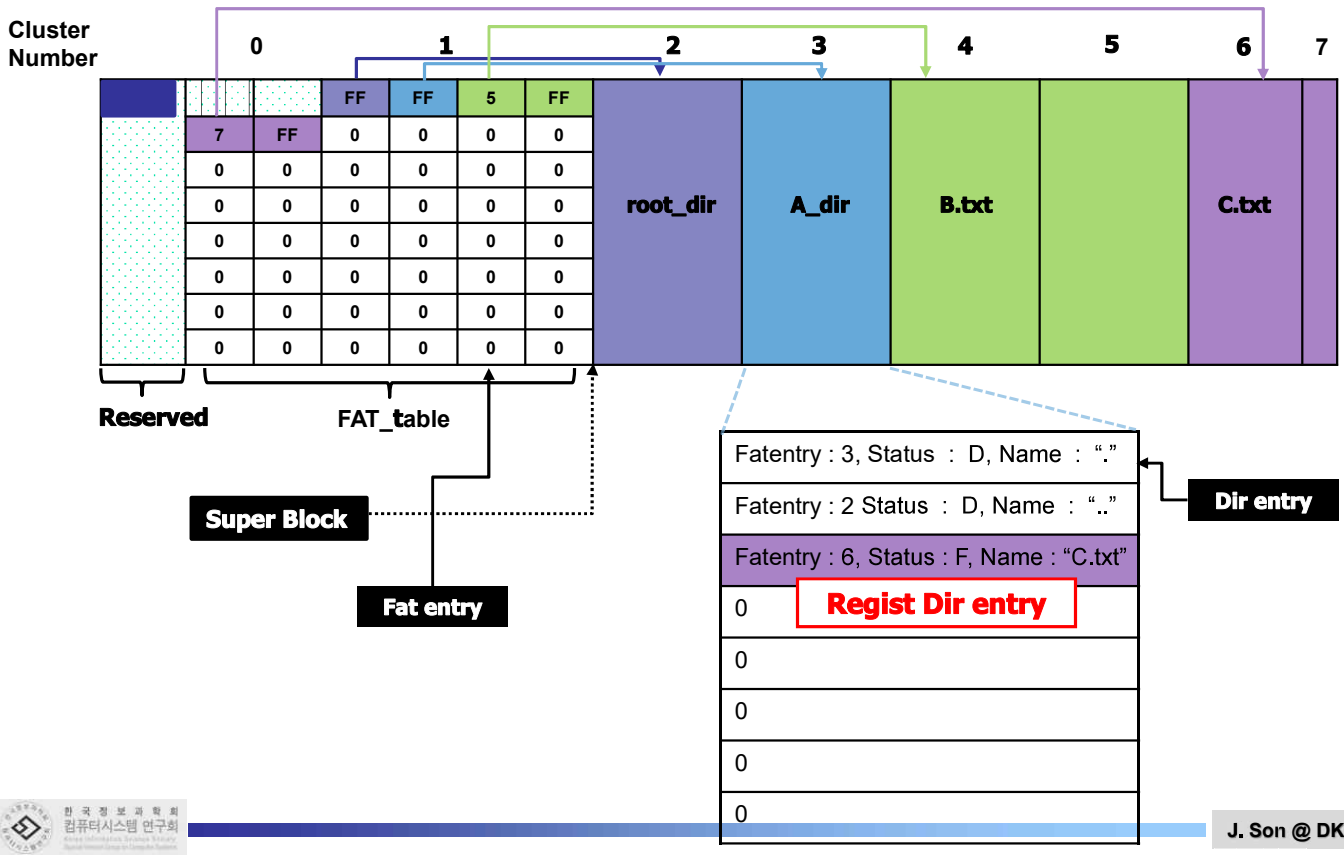
FAT file system Recovery실습

- FAT file system Recovery 시나리오 - A_dir에 C.txt를 생성할 때



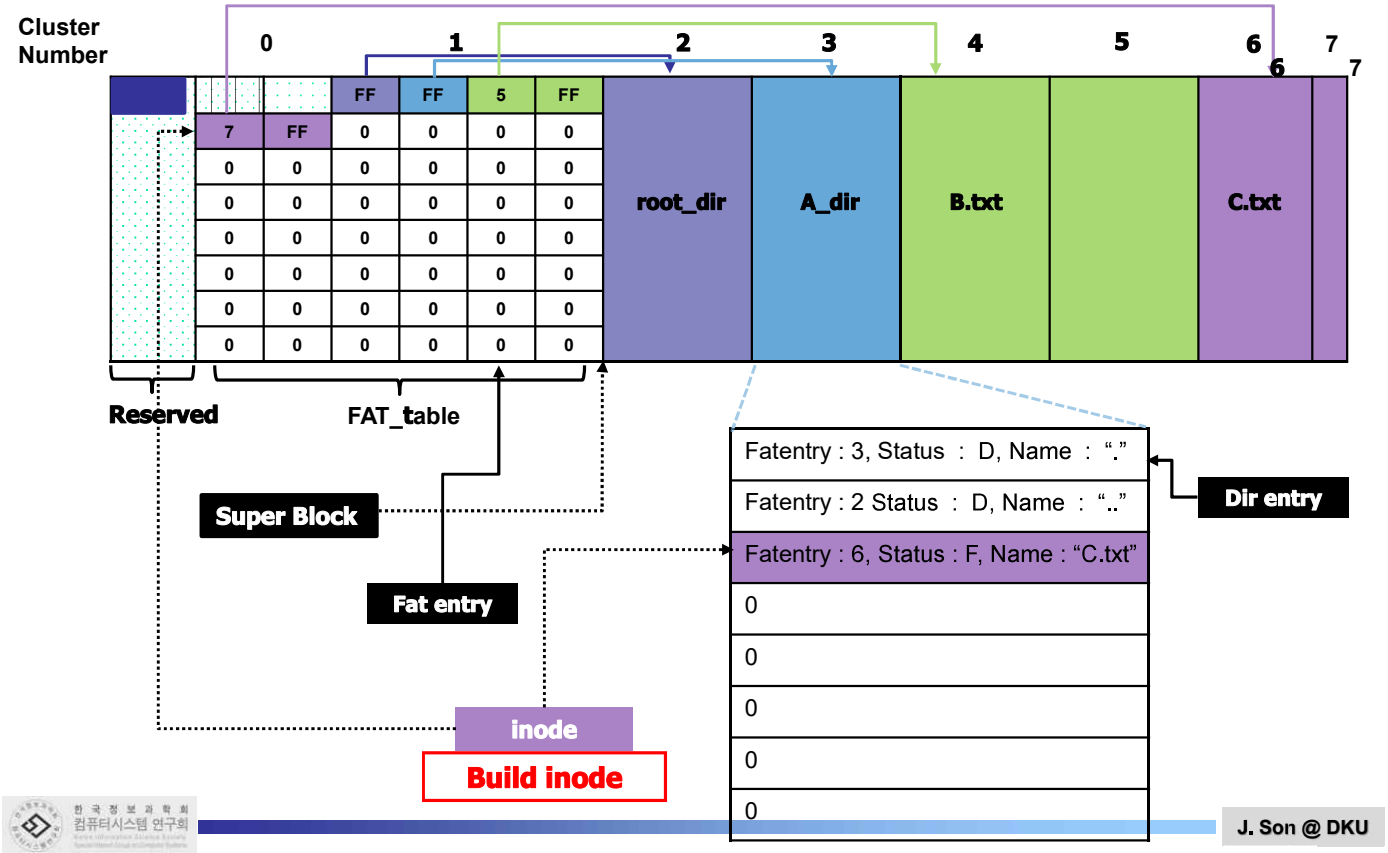
FAT file system Recovery실습

- FAT file system Recovery 시나리오 - A_dir에 C.txt를 생성할 때



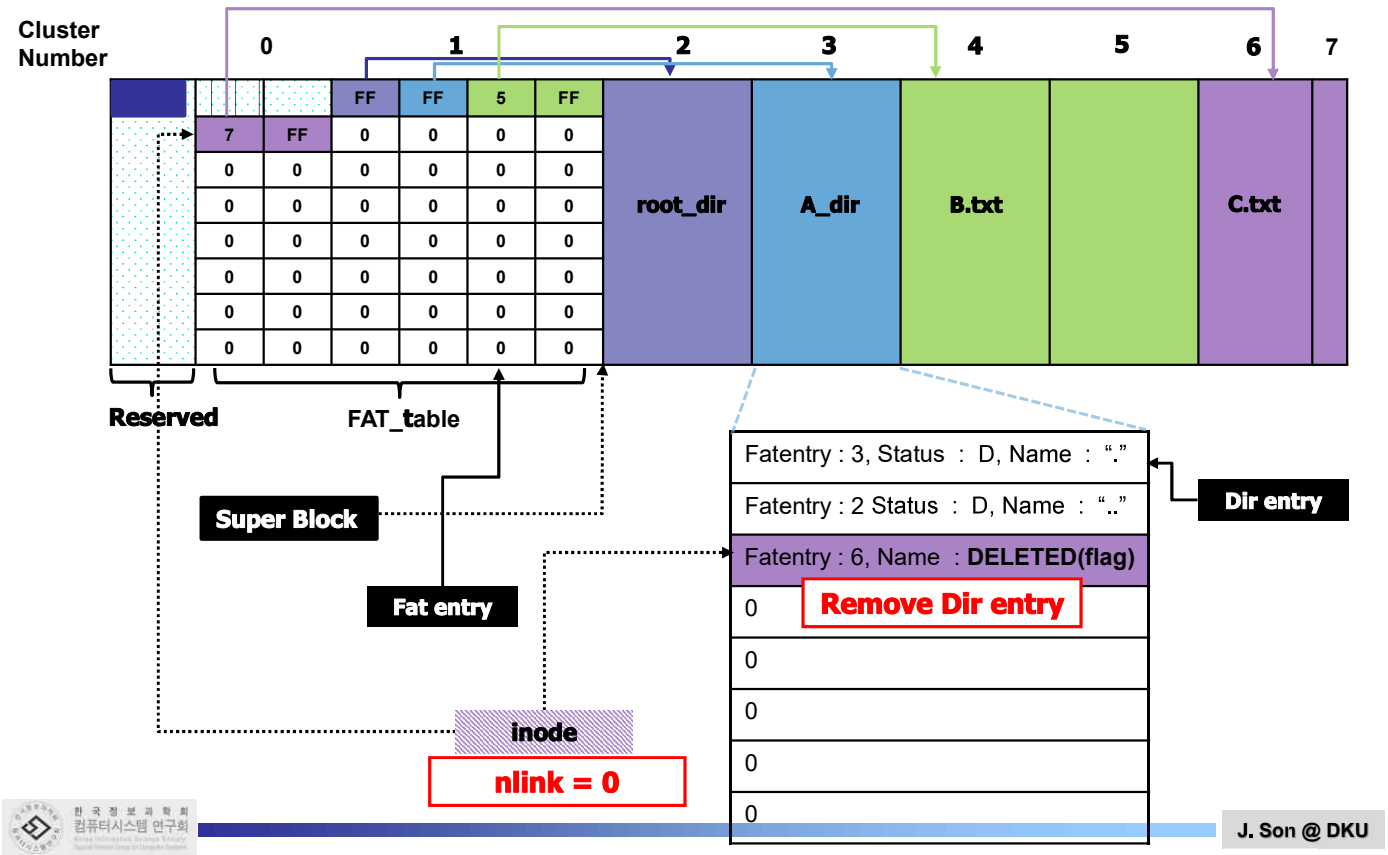
FAT file system Recovery실습

- FAT file system Recovery 시나리오 - A_dir에 C.txt를 생성할 때



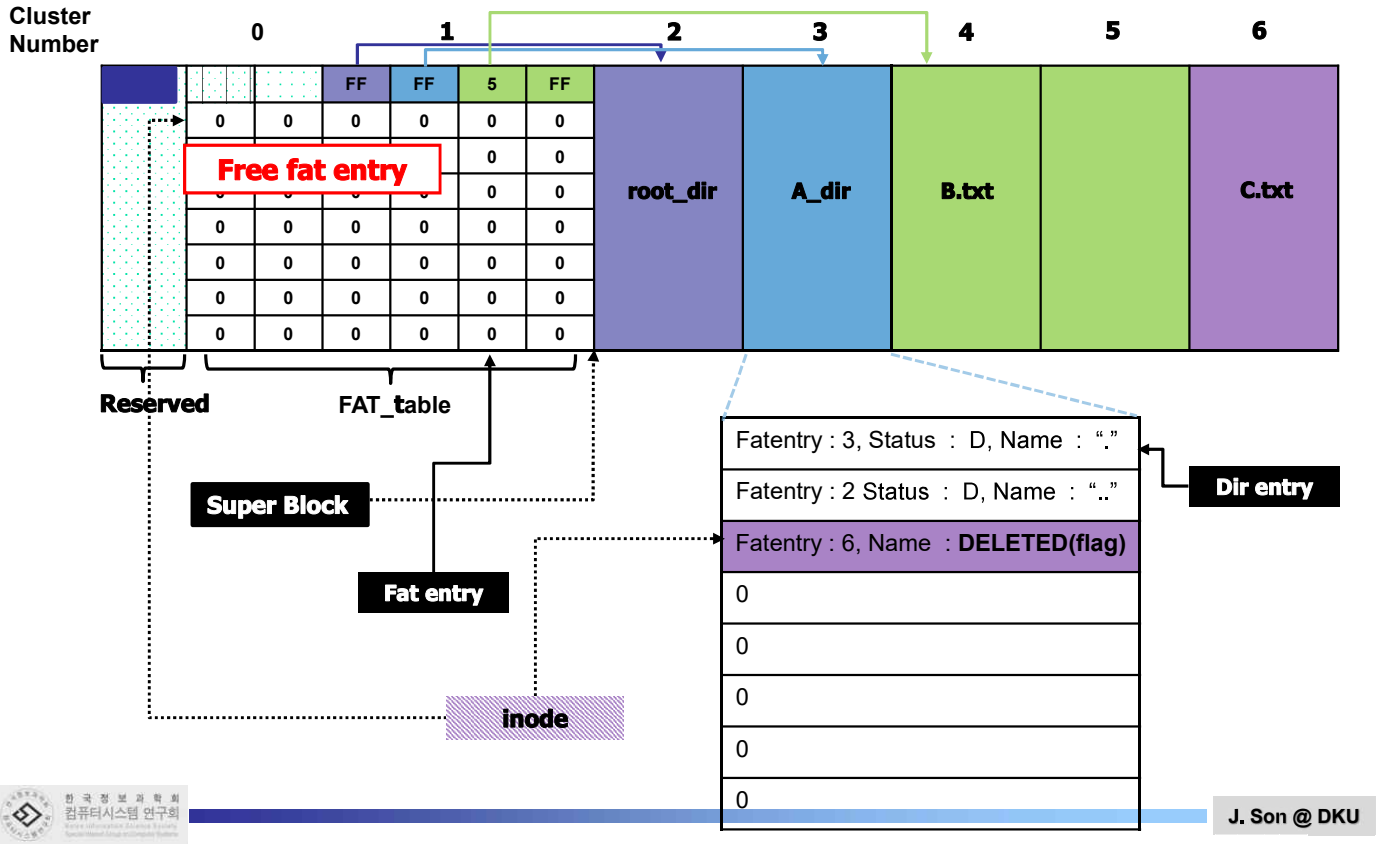
FAT file system Recovery실습

- FAT file system Recovery 시나리오 - A_dir에 C.txt를 삭제할 때



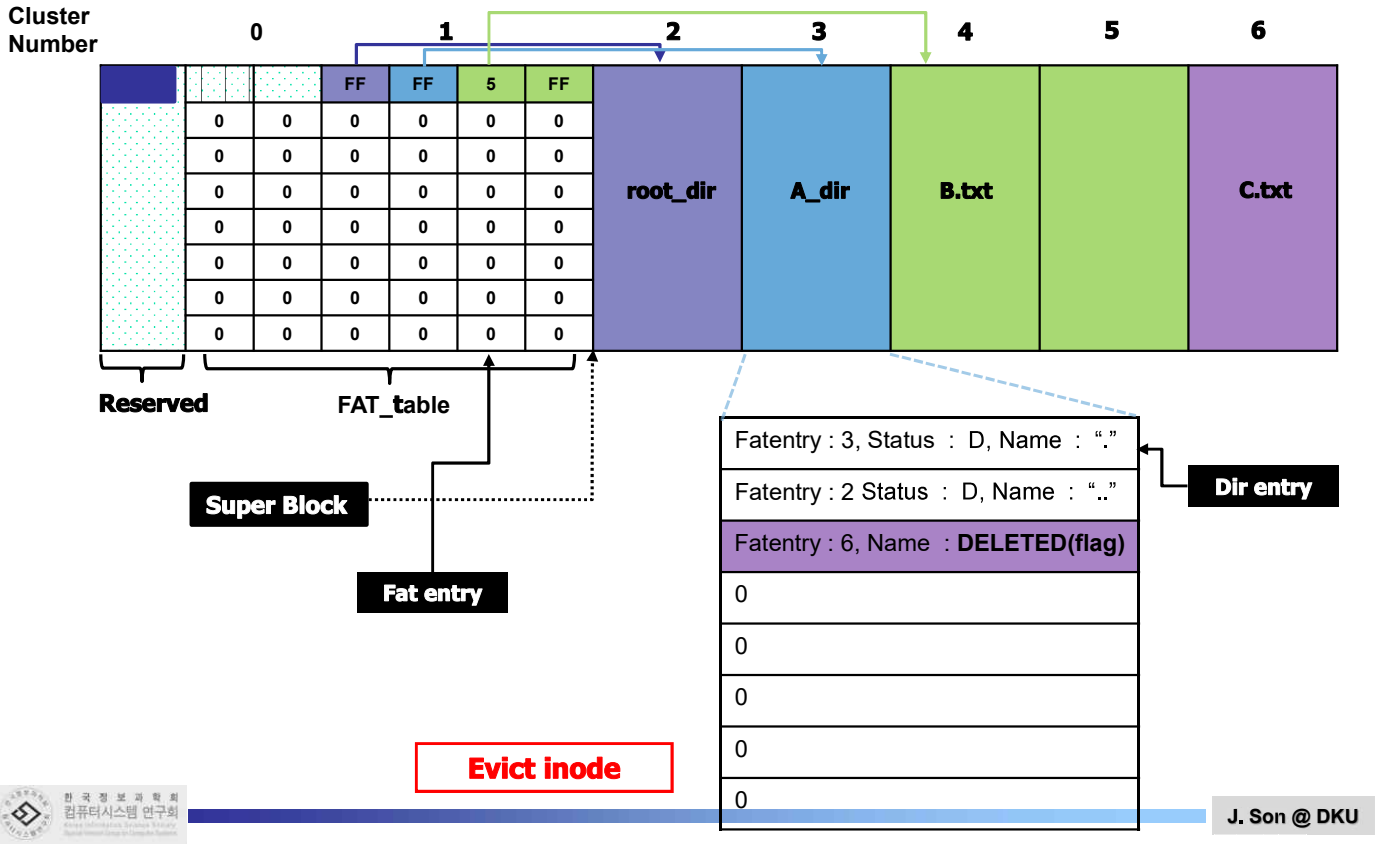
FAT file system Recovery실습

- FAT file system Recovery 시나리오 - A_dir에 C.txt를 삭제할 때



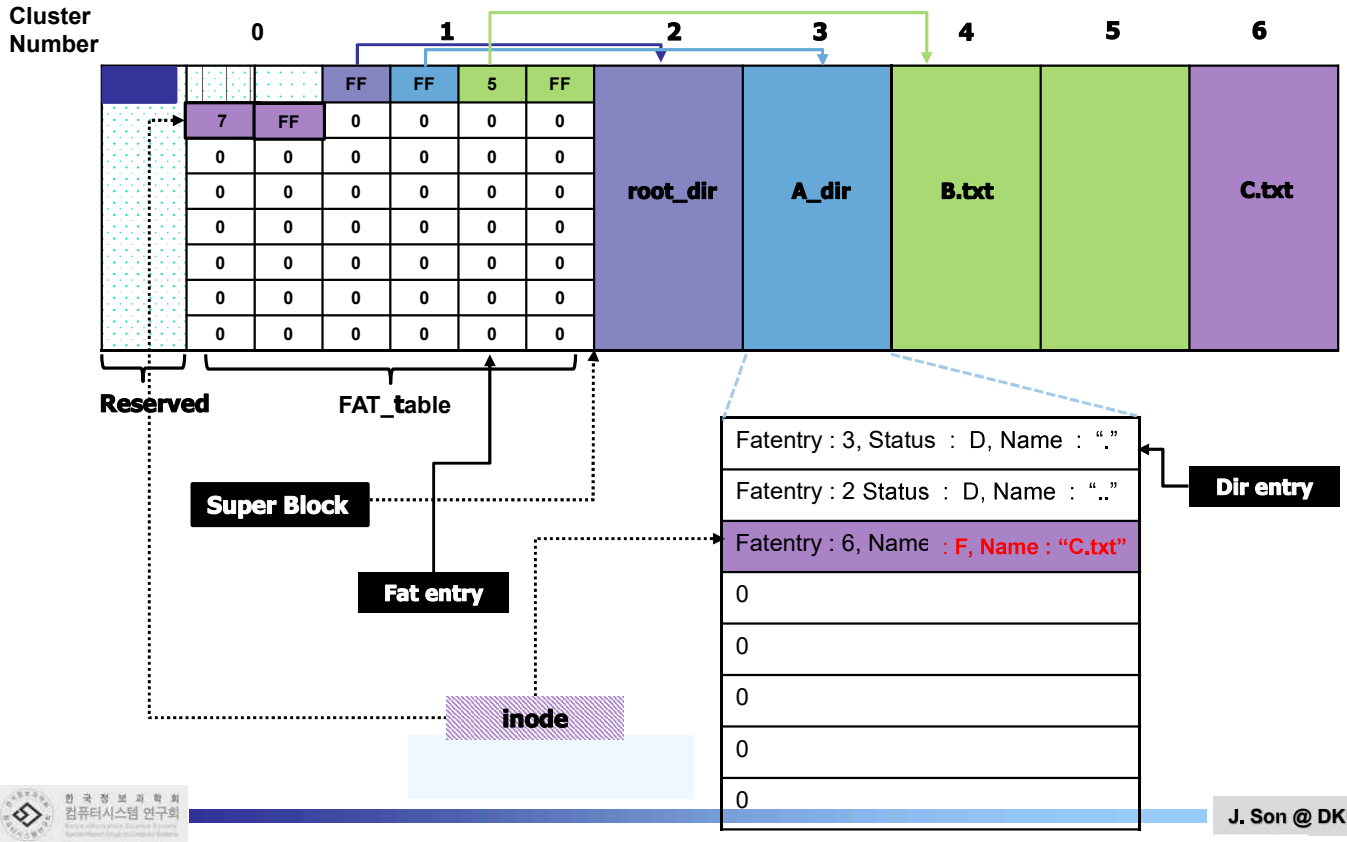
FAT file system Recovery실습

- FAT file system Recovery 시나리오 - A_dir에 C.txt를 삭제할 때



FAT file system Recovery실습

- FAT file system Recovery 시나리오 - Recovery



FAT file system Recovery 실습 참고 자료

- 실습 결과
  - ✓ 참고자료 위치 : /home/fs-lecture/lecture/practice/fat_recovery_lecture/
    - 참고자료 위치로 이동한다. — cd 참고자료 위치
    - 소스파일을 컴파일한다. — "# make"
    - ramdisk 가 attach된 상태에서 A,B,C 파일 생성한다. — "#./0_start.sh"
    - disk directory 에 생성된 A,B,C 파일 확인한다. — "# ls –al disk"
    - 을 통해 disk 디렉터리 내의 파일을 삭제한 한다 — "# rm –rf disk/A"
    - 삭제 후 디렉터리를 확인한다 — "# ls –al disk"
    - 파일 이름을 인자로 하여 삭제된 파일을 복구한다 — "# ./fat_recovery A"
    - 복구 후 디렉터리를 확인한다 — "# ls –al disk"

```
[root@localhost recovery_final]# ls -l
합계 28
-rwxrwxrwx 1 root root  254  8월 10 23:28 0_start.sh
-rwxrwxrwx 1 root root   21  8월 10 00:38 1_exit.sh
-rwxrwxrwx 1 root root  266  8월 10 23:28 2_restart.sh
-rw-r--r-- 1 root root 9229  8월 10 23:29 3_recovery_src.c
drwxr-xr-x 2 root root 4096  8월 10 00:35 disk
[root@localhost recovery_final]# gcc 3_recovery_src.c -o recovery.out
[root@localhost recovery_final]# ls -l
합계 44
-rwxrwxrwx 1 root root   254  8월 10 23:28 0_start.sh
-rwxrwxrwx 1 root root    21  8월 10 00:38 1_exit.sh
-rwxrwxrwx 1 root root   266  8월 10 23:28 2_restart.sh
-rw-r--r-- 1 root root  9229  8월 10 23:29 3_recovery_src.c
drwxr-xr-x 2 root root  4096  8월 10 00:35 disk
-rwxr-xr-x 1 root root 13514  8월 10 23:30 recovery.out
[root@localhost recovery_final]# ./0_start.sh
mkfs.fat 3.0.27 (2014-11-12)
unable to get drive geometry, using default 255/63
Filesystem      Type            Size  Used Avail Use% Mounted on
devtmpfs        devtmpfs        3.9G     0  3.9G   0% /dev
tmpfs           tmpfs           3.9G  652K  3.9G   1% /dev/shm
tmpfs           tmpfs           3.9G  1.1M  3.9G   1% /run
tmpfs           tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/sda3       ext4             50G   18G   29G  39% /
tmpfs           tmpfs           3.9G  712K  3.9G   1% /tmp
/dev/sda5       ext4            860G   14G  803G   2% /home
/dev/sda1       ext4            477M  217M  231M  49% /boot
tmpfs           tmpfs           799M   16K  799M   1% /run/user/1000
/dev/rxd0       vfat_lecture   1022M   16K 1022M   1% /home/hoon/lecture/recovery_final/disk
[root@localhost recovery_final]# ls -l disk
합계 12
-rwxr-xr-x 1 root root 12  8월 10 23:30 A
-rwxr-xr-x 1 root root 12  8월 10 23:30 B
-rwxr-xr-x 1 root root 12  8월 10 23:30 C
[root@localhost recovery_final]# cat disk/*
A_FILE_DATA
B_FILE_DATA
C_FILE_DATA
[root@localhost recovery_final]# rm -rf disk/*
[root@localhost recovery_final]# ls -l disk
합계 0
[root@localhost recovery_final]#
```

/dev/rxd0를 fat filesystem으로 포맷, disk 디렉토리에 mount

A, B, C 파일 생성

A파일 데이터

B파일 데이터

C파일 데이터

삭제 전 disk 디렉토리

삭제 후 disk 디렉토리

FAT file system Recovery실습 참조 자료

```
[root@localhost recovery_final]# ./recovery.out A
-DELETED_msdos_dir_entry-:
de->name:A          | de->attr:32 | de->start_hi:0 | de->start:3 | de->size:c |
START RECOVERY
success
[root@localhost recovery_final]# ./recovery.out B
-DELETED_msdos_dir_entry-:
de->name:B          | de->attr:32 | de->start_hi:0 | de->start:4 | de->size:c |
START RECOVERY
success
[root@localhost recovery_final]# ./recovery.out C
-DELETED_msdos_dir_entry-:
de->name:C          | de->attr:32 | de->start_hi:0 | de->start:5 | de->size:c |
START RECOVERY
success
[root@localhost recovery_final]# ls -l disk
합계 12
-rwxr-xr-x 1 root root 12  8월 10 23:30 A
-rwxr-xr-x 1 root root 12  8월 10 23:30 B
-rwxr-xr-x 1 root root 12  8월 10 23:30 C
[root@localhost recovery_final]# cat disk/*
A_FILE_DATA
B_FILE_DATA
C_FILE_DATA
[root@localhost recovery_final]#
```

A파일 복구

복구 후 disk 디렉토리

- ✓ 실습이 끝난 후
  - " #./1_exit.sh" 수행하여 umount 한다.
  - "# rmmod vfat_lecture"
    "# rmmod fat" 를 수행하여 fat모듈을 내린다.

FAT file system Recovery실습

- 삭제된 파일 복구 실습
  - ✓ 목표
    - 간단하게 User Application에서 disk를 읽어 삭제된 파일을 복구한다.
  - ✓ 참조 소스코드
    - /home/fs-lecture/lecture/practice/fat_recovery_lecture/fat_recovery.c
  - ✓ 실습 위치
    - 소스코드 작성
      - /home/fs-lecture/lecture/practice/fat_recovery_practice/
        - fat_info.h : fat 의 Boot Sector 정보
    - 복구할 파일 생성 위치
      - /home/fs-lecture/lecture/practice/fat_recovery_practice/disk
      - 소스코드 작성 후,
        - "# mount –t vfat_lecture ./disk"
          를 수행하여 disk 디렉터리를 마운트 시킨 후, 파일을 생성하고, 작성한 프로그램을 통해 복구해 본다.

FAT file system Recovery실습

- 삭제된 파일 복구 실습
  - ✓ 실습 내용
    1. FAT File System Off-Disk structure정보들을 받아오기 위해 부트레코드를 읽어 들여 fat_info.h 을 통해 필요한 정보들을 가져와 초기화 한다.
    2. 초기화한 정보들을 기반으로 파일을 생성한 위치(실습에선 편의를 위해 루트 디렉터리에 생성한다.)의 msdos_dir_entry 를 읽어 들인다.
    3. 읽어 들인 msdos_dir_entry의 정보를 통해 해당 파일이 삭제되었는지 검사 후 삭제되었다면 파일 이름의 첫 글자 부분인 파일 상태 플래그를 수정하여 파일을 복구한다.
  - ✓ 관련 함수
    - fseek – 읽어 들일 위치 설정하는 함수

```
int fseek(FILE *stream, long offset, int whence);
```

    - fwrite – 파일에 버퍼의 내용을 쓰는 함수

```
size_t fread(void *ptr, size_t size, size_t nmembFILE *" stream );
```

    - fread – 파일로부터 지정한 개수만큼 자료를 읽는 함수

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb,
FILE *stream);
```

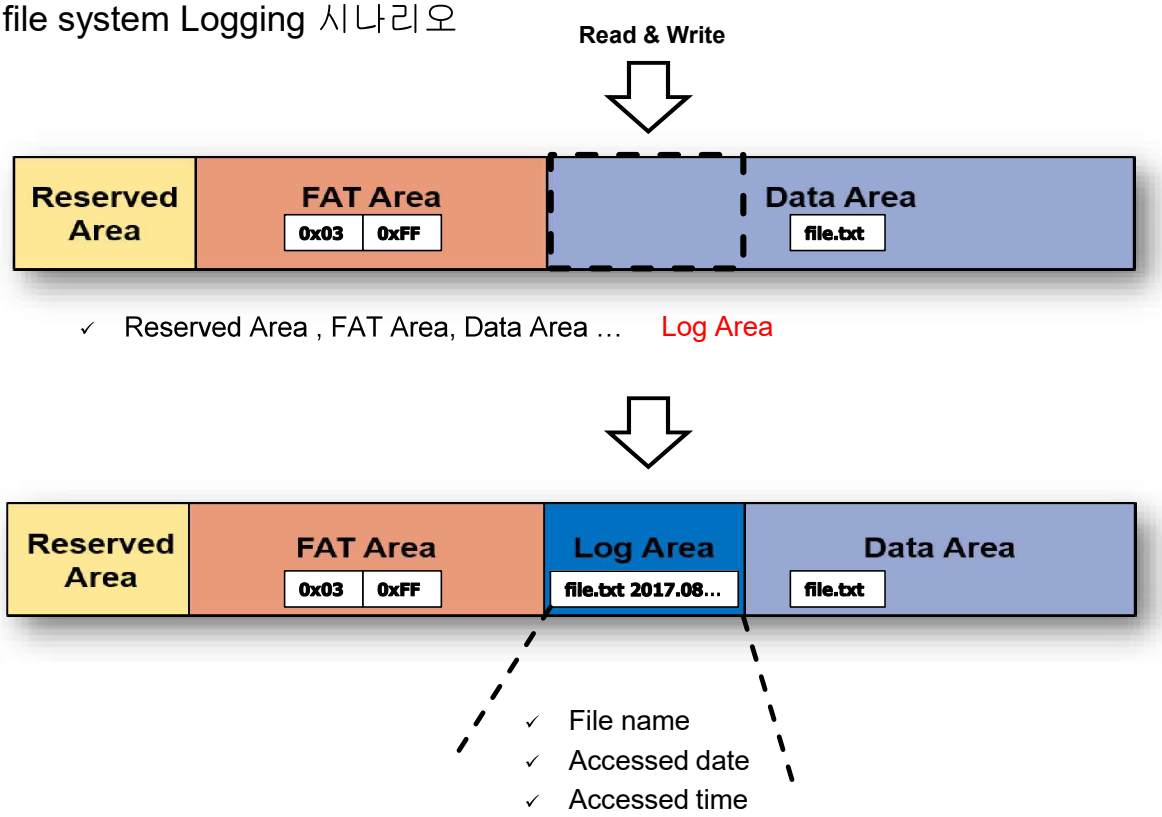
  - ✓ 실습이 끝난 후
    - "# umount ./disk" 를 통해 disk에 mount 를 해제 시킨다.
    - "# rmmod vfat_lecture"
    - "# rmmod fat.ko" 를 수행하여 fat모듈을 내린다.

FAT file system 실습 2

FAT logging

FAT file sytem Logging 실습

- FAT file system Logging 시나리오



✓ Reserved Area , FAT Area, Data Area … Log Area

✓ File name
✓ Accessed date
✓ Accessed time

FAT file system Logging 실습

- 파일시스템의 이름변경 및 컴파일 준비
  - ✓ 실습 위치로 이동
    - "# cd /home/fs-lecture/lecture/practice/fat_log_module_lecture/"
  - ✓ File System의 Mount 관련 수정
    - Superblock의 초기화는 file system이 mount되는 시점에 수행된다.
    - Mount 시 커널에 알려질 File System의 이름을 설정한다.

수정 위치 : 실습 위치/namei_vfat.c – vfat_fs_type (line : 1069)

```
static struct file_system_type vfat_fs_type = {
    .owner      = THIS_MODULE,
    .name       = "vfat_log_lecture",
    .mount      = vfat_mount,
    .kill_sb    = kill_block_super,
    .fs_flags   = FS_REQUIRES_DEV,
};
MODULE_ALIAS_FS("vfat_log_lecture");
```

파일 시스템 이름 수정

  - ✓ Makefile 관련 수정
    - 소스 파일을 추가했을 경우 Makefile 을 수정하여 컴파일 될 수 있도록 한다.

수정 위치 : 실습 위치/ /Makefile

```
KERNELDIR=/lib/modules/$(shell uname -r)/build

NAME=vfat_log_lecture
obj-m += fat.o
obj-m += $(NAME).o
$(NAME)-objs := vfat.o

fat-y := cache.o dir.o fatent.o file.o inode.o misc.o nfs.o fat_log.o
$(NAME)-objs := namei_vfat.o
```

모듈 이름 수정

컴파일할 소스코드 추가

FAT file system Logging 실습

- log될 자료구조 생성 및 log영역의 확보
  - ✓ fat.h 에는 Off-Disk 자료구조들이 들어있다. 이곳에 로그 자료구조를 추가한다.
  - ✓ fat_fill_super 함수에서는 disk의 superblock을 읽어 들여서 superblock, root directory 등의 정보를 초기화 하고 데이터가 기록될 영역을 구분한다.
  - ✓ fa_fill_super의 영역 구분을 수정하여 log영역의 크기를 적당히 정한다(실습 : FAT 테이블 크기) 또한 생성한 로그자료구조를 초기화 한다.

수정 위치 : 실습 위치/ fat.h – msdos_sb_info (line : 65)

```
struct msdos_sb_info {
    unsigned short sec_per_clus;  /* sectors/cluster */
    unsigned short cluster_bits;  /* log2(cluster_size) */
    unsigned int cluster_size;    /* cluster size */
    unsigned char fats, fat_bits; /* number of FATs, FAT bits (12 or 16) */
    unsigned short fat_start;
    unsigned long fat_length;     /* FAT start & length (sec.) */
    unsigned long dir_start;
    unsigned short dir_entries;   /* root dir start & entries */
    unsigned long data_start;     /* first data sector */

                                  ...

    /* Linux Kernel Camp 2016-08-25
     *added for fat log lecture area
     */
    unsigned long log_start;
    unsigned long log_pos;
    unsigned long log_max_pos;
    unsigned long log_count;
    /***********************************/
};
```

log 영역 관련 필드 추가

```
/* Linux Kernel Camp 2016-08-25
 * added for log entry
 */
struct fat_log_entry
{
    __u8    name[MSDOS_NAME];   /* file name */
    __le16  date;               /* log time */
    __le16  time;               /* log date */
};
```

log될 정보 추가

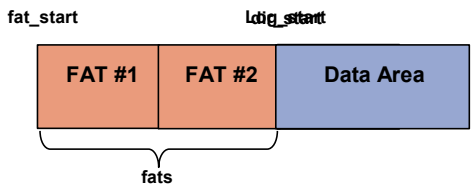
```
sbi->dir_per_block = sb->s_blocksize / sizeof(struct msdos_dir_entry);
sbi->dir_per_block_bits = ffs(sbi->dir_per_block) - 1;
sbi->dir_start = sbi->fat_start + sbi->fats * sbi->fat_length;
sbi->dir_entries = bpb.fat_dir_entries;
```

기존 소스코드

수정 위치 : 실습 위치/inode.c – fat_fill_super (line 1654)

```
sbi->dir_per_block_bits = ffs(sbi->dir_per_block) - 1;

/* Linux Kernel Camp 2016-08-25
 * make log area
 */
sbi->log_start = sbi->fat_start + sbi->fats * sbi->fat_length;
sbi->dir_start = sbi->log_start + sbi->fat_length;
sbi->log_pos = sbi->log_start*logical_sector_size;
sbi->log_max_pos = sbi->dir_start * logical_sector_size;
sbi->log_count = 0;

sbi->dir_entries = bpb.fat_dir_entries;
```

변경된 소스코드



FAT file system Logging 실습

- 파일의 read, write routine 에 logging mechanism 도입

수정 위치 : 실습 위치/ inode.c – fat_get_block (line : 178)

```
static const struct address_space_operations fat_aops = {
    .readpage   = fat_readpage,
    .readpages  = fat_readpages,
    .writepage  = fat_writepage,
    .writepages = fat_writepages,
    .write_begin  = fat_write_begin,
    .write_end  = fat_write_end,
    .direct_IO  = fat_direct_IO,
```

```
static int fat_write_begin(struct file *file, struct address_space *mapping,
            loff_t pos, unsigned len, unsigned flags,
            struct page **pagep, void **fsdata)
{
    int err;

    printk("fat_write_begin \n");
    *pagep = NULL;
    err = cont_write_begin(file, mapping, pos, len, flags,
                pagep, fsdata, fat_get_block,
                &MSDOS_I(mapping->host)->mmu_private);
    if (err < 0)
```

```
static int fat_get_block(struct inode *inode, sector_t iblock,
            struct buffer_head *bh_result, int create)
{
    struct super_block *sb = inode->i_sb;
    unsigned long max_blocks = bh_result->b_size >> inode->i_blkbits;
    int err;

    err = __fat_get_block(inode, iblock, &max_blocks, bh_result, create);
    if (err)
        return err;
    bh_result->b_size = max_blocks << sb->s_blocksize_bits;
    if(iblock == 0)
        log_read_write(sb,inode);
    return 0;
}
```

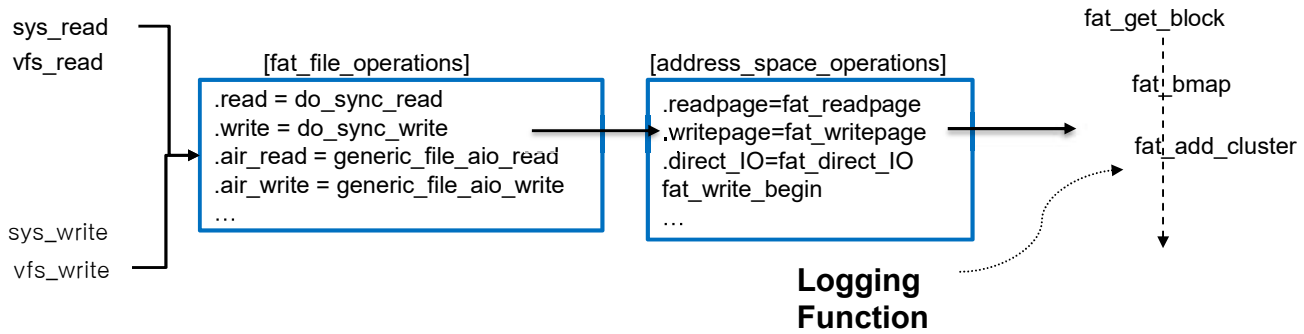
- ✓ FAT 은 파일의 생성 파일을 page cache 에 mapping 하고 read, write 를 수행하기 위해 address_space_operations 인 fat_aops 를 제공한다.
- ✓ File System은 read,write 수행시 block을 요청할 때 각 파일시스템 별로 상이한 가용 block을 구하는 mechanism을 get_block_t 라는 함수포인터를 통해 제공하여 각 File System 마다 그에 맞는 함수를 수행하도록 한다.

```
typedef int (get_block_t)(struct inode *inode, sector_t iblock,
            struct buffer_head *bh_result, int create);
```

- ✓ FAT 은 이를 위해 fat_get_block 함수를 제공하여 write를 위해 cluster 할당 및 cluster chain을 형성하고 read를 위해 cluster 를 읽어 들여 page와 buffer_head를 연결한다.
- ✓ fat_get_block 함수에 logging mechanism을 추가한다.
- ✓ fat_get_block 은 block 이 요청될 때마다 수행되므로 logging이 다수 발생한다. 이를 방지하기 위해 파일에 대한 첫번째 요청 시(파일내의 첫번째 섹터 요청시)에만 logging 하도록 한다.

FAT file system Logging 실습

- 파일에 대한 접근 시 log 수행 함수 생성
  - ✓ Logging mechanism function 추가
    1) file의 이름 접근 시간, 접근 날짜 등의 정보를 알기 위해 msdos_dir_entry 가 필요하다
    2) inode 의 속성인 i_pos를 통해 해당 파일의 msdos_dir_entry 가 위치한 block number 와 offset 을 얻고 이를 통해 msdos_dir_entry를 구한다.
    3) 확보한 log 영역에 파일이름, 날짜, 시간 등의 정보를 기록한다.
    4) log영역의 현재 위치를 옮긴다.
  - ✓ 추가한 함수 등록
    - 수정 위치 : 실습 위치/ fat.h 에 extern 으로 명시



FAT file system Logging 실습

- 파일에 대한 로그 및 바이너리 값 확인 명령어
  - ✓ xxd
    - shell 상에서 바이너리(이진 값)의 hexdump를 보여주는 명령어이다.
    - boot record 의 값의 확인 및 disk등에 기록하고 확인에 사용할 수 있다.
    - 옵션
      - g : 바이너리 값을 인자로 넘긴 숫자 단위로 출력한다.
      - s : 바이너리 값을 인자로 넘긴 주소부터 출력한다.
      - l : 바이너리 값을 인자로 넘긴 값만큼 출력한다.
    - 예시
      - "# mkfs.vfat /dev/rxd0"
        - xxd 의 실습을 위해 ramdisk 를 fat으로 포맷한다.
      - "# xxd –g 4 –l 128 –s+0x4000 /dev/rxd0 "
        - FAT 영역의 시작점인 0x4000부터 4byte 단위로 128byte만큼을 출력한다.

```
root@fslecture-VirtualBox:/home/fs-lecture/lecture/practice/fat_log_module_lecture# xxd -g 4 -l 128 -s+0x4000 /dev/rxd0
0004000: f8ffff0f ffffff0f f8ffff0f 04000000  ................
0004010: 05000000 ffffff0f 00000000 00000000  ................
0004020: 00000000 00000000 00000000 00000000  ................
0004030: 00000000 00000000 00000000 00000000  ................
0004040: 00000000 00000000 00000000 00000000  ................
0004050: 00000000 00000000 00000000 00000000  ................
0004060: 00000000 00000000 00000000 00000000  ................
0004070: 00000000 00000000 00000000 00000000  ................
```

      - "# xxd –g 1 –l 128 –s+0x0 /dev/rxd0 "
        - Boot record 시작점인 0x0부터 1byte 단위로 128byte만큼을 출력한다.

```
root@fslecture-VirtualBox:/home/fs-lecture/lecture/practice/fat_log_module_lecture# xxd -g 1 -l 128 -s+0x0 /dev/rxd0
0000000: eb 58 90 6d 6b 64 6f 73 66 73 00 00 02 08 20 00  .X.mkdosfs.... .
0000010: 02 00 00 00 00 f8 00 00 3f 00 ff 00 00 00 00 00  ........?.......
0000020: 00 00 80 00 f8 1f 00 00 00 00 00 00 02 00 00 00  ................
0000030: 01 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000040: 00 01 29 e1 de 06 09 20 20 20 20 20 20 20 20 20  ..).... 
0000050: 20 20 46 41 54 33 32 20 20 20 0e 1f be 77 7c ac    FAT32   ...w|.
0000060: 22 c0 74 0b 56 b4 0e bb 07 00 cd 10 5e eb f0 32  ".t.V.......^..2
0000070: e4 cd 16 cd 19 eb fe 54 68 69 73 20 69 73 20 6e  .......This is n
```

FAT file system Logging 실습 참조자료

- 실습 결과
  - ✓ 참고자료 위치 : /home/fs-lecture/lecture/practice/fat_log_module_lecture/
    - 작성해 놓은script 를 수행한다.
      - "#./mount.sh"
    - 아무 파일이 없을 때 작성한 script를 통해 FAT1, FAT2, Log, Data 영역의 hex값을 확인한다.
      - "#./xxd.sh"

```
root@fslecture-VirtualBox:/home/fs-lecture/lecture/practice/fat_log_module_lecture# ./xxd.sh
fat1 area - 0x4000
0004000: f8ffff0f ffffff0f f8ffff0f 00000000  ................
0004010: 00000000 00000000 00000000 00000000  ................
0004020: 00000000 00000000 00000000 00000000  ................
0004030: 00000000 00000000 00000000 00000000  ................
0004040: 00000000 00000000 00000000 00000000  ................
0004050: 00000000 00000000 00000000 00000000  ................
0004060: 00000000 00000000 00000000 00000000  ................
0004070: 00000000 00000000 00000000 00000000  ................
fat2 area - 0x203000
0203000: f8ffff0f ffffff0f f8ffff0f 00000000  ................
0203010: 00000000 00000000 00000000 00000000  ................
0203020: 00000000 00000000 00000000 00000000  ................
0203030: 00000000 00000000 00000000 00000000  ................
0203040: 00000000 00000000 00000000 00000000  ................
0203050: 00000000 00000000 00000000 00000000  ................
0203060: 00000000 00000000 00000000 00000000  ................
0203070: 00000000 00000000 00000000 00000000  ................
log area - 0x402000
0402000: 00000000 00000000 00000000 00000000  ................
0402010: 00000000 00000000 00000000 00000000  ................
0402020: 00000000 00000000 00000000 00000000  ................
0402030: 00000000 00000000 00000000 00000000  ................
0402040: 00000000 00000000 00000000 00000000  ................
0402050: 00000000 00000000 00000000 00000000  ................
0402060: 00000000 00000000 00000000 00000000  ................
0402070: 00000000 00000000 00000000 00000000  ................
data area - 0x601000
0601000: 00000000 00000000 00000000 00000000  ................
0601010: 00000000 00000000 00000000 00000000  ................
0601020: 00000000 00000000 00000000 00000000  ................
0601030: 00000000 00000000 00000000 00000000  ................
0601040: 00000000 00000000 00000000 00000000  ................
0601050: 00000000 00000000 00000000 00000000  ................
0601060: 00000000 00000000 00000000 00000000  ................
0601070: 00000000 00000000 00000000 00000000  ................
```

FAT1, FAT2 영역

Log 영역

Data 영역

FAT file system Logging 실습 참조자료

- 실습 결과
  - ✓ 참고자료 위치 : /home/fs-lecture/lecture/practice/fat_log_module_lecture/
    - test director에 data를 write , read 수행 후 결과를 다시 xxd script 를 통해 확인해본다. (실제 쓰여지는 크기가 있는 데이터를 write, read 한다.)
      - 예시 : "# cp file.c test" , "# vim test/test.txt"
    - FAT1, FAT2, Log, Data 영역의 hex값을 확인하고 logging 되는 모습을 확인한다.
      - "#./xxd.sh"

```
root@fslecture-VirtualBox:/home/fs-lecture/lecture/practice/fat_log_module_lecture# ./xxd.sh
fat1 area - 0x4000
0004000: f8ffff0f ffffff0f f8ffff0f 04000000  ................
0004010: 05000000 ffffff0f 00000000 ffffff0f  ................
0004020: 00000000 00000000 00000000 00000000  ................
0004030: 00000000 00000000 00000000 00000000  ................
0004040: 00000000 00000000 00000000 00000000  ................
0004050: 00000000 00000000 00000000 00000000  ................
0004060: 00000000 00000000 00000000 00000000  ................
0004070: 00000000 00000000 00000000 00000000  ................
fat2 area - 0x203000
0203000: f8ffff0f ffffff0f f8ffff0f 04000000  ................
0203010: 05000000 ffffff0f 00000000 ffffff0f  ................
0203020: 00000000 00000000 00000000 00000000  ................
0203030: 00000000 00000000 00000000 00000000  ................
0203040: 00000000 00000000 00000000 00000000  ................
0203050: 00000000 00000000 00000000 00000000  ................
0203060: 00000000 00000000 00000000 00000000  ................
0203070: 00000000 00000000 00000000 00000000  ................
log area - 0x402000
0402000: 46494c45 20202020 4320206e 0f4945b8  FILE    C  n.IE.
0402010: 54455354 54587e31 53575070 0f494fb8  TESTTX~1SWPp.IO.
0402020: 54455354 20202020 54585470 0f4951b8  TEST    TXTp.IQ.
0402030: 46494c45 437e3120 53575070 0f4955b8  FILEC~1 SWPp.IU.
0402040: 00000000 00000000 00000000 00000000  ................
0402050: 00000000 00000000 00000000 00000000  ................
0402060: 00000000 00000000 00000000 00000000  ................
0402070: 00000000 00000000 00000000 00000000  ................
data area - 0x601000
0601000: 41660069 006c0065 002e000f 00856300  Af.i.l.e......c.
0601010: 0000ffff ffffffff ffff0000 ffffffff  ................
0601020: 46494c45 20202020 43202020 000045b8  FILE    C   ..E.
0601030: 0f490f49 000045b8 0f490300 d32d0000  .I.I..E..I...-..
0601040: e5412e00 66006900 6c00650f 00652e00  .A..f.i.l.e..e..
0601050: 63002e00 73007700 70000000 0000ffff  c...s.w.p.......
0601060: e546494c 45437e31 20535720 000055b8  .FILEC~1 SW ..U.
0601070: 0f490f49 000055b8 0f490000 00000000  .I.I..U..I......
```

FAT1, FAT2 영역

Log 영역

Data 영역

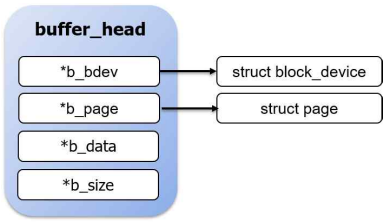
    - 작성한 스크립트를 수행하여 마운트를 해제한다.
      - "#./umount.sh"

FAT file system Logging 실습

- 파일 로깅 실습
  - ✓ 목표
    - FAT file system 을 파일에 대한 로그가 기록되는 본인만의 모듈을 생성한다.
  - ✓ 참조 소스코드
    - /home/fs-lecture/lecture/practice/fat_log_module_lecture/
      - 수정한 소스파일 : fat.h , inode.c, Makefile
      - 생성한 소스파일 : fat_log.c
      - 각 소스 파일의 수정된 부분을 참조
        - # grep –rn “Linux Kernel Camp” ./
  - ✓ 실습 위치
    - /home/fs-lecture/lecture/practice/fat_log_module_practice/

FAT file system Logging 실습

- 파일에 대한 접근 시 log 수행 함수 생성
  - ✓ 관련 자료구조
    - buffer_haed
      - physical sector에 대한 정보와 physical memory 를 mapping 하기 위한 정보를 유지하고 있다.
      - b_dev 에 block number 에 대한 정보를 저장한다. b_data에서 시작하여 b_size만큼의 physical memory 를 mapping 한다.



    - sb_bread
      - 해당 위치를 읽어들여 buffer_head를 반환한다.

```
static inline struct buffer_head *
sb_bread(struct super_block *sb, sector_t block)
{
    return __bread_gfp(sb->s_bdev, block, sb->s_blocksize, __GFP_MOVABLE);
}
```

    - brelse
      - buffer_head의 lock 을 해제한다.

```
static inline void brelse(struct buffer_head *bh)
{
    if (bh)
        __brelse(bh);
}
```

FAT file system Logging 실습

- 파일 로깅 실습
  - ✓ 실습 내용
    1. 파일시스템의 이름을 변경하고 컴파일을 위한 준비를 한다.
    2. disk에 logging 될 영역을 확보하고 로그 될 정보 형태를 만든다.
    3. 파일에 read, write 가 발생되는 부분에 logging mechanism 을 넣어 파일에 대한 접근이 있을 시 log를 남긴다.
  - ✓ 관련 자료구조 및 함수
    - file_system_type (namei_vfat.c , line : 1067)
      - 파일시스템 이름 변경
    - msdos_sb_info (fat.h , line : 65)
      - log 관련 필드 추가
    - extern 작성한 함수 이름 (fat.h , line : 429)
      - fat.h 에 작성한 함수 이름을 extern 선언
    - fat_fill_super (inode.c , line : 1644)
      - log 관련 필드 초기화
    - fat_get_block (inode.c , line : 171)
      - 작성한 logging 함수 도입
    - logging 함수 작성
      - msdos_dir_entry 를 읽어들여 log 영역에 기록
    - Makefile 수정
      - 소스코드 추가 시 컴파일 될 수 있도록 수정

FAT file system Logging 실습

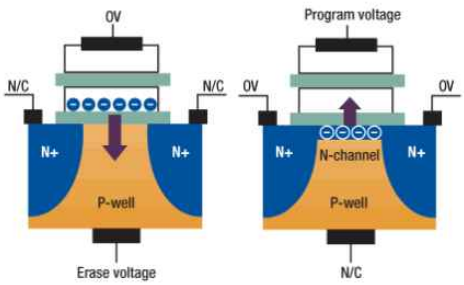
- 파일 로깅 실습
  - ✓ 소스 코드 작성 후
    - 작성한 소스코드를 컴파일 한다.
      - "# make"
    - 작성한 모듈을 올린다.
      - "# insmod fat.ko"
      - "# insmod 만든모듈이름.ko"
    - 작성한 파일시스템을 test 디렉터리에 마운트 한다.
      - "# mount –t 파일시스템이름 /dev/rxd0 ./test"
    - 파일을 쓰고 읽는다.
      - cp, vim 등을 사용하여 파일 읽고 쓰기
    - xxd 명령어를 통해 hex값을 확인하여 log영역에 기록되는 것을 확인한다.
      - ramdisk 를 1024MB로 생성한 경우
        - FAT1 영역 : 0x4000
        - FAT2 영역 : 0x203000
        - Log 영역 : 0x402000
        - Data 영역 : 0x601000
      - ramdisk 를 2048MB로 생성한 경우
        - FAT1 영역 : 0x4000
        - FAT2 영역 : 0x104000
        - Log 영역 : 0x204000
        - Data 영역 : 0x304000

      (그 외의 값으로 설정했을 시 각 영역의 시작 주소를 확인해 보아야 한다.)
  - ✓ 실습이 끝난 후
    - "umount ./disk" 를 통해 마운트 해제한다.
    - "# rmmod 만든모듈이름.ko"
      "# rmmod fat.ko" 작성한 모듈을 내린다.

2. NAND FLASH & F2FS file system

Flash memory란?

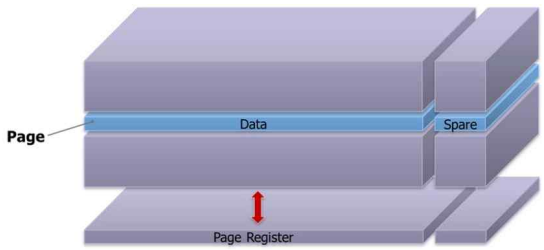
- Flash memory
  - ✓ 1984년 Toshiba에서 개발
  - ✓ ROM의 일종, EEPROM에서 변형
  - ✓ 전기적으로 데이터를 지우고 기록하는 비 휘발성 저장매체
  - ✓ 스마트폰, 디지털 카메라, SSD, MP3 등 데이터를 저장하는 대부분의 매체에 사용
  - ✓ 데이터를 저장하기 위한 방법으로 Floating Gate를 사용
  - ✓ NAND Flash와 NOR Flash로 구분



<http://icrontic.com/article/how_ssds_work>

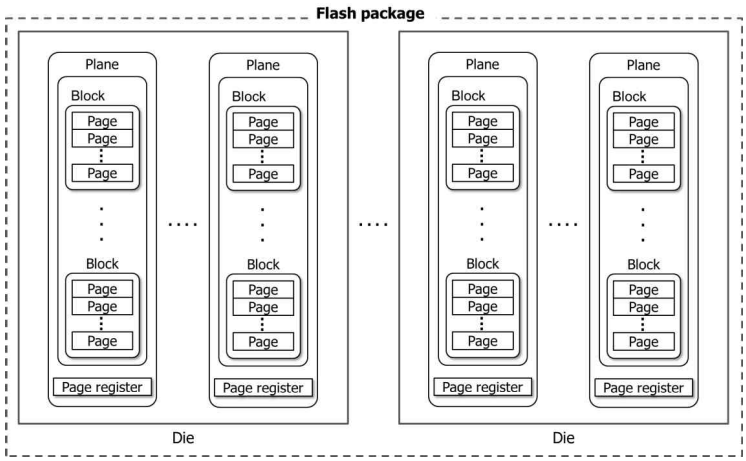
Flash memory의 구조

- NAND Flash의 구조
  - ✓ Page - R/W operation의 최소 단위
  - ✓ Spare - Page의 데이터가 저장되는 영역 이외의 별도의 영역으로 LSN(Logical Sector Number), ECC 등이 저장됨
  - ✓ Block - Erase operation의 최소 단위, 하나의 block은 여러 개의 page로 구성
  - ✓ Plane - 여러 개의 block으로 구성, 구조적으로 동시 연산이 가능하게 함
  - ✓ Die - 하나 이상의 plane과 이를 제어하기 위한 control unit으로 구성



Block

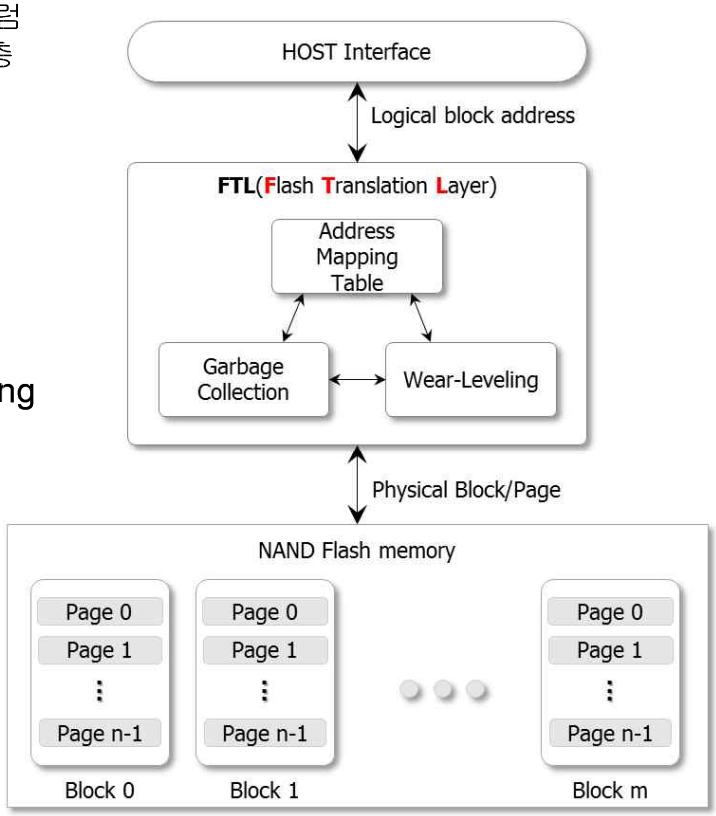
Block diagram



NAND Flash chip

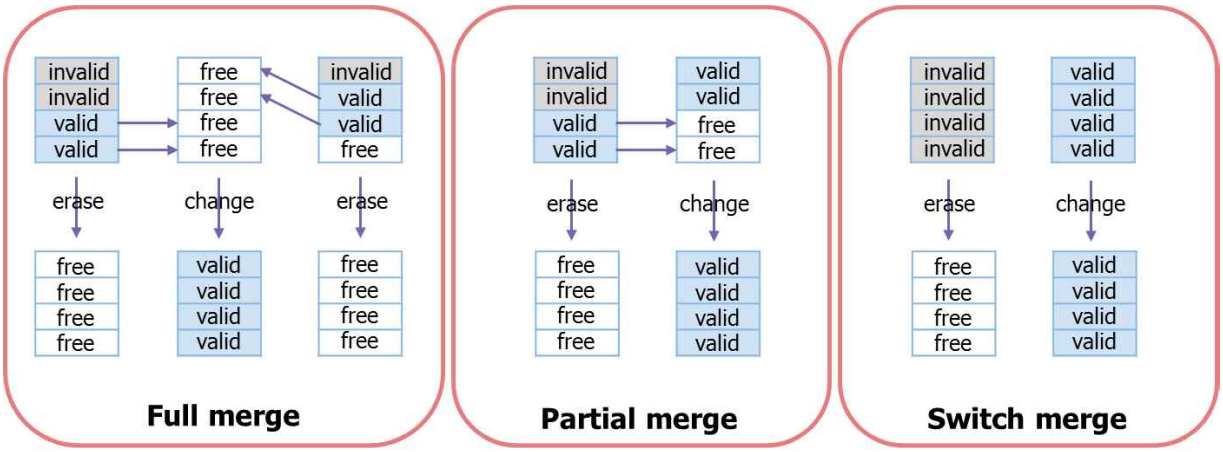
FTL 이란?

- FTL (Flash Translation Layer)
  - ✓ Host가 flash memory를 마치 disk처럼 다룰 수 있게 해주는 소프트웨어 계층
  - ✓ Flash memory 단점 보완
    - 비대칭적인 연산 단위 (R/W : page, Erase : block)
    - Out-of-place update
    - 제한된 수명
  - ✓ Address mapping 기법
    - Sector, Block, Hybrid mapping
  - ✓ Garbage Collection
    - Full, Partial, Switch merge
  - ✓ Wear-leveling



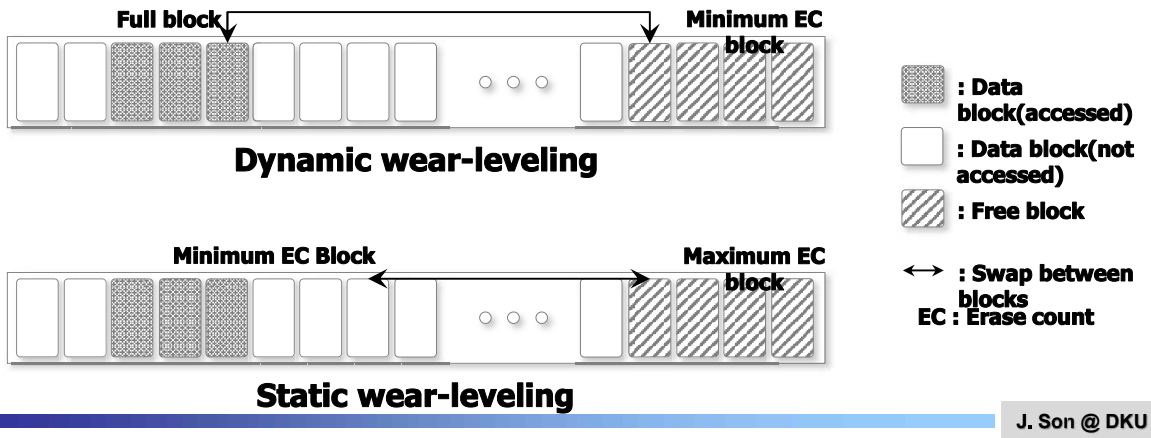
Garbage Collection

- ✓ Out-of-place update로 생겨난 invalid page들이 일정 개수만큼 쌓이거나 특정 시점에 한 번에 free pages로 바꾸기 위한 작업
- ✓ Merge를 통해 valid pages를 free page에 copy하고 invalid page가 포함된 block을 erase 처리
  - Full merge
  - Partial merge
  - Switch merge



Wear-leveling

- ✓ Flash memory의 수명을 연장하기 위한 작업
- ✓ 특정 블록에만 erase operation이 집중되는 것을 막고 모든 블록이 균등하게 쓰이도록 해주는 작업
- ✓ Static wear-leveling
  - Background로 수행되면서 플래시 메모리의 전 영역에 대해 wear-leveling이 적용되는 기법
- ✓ Dynamic wear-leveling
  - 현재 write operation이 수행하려고 하는 block에 wear-leveling이 수행되는 기법

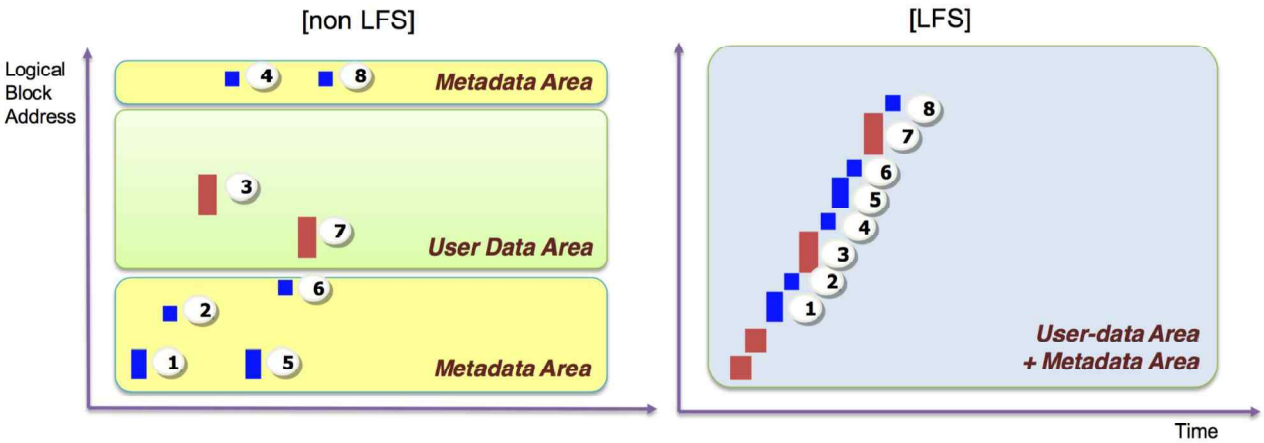


F2FS 실습

- F2FS 실습의 목표
  - ✓ F2FS의 전체적인 구조를 이해한다.
  - ✓ 자신이 직접 kernel source code를 분석하고 동작원리를 이해한다.
  - ✓ 이를 위해 kernel source code 분석에 필수적인 ctags의 사용법을 인지한다.
  - ✓ Kernel source code를 분석하는 능력을 향상시키고 이를 수정하여 현재의 kernel에 적용시키는 방법을 인지한다.
- F2FS 실습 내용
  - ✓ NAND FLASH의 경우 물리적인 특성으로 인해 garbage collection이 필수적이다.
  - ✓ Garbage collection은 overhead가 굉장히 큰 작업이기 때문에 전체적인 performance에 많은 영향을 끼칠 수 있다.
  - ✓ 현재 F2FS의 garbage collection이 실행되는 조건을 확인한 후 성능이 개선될 수 있도록 커널 소스 코드를 수정한다.
  - ✓ 소스 코드를 수정한 후 performance에 영향이 있는지 확인한다. (chapter 3. performance evaluation에서 진행)

LFS (Log-Structured File System)

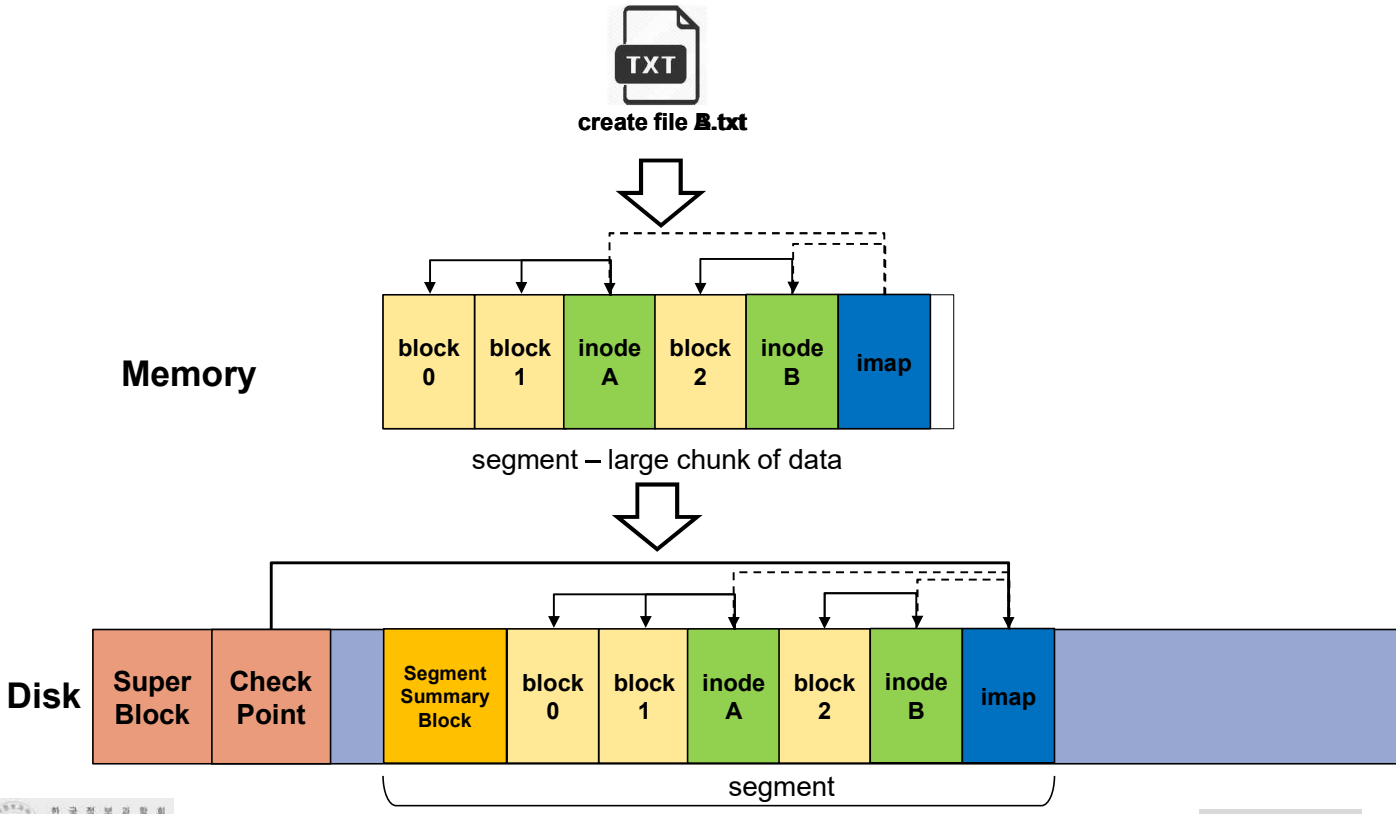
- LFS의 Main Idea
  - ✓ Disk 전체를 하나의 큰 log로 보며, 이 log는 다수의 segment로 이루어져있다.
  - ✓ Disk에 모든 write를 (modification 포함) log내에 sequential하게 write한다.
  - ✓ 이를 통해 small random write를 large sequential write로 변환한다.



<Source: J. Hwang, "Flash-Friendly File system", Feb. 2013>

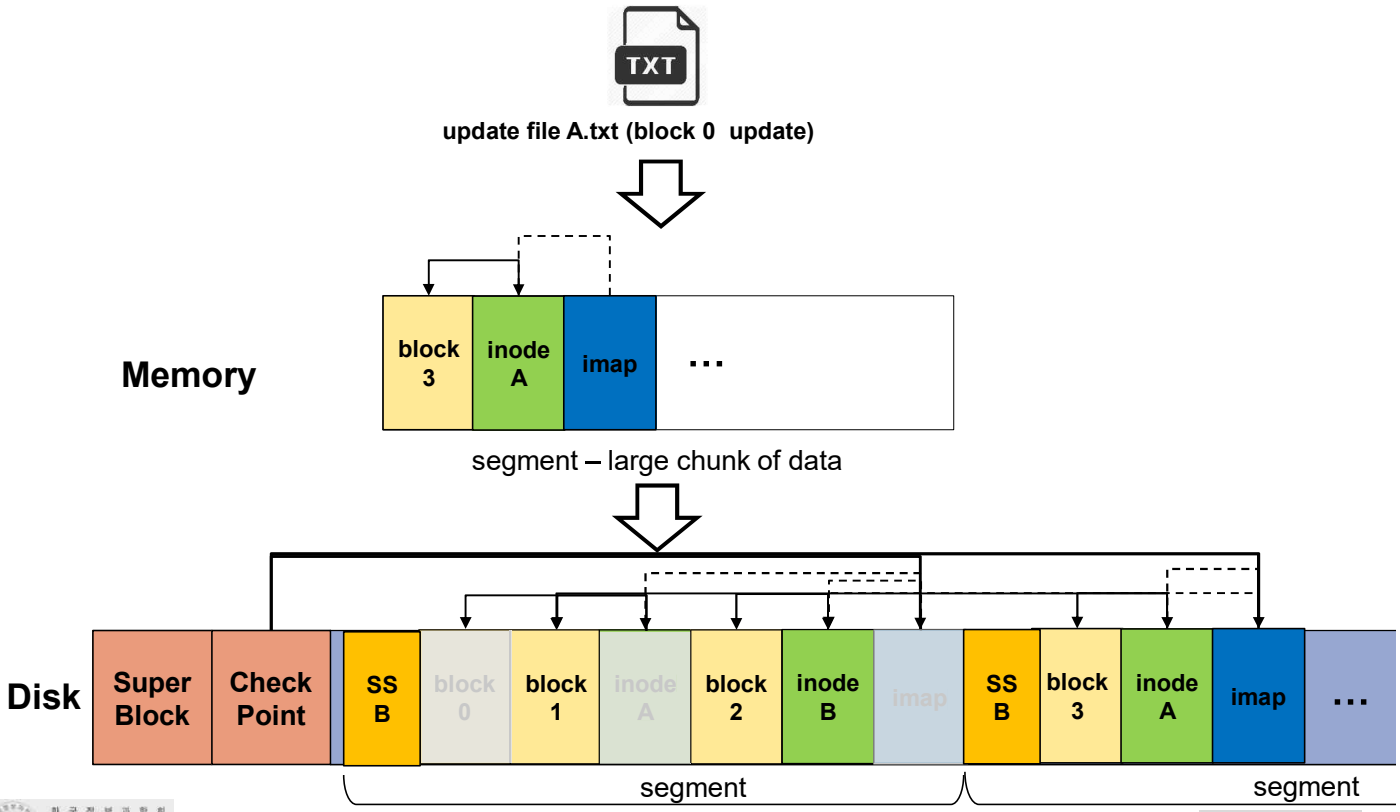
LFS의 구조

- log structured file system mechanism



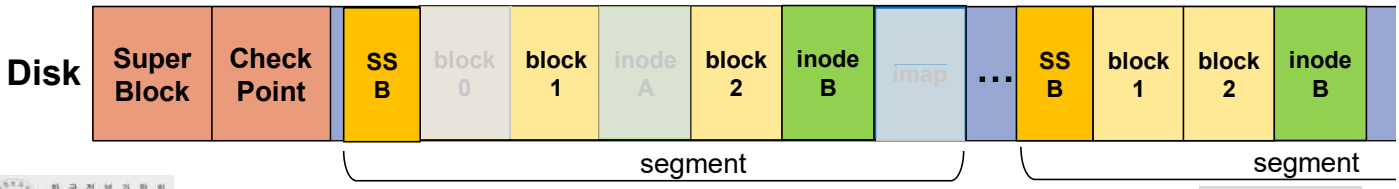
LFS의 구조

- log structured file system mechanism



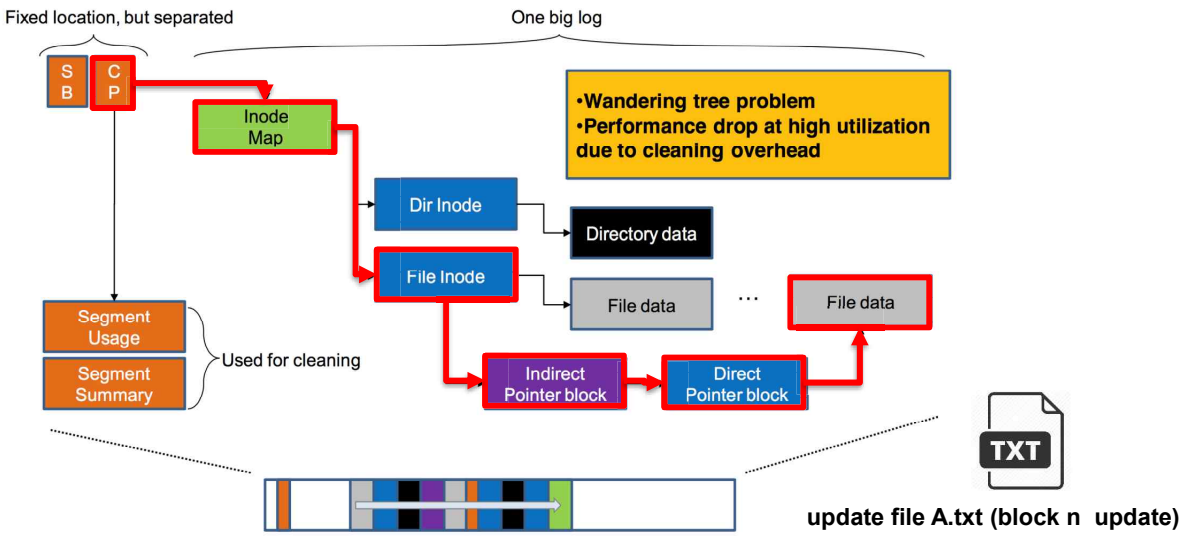
LFS

- LFS의 segment cleaning mechanism
  - ✓ 모든 data를 sequential하게 write하면 segment내에 free space가 부족한 경우가 발생한다.
  - ✓ Sequential write을 하기 위해 segment cleaning을 통해 free space를 확보해야 한다.
  - ✓ Segment cleaning 기법으로 다음 두 가지가 존재한다.
    - LFS greedy policy : segment들 중에 사용량이 가장 작은 segment를 cleaning해주는 기법이다.
    - LFS cost-benefit policy : segment를 cleaning하는 cost와 cleaning으로 인해 얻어지는 benefit을 수식화한다. 이 수치가 가장 높은 segment를 cleaning함으로써 cleaning cost는 적고, cleaning으로 인해 얻어지는 benefit은 가장 높은 segment를 선정하는 기법이다.

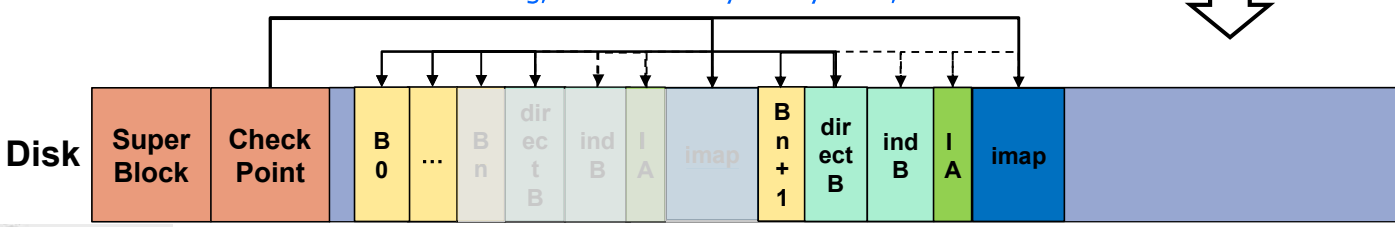


LFS의 구조

✓ LFS의 경우 metadata update시 metadata update propagation이 발생할 수 있다.



<Source: J. Hwang, "Flash-Friendly File system", Feb. 2013>

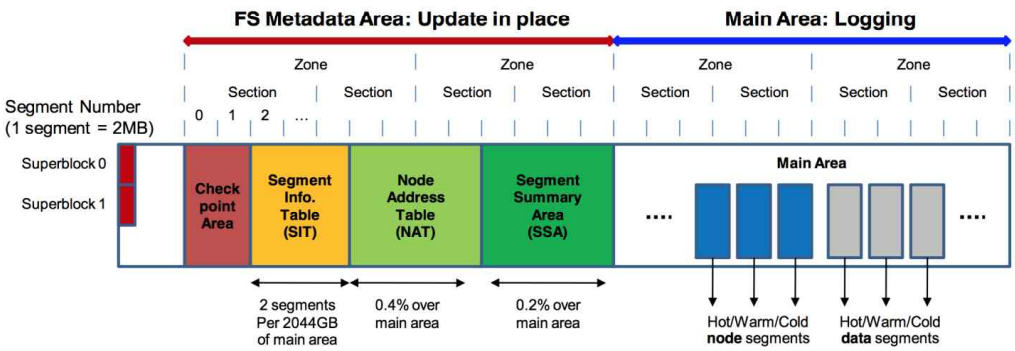


F2FS(Flash Friendly File System) Design Overview

- Goal of F2FS
  - ✓ flash memory의 경우 erase-before-write, block당 write cycle 등의 여러가지 한계점을 지니고 있다.
  - ✓ flash memory의 한계점을 극복하기위해 최적화된 file system이 F2FS이다.
- Metadata update propagation 방지.
- Efficient cleaning using Multi-head logs and Hot/Cold data separation.
  - ✓ Foreground/background cleaning.
  - ✓ Automatic background cleaning.
- Adaptive write policy.
  - ✓ Threaded logging/logging.

F2FS의 구조

- F2FS On-Disk Layout
  - ✓ Super Block : 기본적인 F2FS의 모든 정보를 가지고있다.
  - ✓ Check Point : system crash로 인해 system을 복구하기위해 필요한 정보를 가지고있다.
  - ✓ Segment Information Table : segment당 현재 segment의 정보를 가지고있다.
  - ✓ Node Address Table : Main Area의 node의 물리적인 주소를 가지고있다.
  - ✓ Segment Summary Area : Main Area의 모든 block들에 관한 정보를 가지고있다.
  - ✓ Main Area : Node(meta data)와 Data들이 할당되어지는 부분이다.



* Block size = 4KB

<Source: J. Hwang, "Flash-Friendly File system", Feb. 2013>

F2FS의 구조

- F2FS On-Disk Layout
  - ✓ Super Block : total segment count, total block count, sector size 등의 file system 포맷 후 변하지 않는 file system의 특성이 되는 기본적인 정보들을 담고있다. Size에 관한 것 뿐만 아니라 SIT,NAT,SSA등의 시작 주소와 해당 구역의 크기에 대한 정보를 관리한다. 또한 root inode, node inode meta inode에 관한 정보도 관리한다.

</usr/linux/include/linux/f2fs_fs.h>

```
/*
 * For superblock
 */
struct f2fs_super_block {
    __le32 magic;           /* Magic Number */
    __le16 major_ver;       /* Major Version */
    __le16 minor_ver;       /* Minor Version */
    __le32 log_sectorsize;      /* log2 sector size in bytes */
    __le32 log_sectors_per_block;   /* log2 # of sectors per block */
    __le32 log_blocksize;       /* log2 block size in bytes */
    __le32 log_blocks_per_seg;  /* log2 # of blocks per segment */
    __le32 segs_per_sec;        /* # of segments per section */
    __le32 secs_per_zone;       /* # of sections per zone */
    __le32 checksum_offset;     /* checksum offset inside super block */
    __le64 block_count;     /* total # of user blocks */
    __le32 section_count;       /* total # of sections */
    __le32 segment_count;       /* total # of segments */
    __le32 segment_count_ckpt;  /* # of segments for checkpoint */
    __le32 segment_count_sit;   /* # of segments for SIT */
    __le32 segment_count_nat;   /* # of segments for NAT */
    __le32 segment_count_ssa;   /* # of segments for SSA */
    __le32 segment_count_main;  /* # of segments for main area */
    __le32 segment0_blkaddr;    /* start block address of segment 0 */
    __le32 cp_blkaddr;      /* start block address of checkpoint */
    __le32 sit_blkaddr;     /* start block address of SIT */
    __le32 nat_blkaddr;     /* start block address of NAT */
    __le32 ssa_blkaddr;     /* start block address of SSA */
    __le32 main_blkaddr;        /* start block address of main area */
    __le32 root_ino;        /* root inode number */
    __le32 node_ino;        /* node inode number */
    __le32 meta_ino;        /* meta inode number */
    __u8 uuid[16];          /* 128-bit uuid for volume */
    __le16 volume_name[512];    /* volume name */
    __le32 extension_count;     /* # of extensions below */
    __u8 extension_list[F2FS_MAX_EXTENSION][8]; /* extension array */
} __packed;
```


F2FS의 구조

- F2FS On-Disk Layout
  - ✓ Check Point : 현재 남은 valid block 혹은 inode를 관리하며 여러 종류의 segment 를 관리하고 있다. File system이 mount된 시점의 시간, SIT, NAT의 현재 정보또 한 관리한다.

</usr/linux/include/linux/f2fs_fs.h>

```
struct f2fs_checkpoint {
    __le64 checkpoint_ver;      /* checkpoint block version number */
    __le64 user_block_count;    /* # of user blocks */
    __le64 valid_block_count;   /* # of valid blocks in main area */
    __le32 rsvd_segment_count;  /* # of reserved segments for gc */
    __le32 overprov_segment_count;  /* # of overprovision segments */
    __le32 free_segment_count;  /* # of free segments in main area */

    /* information of current node segments */
    __le32 cur_node_segno[MAX_ACTIVE_NODE_LOGS];
    __le16 cur_node_blkoff[MAX_ACTIVE_NODE_LOGS];
    /* information of current data segments */
    __le32 cur_data_segno[MAX_ACTIVE_DATA_LOGS];
    __le16 cur_data_blkoff[MAX_ACTIVE_DATA_LOGS];
    __le32 ckpt_flags;      /* Flags : umount and journal_present */
    __le32 cp_pack_total_block_count;   /* total # of one cp pack */
    __le32 cp_pack_start_sum;   /* start block number of data summary */
    __le32 valid_node_count;    /* Total number of valid nodes */
    __le32 valid_inode_count;   /* Total number of valid inodes */
    __le32 next_free_nid;       /* Next free node number */
    __le32 sit_ver_bitmap_bytesize; /* Default value 64 */
    __le32 nat_ver_bitmap_bytesize; /* Default value 256 */
    __le32 checksum_offset;     /* checksum offset inside cp block */
    __le64 elapsed_time;        /* mounted time */
    /* allocation type of current segment */
    unsigned char alloc_type[MAX_ACTIVE_LOGS];

    /* SIT and NAT version bitmap */
    unsigned char sit_nat_version_bitmap[1];
} __packed;
```

F2FS의 구조

- F2FS On-Disk Layout
  - ✓ NAT(Node Address Table) : inode number와 block address를 가지며 Main Area 에 저장된 모든 node들의 block adress를 관리한다.

</usr/linux/include/linux/f2fs_fs.h>

```
struct f2fs_nat_block {
    struct f2fs_nat_entry entries[NAT_ENTRY_PER_BLOCK];
} __packed;
```

```
struct f2fs_nat_entry {
    __u8 version;        /* latest version of cached nat entry */
    __le32 ino;      /* inode number */
    __le32 block_addr;  /* block address */
} __packed;
```

  - ✓ SAT(Segment Information Table) : 각 segment들의 valid block bitmap과 age를 관리 한다.

</usr/linux/include/linux/f2fs_fs.h>

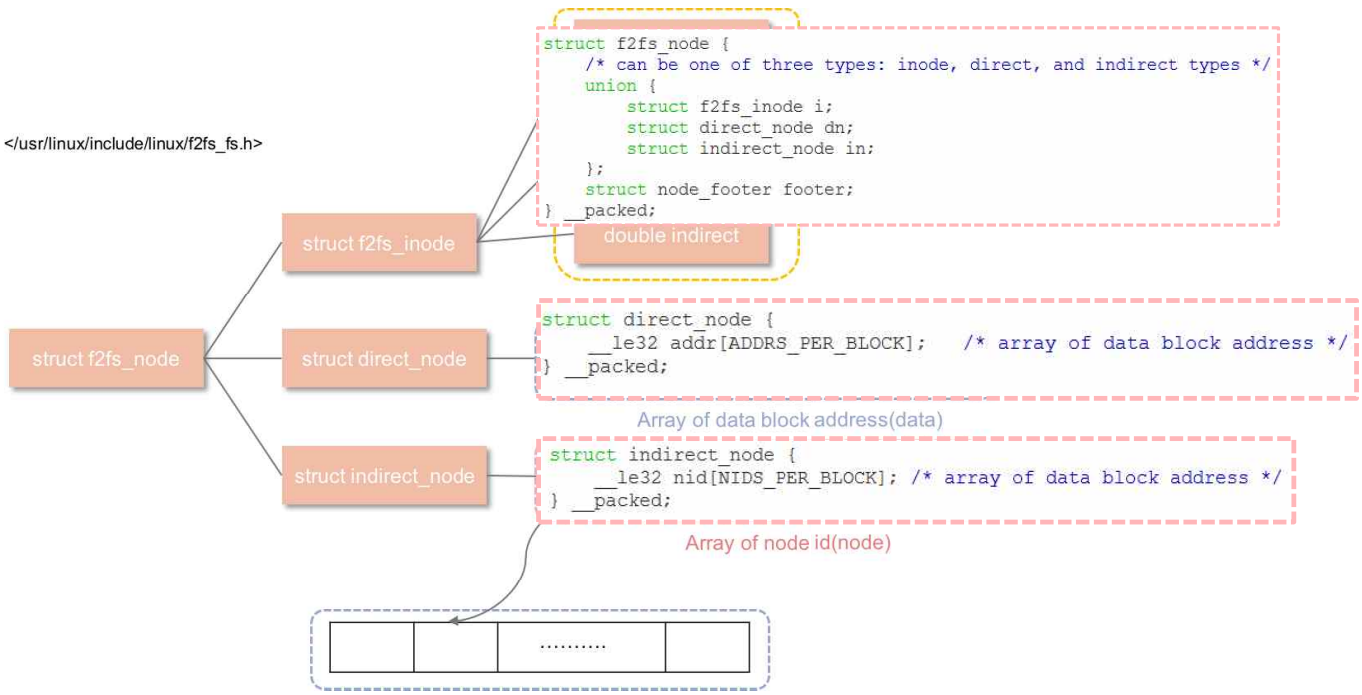
```
struct f2fs_sit_block {
    struct f2fs_sit_entry entries[SIT_ENTRY_PER_BLOCK];
} __packed;
```

```
struct f2fs_sit_entry {
    __le16 vblocks;              /* reference above */
    __u8 valid_map[SIT_VBLOCK_MAP_SIZE];    /* bitmap for valid blocks */
    __le64 mtime;                /* segment age for cleaning */
} __packed;
```


F2FS 구조

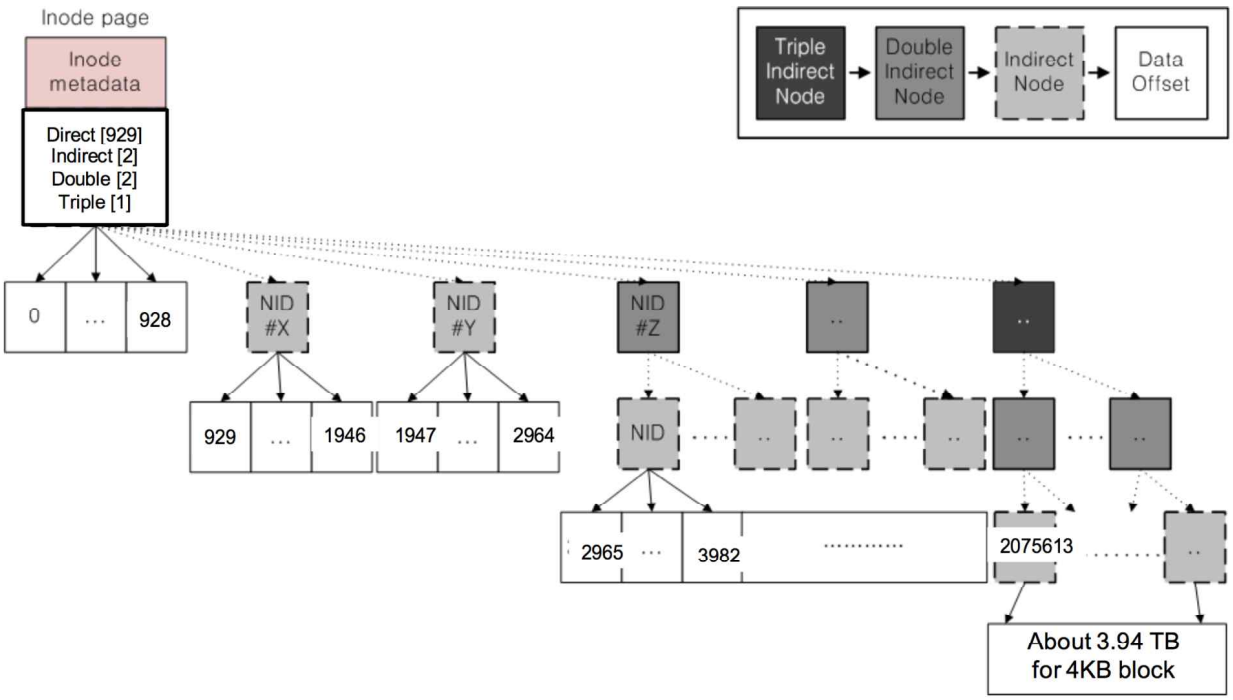
- F2FS file structure
  - ✓ F2FS file 구조의 주요 구조체는 다음과 같다.

</usr/linux/include/linux/f2fs_fs.h>



F2FS 구조

- F2FS file structure



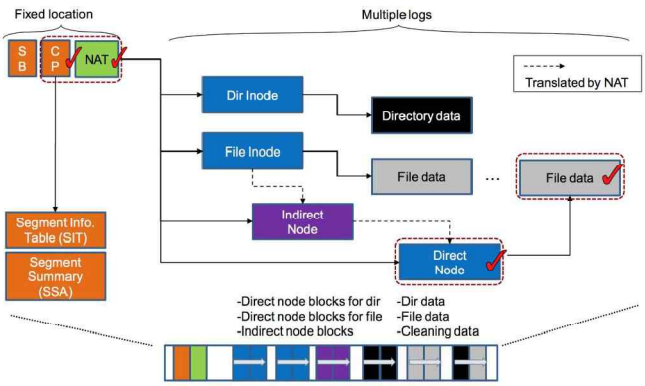
<Source: J. Hwang, "Flash-Friendly File system", Feb. 2013>

F2FS – update propagation issue

✓ F2FS의 구조는 기존의 LFS에서 존재하던 update propagation issue를 해결하였다.

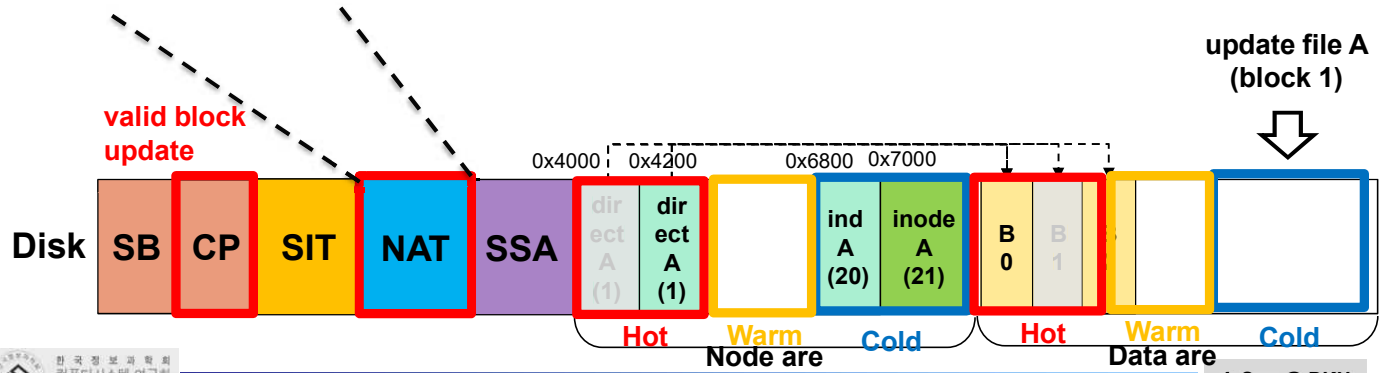
Node Address Table

nid	blocknr
1	0x4200
20	0x6800
21	0x7000
…	…



<Source: J. Hwang, "Flash-Friendly File system", Feb. 2013>

Type	Temp.	Objects
Node	Hot	Direct node blocks for directories
	Warm	Direct node blocks for regular files
	Cold	Indirect node blocks
Data	Hot	Directory entry blocks
	Warm	Data blocks made by users
	Cold	Data blocks moved by cleaning; Cold data blocks specified by users; Multimedia file data



F2FS 구조

- F2FS directory entry structure
  - ✓ F2FS에서 directory entry(dentry)는 struct f2fs_dir_entry와 f2fs_dentry_block을 통해 표현되고 이 구조체를 통해 원하는 (dentry)로 이동하는 연산을 수행한다.
  - ✓ Directory entry는 multi-level hash table 구조를 이루고있으며 이를 통해 효율적으로 dentry를 찾는다.
  - ✓ 해당 directory를 lookup하기 위한 연산은 /usr/src/linux/3.19.6/fs/f2fs/namei.c의 f2fs_lookup 함수부터 시작된다.

</usr/linux/include/linux/f2fs_fs.h>

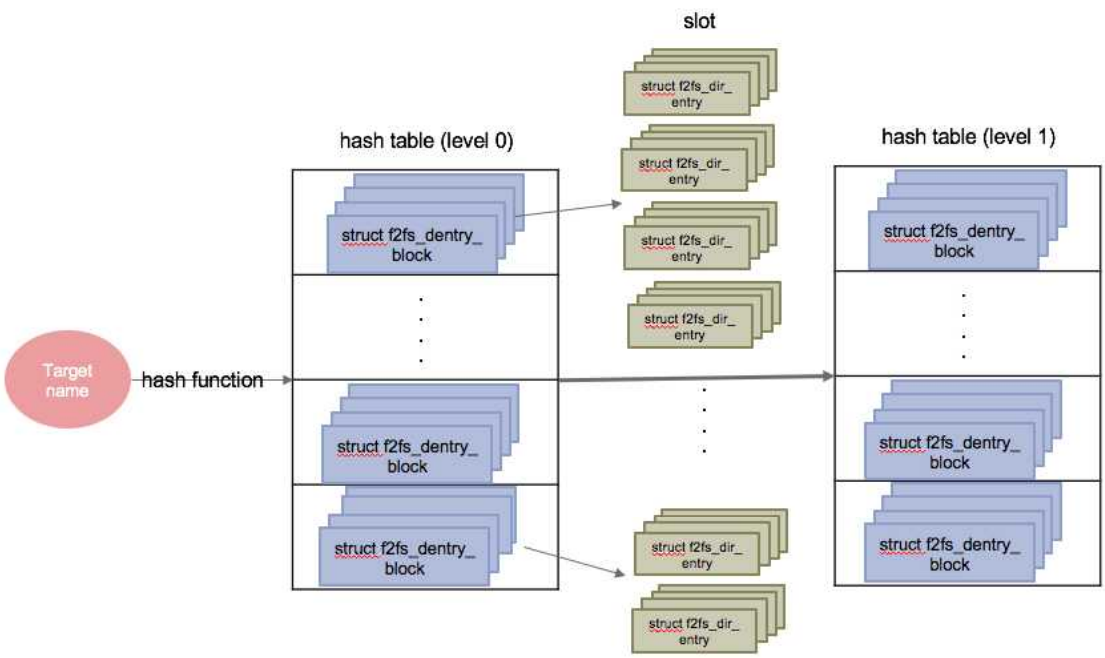
```
/* One directory entry slot representing F2FS_SLOT_LEN-sized file name */
struct f2fs_dir_entry {
    __le32 hash_code;   /* hash code of file name */
    __le32 ino;     /* inode number */
    __le16 name_len;    /* lengh of file name */
    __u8 file_type;     /* file type */
} __packed;
```

</usr/linux/include/linux/f2fs_fs.h>

```
/* 4KB-sized directory entry block */
struct f2fs_dentry_block {
    /* validity bitmap for directory entries in each block */
    __u8 dentry_bitmap[SIZE_OF_DENTRY_BITMAP];
    __u8 reserved[SIZE_OF_RESERVED];
    struct f2fs_dir_entry dentry[NR_DENTRY_IN_BLOCK];
    __u8 filename[NR_DENTRY_IN_BLOCK][F2FS_SLOT_LEN];
} __packed;
```

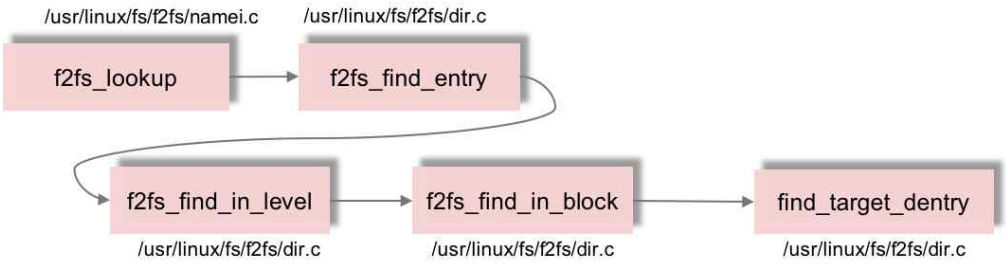

F2FS 구조

- F2FS directory entry structure
  - ✓ F2FS의 multi-level hash 구조는 다음과 같다.



F2FS 구조

- F2FS directory entry structure
  - ✓ 원하는 directory entry를 lookup하기 위한 함수 call flow는 다음과 같다.



  - ✓ 주어진 이름으로 level ,f2fs_dentry_block(bucket) ,f2fs_dir_entry(slot) 순으로 lookup을 수행한다.
  - ✓ 최종적으로 ino(inode number)를 통해 주어진 이름을 가진 inode를 가져오고 이를 dentry와 연결시키면서 VFS상으로 user가 원하는 정보를 넘겨준다.

F2FS 실습

실습 환경 구축 – f2fs-tools

- f2fs-tools
  - ✓ 현재 커널 버전인 linux-3.19.6에는 디바이스를 F2FS의 형태로 포맷시켜주는 명령어인 mkfs.f2fs가 존재하지 않는다. 이를 위해 f2fs-tools라는 tool을 사용한다.
  - ✓ f2fs-tools를 사용하기위해는 여러 패키지들이 필요하다. 다음 명령어를 이용해 패키지를 설치한다.
    - # apt-get install libuuid-devel" or "# apt-get install uuid-dev
    - # apt-get install pkg-config
    - # apt-get install autoconf
    - # apt-get install libtool
  - ✓ "# tar –xvf f2fs-tools_1.4.1.orig.tar.xz"를 통해 압축을 해제 시킨다.
  - ✓ f2fs-tools-1.4.1 디렉토리로 이동 후, 다음의 명령어들을 통해 f2fs-tools 설치
    - "# cd /home/fs-lecture/lecture/f2fs-tools-14.1"
    - "# autoreconf --install"
    - "# ./configure"

실습 환경 구축 – f2fs-tools

- f2fs-tools
  - ✓ f2fs-tools-1.4.1 디렉토리의 mkfs 디렉로리로 이동한 후 Makefile.am 파일의 맨 마지막에 다음 명령어를 입력한다.
    - "# vim mkfs/Makefile.am"
    - mkfs_f2fs_LDFLAGS = -all-static
  - ✓ f2fs-tools-1.41 디렉토리의 fsck 디렉로리로 이동한 후 Makefile.am 파일의 맨 마지막에 다음 명령어를 입력한다.
    - "# vim fsck/Makefile.am"
    - fsck_f2fs_LDFLAGS = -all-static
  - ✓ 수정된 파일을 저장한 후에 f2fs-tools-1.41 디렉토리로 이동한 후에 다음 명령어를 입력한다.
    - " # LDFLAGS=--static ./configure --host=arm-none-linux-gnueabi --target=arm-none-linux-gnueabi"
  - ✓ make 명령어를 통해 컴파일 한다.
  - ✓ 위의 과정을 통해 "f2fs-tools-1.4.1/mkfs" 디렉토리에 생성된 "mkfs.f2fs" 실행파일을 통해 기존의 리눅스에 존재하던 포맷 명령어인 mkfs를 대신할 수 있게 되었다.

실습 환경 구축 – MTD

- MTD module mount
  - ✓ 현재 우리가 사용중인 ram disk는 block device의 형태이고 우리가 mount시키려는 F2FS file system은 flash memory 즉 SSD와 같은 device를 위해 만들어진 file system이다. 이를 위해 block device를 flash memory처럼 다룰 수 있게 해주는 device driver인 MTD module을 먼저 적재시켜야 한다.
  - ✓ MTD에서 내부적인 ram disk를 제공하기 때문에 rapiddisk를 사용할 필요가 없다.
  - ✓ MTD ram disk를 적재시키기 위해 필요한 module은 다음과 같다.
    - mtd.ko
    - mtd_blkdevs.ko
    - mtdblock.ko
    - mtdram.ko (/home/fs-lecture/lecture/mtd_module/devices 에 존재)

실습 환경 구축 – MTD

- MTD module mount
  - ✓ "# cd mtd_module"을 통해 배포한 MTD module이 위치한 디렉토리로 이동한다.
  - ✓ 컴파일한다.
    - "# make"
  - ✓ 다음의 명령어를 통해 필요한 mtd module들을 적재시키고 확인한다.
    - "# insmod mtd.ko"
    - "# insmod mtd_blkdevs.ko"
    - "# insmod mtdblock.ko"
    - "# insmod devices/mtdram.ko"
      (mtdram.ko는 mtd_module/devices에 존재한다.)

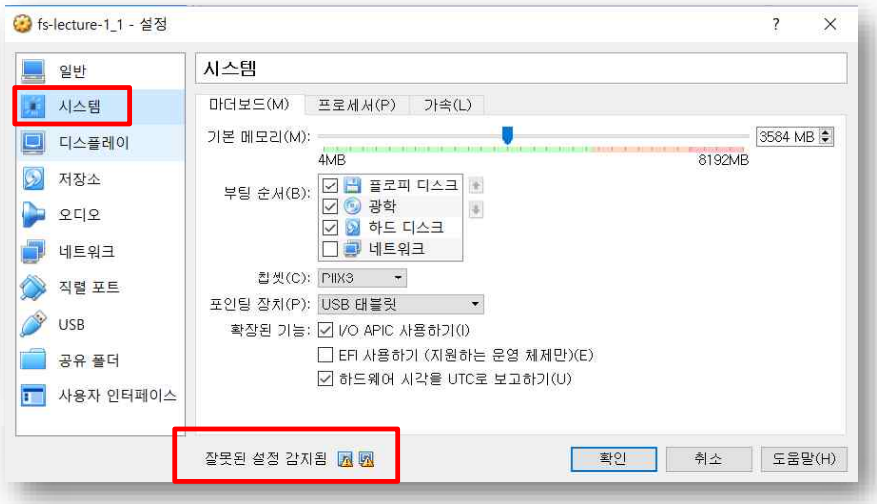
```
Module                  Size  Used by
cmdlinepart            12779  0
mtdram                 12972  0
mtdblock               13463  0
mtd_blkdevs            18407  1 mtdblock
mtd                    60252  6 cmdlinepart,mtdram,mtdblock,mtd_blkdevs
```

  - ✓ 위의 module들을 적재시키면 /dev에 mtdblock0가 생성되고 이는 MTD 내부에 존재하는 ram disk이며 mtdram.ko module을 적재시키면서 생기는 device이다.

```
crw-------   1 root root     90,   0  7월  15 15:56 mtd0
crw-------   1 root root     90,   1  7월  15 15:56 mtd0ro
brw-rw----   1 root disk     31,   0  7월  15 15:56 mtdblock0
```

실습 환경 구축 – MTD

- MTD module mount 주의점 1
  - ✓ "# insmod devices/mtdram.ko" 수행을 통해 모듈을 올릴 시 가상 머신을 생성한 설정의 메모리가 부족하거나 현재 시스템의 상태와 맞지 않는다면 문제가 다음과 같이 Host Memory Low오류가 발생할 수 있다. 가상머신 설정을 재조정하여 실행하여 잘못된 설정을 고침으로써 오류를 수정할 수 있다.



실습 환경 구축 – F2FS mount

- MTD module mount 주의점 2
  - ✓ "# insmod devices/mtdram.ko" 수행을 통해 모듈을 올릴 시 다음과 같은 에러가 발생할 수 있다.

```
root@fslecture-VirtualBox:/home/fs-lecture/lecture/mtd_module# insmod devices/mtdram.ko
insmod: error inserting 'devices/mtdram.ko': -1 Cannot allocate memory
```

  - ✓ 해당 error는 현재 가상 이미지의 ram크기보다 /dev/mtdblock0의 MTD_RAM_TOTAL_SIZE 가 크기 때문에 발생하는 error이다.
  - ✓ 이 error가 발생하는 경우 /mtd_module/devices 의 mtdram.c 를 수정해주어야 한다.(line 22 - MTDRAM_TOTAL_SIZE)

- static unsigned long total_size = CONFIG_MTDRAM_TOTAL_SIZE;
  - /usr/src/linux-3.19.6/include/generated/autoconf.h에 CONFIG_MTDRAM_TOTAL_SIZE가 4096으로 define되어있다.

다음 값을 하향조정해 준다.

- #define MTD_RAM_TOTAL_SIZE (total_size * 1024 * 512)
- 단, MTD_RAM_TOTAL_SIZE를 F2FS를 format하기 위한 minimum size이상으로 설정해준다.
  - ex) #define MTD_RAM_TOTAL_SIZE(total_size * 1024 * (25이상))

  - ✓ 수정한 후 다시 compile을 한다.
    - " # make"

실습 환경 구축 – F2FS mount

- F2FS mount
  - ✓ FAT file system mount와는 다르게 F2FS에서는 MTD module을 적재시킴으로써 생성된 /dev/mtdblock0를 사용한다.
  - ✓ mtdblock을 F2FS로 포맷하기위해 f2fs-tools-1.4.1/mkfs 디렉토리에 있는 mkfs.f2fs 실행파일을 이용한다.
    - "# ./mkfs.f2fs /dev/mtdblock0"

실습 환경 구축 – F2FS mount

- F2FS mount
  - ✓ ./mkfs.f2fs를 이용하여 device를 format할 때 주의할 점.
  - ✓ "# ./mkfs.f2fs /dev/mtdblock0" 로 디바이스를 format할 때 다음과 같은 결과가 나오면 성공한 것이다.

```
        F2FS-tools: mkfs.f2fs Ver: 1.4.1 (2015-05-07)

Info: Debug level = 0
Info: Label =
Info: Overprovision ratio = 5%
Info: Segments per section = 1
Info: Sections per zone = 1
Info: Trim is enabled
Info: sector size = 512
Info: total sectors = 8192000 (4000 MB)
Info: zone aligned segment0 blkaddr: 512
Info: format version with
  "Linux version 3.19.6 (root@donghwa-com) (gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5) ) #2 SMP Mon Jul 13 21:55:21 KST 2015"
Info: Discarding device
Info: This device doesn't support TRIM
Info: format successful
```

  - ✓ format에 실패하는 경우 다음과 같은 결과가 나온다.

```
        F2FS-tools: mkfs.f2fs Ver: 1.4.1 (2015-05-07)

Info: Debug level = 0
Info: Label =
Info: Overprovision ratio = 5%
Info: Segments per section = 1
Info: Sections per zone = 1
Info: Trim is enabled
Info: sector size = 512
Info: total sectors = 8192 (4 MB)
Error: Min volume size supported is 104857600
```

mtd module mount 주의점 2 에서의 조정값이 25보다 크지 않아서 발생하는 문제이다. 이를 25이상으로 조정한다.

실습 환경 구축 – ctags

- ctags 설정
  - ✓ Ctags는 프로그래밍 소스코드의 태그(전역변수 선언, 함수 정의, 매크로 정의)들의 데이터 베이스(tags 파일)을 생성하는 명령어이다.
  - ✓ "# ctags --version" 명령어를 입력한다.
    - 설치되어 있는 경우 ctags의 version 정보가 출력된다.
    - 설치되어 있지 않은 경우 error가 출력된다.
  - ✓ 설치되어 있지 않은 경우 이를 설치한다.
    - "# apt-get install ctags"
  - ✓ Ctags를 사용하고자하는 directory로 이동하여 "# ctags –R"명령어를 이용하여 tags 파일을 생성한다.
    - "# cd /home/fs-lecture/lecture/f2fs_module"
    - "# ctags –R"
  - ✓ 해당 directory로 이동하여 tags파일이 생성되었는지 확인한다.
  - ✓ "# apt-get install vim"을 통해 vim을 설치한다.

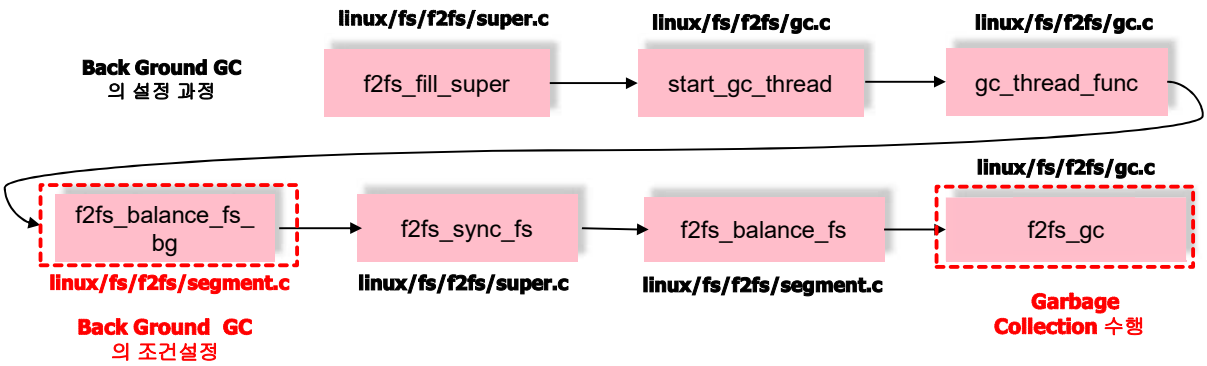
실습 환경 구축 – ctags

- ctags 설정
  - ✓ 설치가 완료되고 tags파일이 생성되었으면 .vimrc 파일을 생성해야한다.
  - ✓ vimrc 파일을 open한다.
    - "# vi ~/.vimrc"
  - ✓ vimrc파일에 다음 라인을 입력한 후 저장한다.
    - "set tags=/home/fs-lecture/lecture/f2fs_module/tags"
- ctags 사용법
  - ✓ vi에서 ":"을 눌러서 명령어 입력모드로 변환한다.
  - ✓ ex 모드에서 ":tj 함수명 or 구조체명"을 통해서 원하는 source code를 찾는다.
  - ✓ 혹은 vi를 열어서 source code를 분석하는 중에 함수 원형이나 구조체이름에 커서를 위치시키고 "ctrl +]"를 누르면 자동으로 태그를 찾아간다.
  - ✓ 이전으로 다시 되돌아 오고 싶은경우 "ctrl + t"를 사용해서 돌아올 수 있다.

F2FS 실습

- F2FS 실습

  - ✓ F2FS의 garbage collection의 function call flow

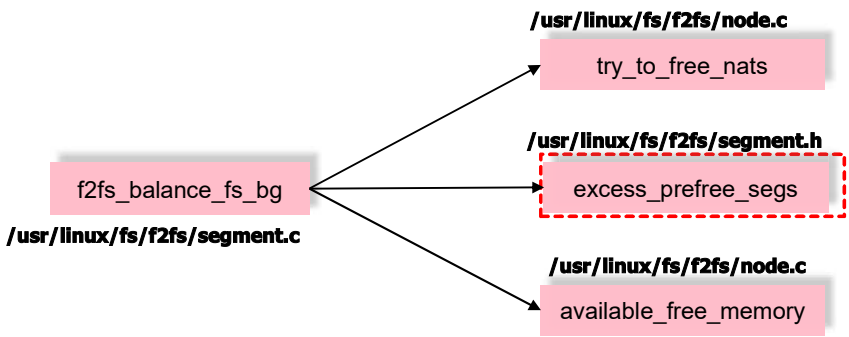


  - ✓ Garbage collection은 언제, 어떻게 수행하느냐에 따라 전체 file system의 performance에 큰 영향을 미칠 수 있다.

  - ✓ 이러한 점을 감안하여 간단히 수정할 수 있는 f2fs_balance_fs_bg를 분석하고 이를 수정한다. 또한 함수를 따라가며 각 함수들이 어떠한 역할을 하는지 간단히 파악할 수 있도록 한다.

F2FS 실습

- F2FS 실습



  - ✓ ctags를 이용하여 f2fs_balance_fs_bg를 찾는다. 이 함수는 3가지 함수의 return value가 하나라도 true인 경우 실행된다.

  - ✓ 본 실습에서는 간단히 분석과 수정이 가능한 excess_prefree_segs를 분석하고 수정하는 것을 목표로한다.

  - ✓ 수정을 하기 전에 f2fs_module 디렉토리를 복사해 놓은 후에 실습을 진행한다.
    - 이후에 진행할 performance evaluation 실습을 위해서 수정을 하는 f2fs_module과 수정하지 않은 f2fs_module을 구분해 놓는다.

F2FS 실습

- F2FS 실습 tip
  - ✓ excess_prefree_segs 함수를 분석하면 기존에 정의되어있는 값인 rec_prefree_segement 변수를 확인 할 수 있을 것이다.
  - ✓ 현재 f2fs의 경우에는 dirty한 block이 rec_prefree_segment 값 이상인 경우 garbage collection을 수행한다는 것을 확인할 수 있을 것이다.
  - ✓ 간단히 수정을 하고자 한다면 rec_prefree_segment 변수가 초기화 되는 부분을 찾아서 이 값을 수정하여 garbage collection이 발생되는 조건을 변경시킬 수 있다.
  - ✓ 이를 위해 rec_prefree_segment가 초기화 되는 부분을 찾고자 한다면 다음의 명령어를 사용하여 찾도록 한다.
  - ✓ "# grep –rn "rec_prefree_segment" ./ (찾을 문자열)
  - ✓ grep 명령어는 파일에서 특정한 패턴(문자열)을 찾는 명령어이다. 이는 해당 문자열이 있는 파일을 찾아 그 문자열이 들어가 있는 라인(행)을 출력한다.
  - ✓ "-rn"옵션은 각각 r은 서브 디렉토리의 파일의 결과 값까지 출력하는 옵션이며 n은 문자열이 들어있는 라인과 문두에 라인번호를 출력하는 옵션이다.
  - ✓ "./" 은 grep 명령어를 통해 찾고자하는 문자열을 현재 디렉토리에서 찾으라는 의미이다.

F2FS 실습

- F2FS 실습 tip
  - ✓ grep 수행 결과

```
./segment.c:2103:        sm_info->rec_prefree_segments = sm_info->main_segments *
./f2fs.h:433:    unsigned int rec_prefree_segments;
./segment.h:472:         return prefree_segments(sbi) > SM_I(sbi)->rec_prefree_segments;
./super.c:193:F2FS_RW_ATTR(SM_INFO, f2fs_sm_info, reclaim_segments, rec_prefree_segments);
```

문자열이 존재하는 파일:파일내에 문자열이 존재하는 line number　　찾고자하는 문자열이 존재하는 line의 문장

  - ✓ 기본적으로 kernel source code를 분석하는데 ctags, grep은 필수적이다.
  - ✓ 이외에 cscope, tlist등이 존재한다.

```
" Press <F1> to display hel
dir.c (/home/donghwa/lectur
  macro
    S_SHIFT

  variable
    f2fs_filetype_table
    f2fs_type_by_mode
    f2fs_dir_operations

  function
    dir_blocks
    dir_buckets
    bucket_blocks
    set_de_type
    dir_block_index
    early_match_name
    find_in_block
    find_target_dentry
    find_in_level
    f2fs_find_entry
```

```
/*
 * fs/f2fs/dir.c
 *
 * Copyright (c) 2012 Samsung Electronics Co., Ltd.
 *             http://www.samsung.com/
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */
#include <linux/fs.h>
#include <linux/f2fs_fs.h>
#include "f2fs.h"
#include "node.h"
#include "acl.h"
#include "xattr.h"

static unsigned long dir_blocks(struct inode *inode)
{
    return ((unsigned long long) (i_size_read(inode) + PAGE_CACHE_SIZE - 1))
                        >> PAGE_CACHE_SHIFT;
}
```

실습 환경 구축 – F2FS mount

- F2FS compile & mount
  - ✓ 수정한 f2fs을 컴파일 한다.
    - "# make"
  - ✓ f2fs모듈을 적재시킨다. (컴파일이 되어 있지 않다면 "# make" 해준다)
    - "# insmod f2fs_lecture.ko"
  - ✓ "# mount –t f2fs_lecture /dev/mtdblock0 /mnt"를 통해 mount point인 /mnt로 장치를 mount시킨다.
  - ✓ 다음 두 명령어를 이용해서 file system이 제대로 mount되었는지 확인한다.
    - "# mount"

```
tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
none on /run/lock type tmpfs (rw,noexec,nosuid,nodev,size=5242880)
none on /run/shm type tmpfs (rw,nosuid,nodev)
gvfs-fuse-daemon on /var/lib/lightdm/.gvfs type fuse.gvfs-fuse-daemon (rw,nosuid,nodev,user=lightdm)
/dev/rxd0 on /mnt/fat_test type vfat (rw)
/dev/mtdblock0 on /mnt/f2fs_test type f2fs_lecture (rw)
```

    - "# df –Th"

```
Filesystem     Type        Size  Used Avail Use% Mounted on
/dev/sda1      ext4        222G   87G  124G  42% /
udev           devtmpfs    3.8G  4.0K  3.8G   1% /dev
tmpfs          tmpfs       774M  832K  773M   1% /run
none           tmpfs       5.0M     0  5.0M   0% /run/lock
none           tmpfs       3.8G  144K  3.8G   1% /run/shm
/dev/rxd0      vfat_lecture  10G  8.0K   10G   1% /mnt/fat_test
/dev/mtdblock0 f2fs_lecture 4.0G  295M  3.7G   8% /mnt/f2fs_test
```


3. Linux Tracing Tools

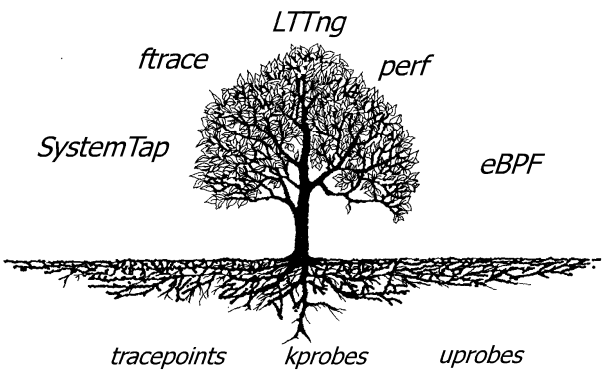
Kernel Tracing 실습

- Linux Tracing 실습의 목표
  - ✓ Linux 내부의 tracing framework 사용 구현 방법을 익힌다.
  - ✓ debugging 및 profiling 을 위한 tracing tool을 사용해 본다.
  - ✓ tracing tool 에 제공하는 기능들을 사용하여 커널이 동작하는 함수들의 흐름을 분석해 본다.

- Linux Tracing 실습의 내용
  - ✓ **파일 시스템 내의 분석하려는 위치에** Linux 내부의 Tracing framework 인 **tracepoint 를 삽입하여 커널 내부의 이벤트를 trace 해본다.**
  - ✓ 다양한 trace tool 을 통해 생성한 tracepoint 의 동작 방식을 분석한다.

Kernel Tracing Tools

- Linux Kernel Tracing
  - ✓ 복잡한 커널을 이해하기 위한 도구로써 커널 함수들의 흐름 및 발생하는 event들을 확인 할 수 있으며 system crash, page fault 등의 원인을 분석하기 위한 debugging 용도로 활용 될 수 있다.
  - ✓ 본 실습에서는 리눅스 내부의 kernel tracing mechanism 을 소개하고 이를 이용하여 Kernel 의 동작을 분석하는 방법을 알아본다.
  - ✓ Linux Tracing Tool tree.
    - trace source : tracepoints, kprobes, uprobes
    - trace tools : ftrace, SystemTap, LTTng, perf …



Kernel Tracing Tools

- Tracepoints
  - ✓ Linux-v2.6.32 이후 부터 도입 및 커널 코드 내의 원하는 부분에 callback 함수를 생성하여 event 를 모니터링, 디버깅에 사용 가능하다.
  - ✓ ftrace, perf, LTTng, SystemTap 등 다양한 tracing tool 에서 사용된다.
  - ✓ debugfs 를 통해 user space 로부터 값을 입력 받고, tracing 결과를 ring buffer 에 저장하여 user space 에 보여준다.
  - ✓ TRACE_EVENT 매크로에 작성한 내용에 따라 자동으로 callback 함수가 생성되며 event 를 사용할 시, ring buffer 에 저장된다.
    - jbd2_journal_commit_transaction 의 tracepoints (kernel/include/trace/events/jbd2.h)
      - trace_jbd2_start_comit
      - trace_jbd2_commit_logging
      - …

Kernel Tracing Tools

- 커널의 Tracepoints 예시
  - ✓ tracepoint function

    (kernel/fs/jbd2/transaction.c)
    - 커널 함수 내부에서 tracepoint 를 생성하려 할 경우, 해당 소스 파일에 사용 할, tracepoint 의 위치를 지정한다.
    - tracepoint 가 정의된 headerfile 을 해당 소스파일 내 include 한다.

```
33 #include <trace/events/jbd2.h>
```

  - ✓ CREATE_TRACE_POINTS 정의

    (kernel/fs/jbd2/journal.c)
    - tracepoint 가 enable 될 시, 생성한 trace point 함수가 callback 함수로 설정 되기 위해 CREATE_TRACE_POINTS를 1번 정의한다.

```
47 #define CREATE_TRACE_POINTS
48 #include <trace/events/jbd2.h>
```

```
414 handle_t *jbd2__journal_start(journal_t *journal, int nblocks, int rsv_blocks,
415                               gfp_t gfp_mask, unsigned int type,
416                               unsigned int line_no)
417 {
418     handle_t *handle = journal_current_handle();
419     int err;
420
421     if (!journal)
422         return ERR_PTR(-EROFS);
423
424     if (handle) {
425         J_ASSERT(handle->h_transaction->t_journal == journal);
426         handle->h_ref++;
427         return handle;
428     }
429
430     handle = new_handle(nblocks);
431     if (!handle)
432         return ERR_PTR(-ENOMEM);
433     if (rsv_blocks) {
434         handle_t *rsv_handle;
435
436         rsv_handle = new_handle(rsv_blocks);
437         if (!rsv_handle) {
438             jbd2_free_handle(handle);
439             return ERR_PTR(-ENOMEM);
440         }
441         rsv_handle->h_reserved = 1;
442         rsv_handle->h_journal = journal;
443         handle->h_rsv_handle = rsv_handle;
444     }
445
446     err = start_this_handle(journal, handle, gfp_mask);
447     if (err < 0) {
448         if (handle->h_rsv_handle)
449             jbd2_free_handle(handle->h_rsv_handle);
450         jbd2_free_handle(handle);
451         return ERR_PTR(err);
452     }
453     handle->h_type = type;
454     handle->h_line_no = line_no;
455     trace_jbd2_handle_start(journal->j_fs_dev->bd_dev,
456                             handle->h_transaction->t_tid, type,
457                             line_no, nblocks);
458     return handle;
459 }
460 EXPORT_SYMBOL(jbd2__journal_start);
```

Kernel Tracing Tools

- 커널에서의 Tracepoints 예시
  - ✓ Tracepoint 를 정의하기 위해 정해진 포맷에 맞추어 kernel/include/trace/events 경로에 tracepoint header file을 생성해 주어야 한다.

kernel/include/trace/define_trace.h

```
#define TRACE_EVENT(name, proto, args, tstruct, assign, print)  \
    DEFINE_TRACE(name)
```

kernel/include/trace/events/jbd2.h

```
TRACE_EVENT(jbd2_handle_start,
    TP_PROTO(dev_t dev, unsigned long tid, unsigned int type,
         unsigned int line_no, int requested_blocks),

    TP_ARGS(dev, tid, type, line_no, requested_blocks),

    TP_STRUCT__entry(
        __field(        dev_t,  dev     )
        __field(    unsigned long,  tid     )
        __field(     unsigned int,  type        )
        __field(     unsigned int,  line_no     )
        __field(          int,  requested_blocks)
    ),

    TP_fast_assign(
        __entry->dev          = dev;
        __entry->tid          = tid;
        __entry->type         = type;
        __entry->line_no      = line_no;
        __entry->requested_blocks = requested_blocks;
    ),

    TP_printk("dev %d,%d tid %lu type %u line_no %u "
          "requested_blocks %d",
          MAJOR(__entry->dev), MINOR(__entry->dev), __entry->tid,
          __entry->type, __entry->line_no, __entry->requested_blocks)
);
```

- name : 생성할 tracepoint 의 이름, 실제 tracepoint 는 prefix(trace_)가 붙어져 callback 함수 생성됨

Kernel Tracing Tools

- 커널에서의 Tracepoints 예시
  - ✓ Tracepoint 를 정의하기 위해 정해진 포맷에 맞추어 kernel/include/trace/events 경로에 tracepoint header file을 생성해 주어야 한다.

kernel/include/trace/define_trace.h

```
#define TRACE_EVENT(name, proto, args, tstruct, assign, print)  \
    DEFINE_TRACE(name)
```

kernel/include/trace/events/jbd2.h

```
TRACE_EVENT(jbd2_handle_start,
    TP_PROTO(dev_t dev, unsigned long tid, unsigned int type,
         unsigned int line_no, int requested_blocks),

    TP_ARGS(dev, tid, type, line_no, requested_blocks),

    TP_STRUCT__entry(
        __field(        dev_t,  dev     )
        __field(    unsigned long,  tid     )
        __field(     unsigned int,  type        )
        __field(     unsigned int,  line_no     )
        __field(          int,  requested_blocks)
    ),

    TP_fast_assign(
        __entry->dev          = dev;
        __entry->tid          = tid;
        __entry->type         = type;
        __entry->line_no      = line_no;
        __entry->requested_blocks = requested_blocks;
    ),

    TP_printk("dev %d,%d tid %lu type %u line_no %u "
          "requested_blocks %d",
          MAJOR(__entry->dev), MINOR(__entry->dev), __entry->tid,
          __entry->type, __entry->line_no, __entry->requested_blocks)
);
```

- name : 생성할 tracepoint 의 이름, 실제 tracepoint 는 prefix(trace_)가 붙어져 callback 함수 생성됨

- proto : TP_PROTO 매크로를 통해 callback 함수의 prototype 지정

Kernel Tracing Tools

- 커널에서의 Tracepoints 예시
  - ✓ Tracepoint 를 정의하기 위해 정해진 포맷에 맞추어 kernel/include/trace/events 경로에 tracepoint header file을 생성해 주어야 한다.

kernel/include/trace/define_trace.h

```
#define TRACE_EVENT(name, proto, args, tstruct, assign, print)  \
    DEFINE_TRACE(name)
```

kernel/include/trace/events/jbd2.h

```
TRACE_EVENT(jbd2_handle_start,
    TP_PROTO(dev_t dev, unsigned long tid, unsigned int type,
         unsigned int line_no, int requested_blocks),

    TP_ARGS(dev, tid, type, line_no, requested_blocks),

    TP_STRUCT__entry(
        __field(        dev_t,  dev     )
        __field(    unsigned long,  tid     )
        __field(     unsigned int,  type        )
        __field(     unsigned int,  line_no     )
        __field(          int,  requested_blocks)
    ),

    TP_fast_assign(
        __entry->dev          = dev;
        __entry->tid          = tid;
        __entry->type         = type;
        __entry->line_no      = line_no;
        __entry->requested_blocks = requested_blocks;
    ),

    TP_printk("dev %d,%d tid %lu type %u line_no %u "
          "requested_blocks %d",
          MAJOR(__entry->dev), MINOR(__entry->dev), __entry->tid,
          __entry->type, __entry->line_no, __entry->requested_blocks)
);
```

- name : 생성할 tracepoint 의 이름, 실제 tracepoint 는 prefix(trace_)가 붙어져 callback 함수 생성됨
- proto : TP_PROTO 매크로를 통해 callback 함수의 prototype 지정
- args : TP_ARGS 매크로를 통해 함수의 매개변수 이름 지정

```
#define TRACE_POINT(name, proto, args) \
  void trace_##name(proto)             \
  {                                    \
        if (trace_##name##_active)     \
              callback(args);          \
  }
```


Kernel Tracing Tools

- 커널에서의 Tracepoints 예시
  - ✓ Tracepoint 를 정의하기 위해 정해진 포맷에 맞추어 kernel/include/trace/events 경로에 tracepoint header file을 생성해 주어야 한다.

kernel/include/trace/define_trace.h

```
#define TRACE_EVENT(name, proto, args, tstruct, assign, print)  \
    DEFINE_TRACE(name)
```

kernel/include/trace/events/jbd2.h

```
TRACE_EVENT(jbd2_handle_start,
    TP_PROTO(dev_t dev, unsigned long tid, unsigned int type,
         unsigned int line_no, int requested_blocks),

    TP_ARGS(dev, tid, type, line_no, requested_blocks),

    TP_STRUCT__entry(
        __field(        dev_t,  dev     )
        __field(    unsigned long,  tid     )
        __field(     unsigned int,  type        )
        __field(     unsigned int,  line_no     )
        __field(          int,  requested_blocks)
    ),

    TP_fast_assign(
        __entry->dev          = dev;
        __entry->tid          = tid;
        __entry->type         = type;
        __entry->line_no      = line_no;
        __entry->requested_blocks = requested_blocks;
    ),

    TP_printk("dev %d,%d tid %lu type %u line_no %u "
          "requested_blocks %d",
          MAJOR(__entry->dev), MINOR(__entry->dev), __entry->tid,
          __entry->type, __entry->line_no, __entry->requested_blocks)
);
```

- name : 생성할 tracepoint 의 이름, 실제 tracepoint 는 prefix(trace_)가 붙어져 callback 함수 생성됨
- proto : TP_PROTO 매크로를 통해 callback 함수의 prototype 지정
- args : TP_ARGS 매크로를 통해 함수의 매개변수 이름 지정
- tstruct : TP_STRUCT__entry 매크로를 통해 user space 에 보여질 ring buffe에 저장할 형식의 구조체 생성.
  - __field() : 일반 변수 정의
  - __aray() : 배열 변수 정의
  - , 로 자료형, 이름 구분

Kernel Tracing Tools

- 커널에서의 Tracepoints 예시
  - ✓ Tracepoint 를 정의하기 위해 정해진 포맷에 맞추어 kernel/include/trace/events 경로에 tracepoint header file을 생성해 주어야 한다.

kernel/include/trace/define_trace.h

```
#define TRACE_EVENT(name, proto, args, tstruct, assign, print)  \
    DEFINE_TRACE(name)
```

kernel/include/trace/events/jbd2.h

```
TRACE_EVENT(jbd2_handle_start,
    TP_PROTO(dev_t dev, unsigned long tid, unsigned int type,
         unsigned int line_no, int requested_blocks),

    TP_ARGS(dev, tid, type, line_no, requested_blocks),

    TP_STRUCT__entry(
        __field(        dev_t,  dev     )
        __field(    unsigned long,  tid     )
        __field(     unsigned int,  type        )
        __field(     unsigned int,  line_no     )
        __field(          int,  requested_blocks)
    ),

    TP_fast_assign(
        __entry->dev          = dev;
        __entry->tid          = tid;
        __entry->type         = type;
        __entry->line_no      = line_no;
        __entry->requested_blocks = requested_blocks;
    ),

    TP_printk("dev %d,%d tid %lu type %u line_no %u "
          "requested_blocks %d",
          MAJOR(__entry->dev), MINOR(__entry->dev), __entry->tid,
          __entry->type, __entry->line_no, __entry->requested_blocks)
);
```

- name : 생성할 tracepoint 의 이름, 실제 tracepoint 는 prefix(trace_)가 붙어져 callback 함수 생성됨
- proto : TP_PROTO 매크로를 통해 callback 함수의 prototype 지정
- args : TP_ARGS 매크로를 통해 함수의 매개변수 이름 지정
- tstruct : TP_STRUCT__entry 매크로를 통해 user space 에 보여질 ring buffe에 저장할 형식의 구조체 생성.
- assign : TP_ fast_assign 매크로를 통해 TP_STRUCT__entry 에서 생성한 구조체 초기화 방법 지정
  - __entry : TP_STRUCT_entry를 통해 생성한 구조체

Kernel Tracing Tools

- 커널에서의 Tracepoints 예시
  - ✓ Tracepoint 를 정의하기 위해 정해진 포맷에 맞추어 kernel/include/trace/events 경로에 tracepoint header file을 생성해 주어야 한다.

kernel/include/trace/define_trace.h

```
#define TRACE_EVENT(name, proto, args, tstruct, assign, print)  \
    DEFINE_TRACE(name)
```

kernel/include/trace/events/jbd2.h

```
TRACE_EVENT(jbd2_handle_start,
    TP_PROTO(dev_t dev, unsigned long tid, unsigned int type,
         unsigned int line_no, int requested_blocks),

    TP_ARGS(dev, tid, type, line_no, requested_blocks),

    TP_STRUCT__entry(
        __field(        dev_t,  dev     )
        __field(    unsigned long,  tid     )
        __field(     unsigned int,  type        )
        __field(     unsigned int,  line_no     )
        __field(          int,  requested_blocks)
    ),

    TP_fast_assign(
        __entry->dev          = dev;
        __entry->tid          = tid;
        __entry->type         = type;
        __entry->line_no      = line_no;
        __entry->requested_blocks = requested_blocks;
    ),

    TP_printk("dev %d,%d tid %lu type %u line_no %u "
          "requested_blocks %d",
          MAJOR(__entry->dev), MINOR(__entry->dev), __entry->tid,
          __entry->type, __entry->line_no, __entry->requested_blocks)
);
```

- name : 생성할 tracepoint 의 이름, 실제 tracepoint 는 prefix(trace_)가 붙어져 callback 함수 생성됨
- proto : TP_PROTO 매크로를 통해 callback 함수의 prototype 지정
- args : TP_ARGS 매크로를 통해 함수의 매개변수 이름 지정
- tstruct : TP_STRUCT__entry 매크로를 통해 user space 에 보여질 ring buffer 에 저장할 형식의 구조체 생성.
- assign : TP_ fast_assign 매크로를 통해 TP_STRUCT__entry 에서 생성한 구조체 초기화 방법 지정
- print : TP_ printk 매크로를 통해 TP_STRUCT__entry 에서 생성한 구조체 초기화의 ring buffer 로의 출력 형식 지정

Kernel Tracing Tools

- 커널에서의 Tracepoints 예시
  - ✓ Tracepoint 를 정의하기 위해 정해진 포맷에 맞추어 kernel/include/trace/events 경로에 tracepoint header file을 생성해 주어야 한다.

kernel/include/trace/events/jbd2.h

```
1 #undef TRACE_SYSTEM
2 #define TRACE_SYSTEM jbd2
3
4 #if !defined(_TRACE_JBD2_H) || defined(TRACE_HEADER_MULTI_READ)
5 #define _TRACE_JBD2_H
6
7 #include <linux/jbd2.h>
8 #include <linux/tracepoint.h>
```

```
382 #endif /* _TRACE_JBD2_H */
383
384 /* This part must be outside protection */
385 #include <trace/define_trace.h>
```

- TRACE_EVENT macro 가 속한 group 을 정의
- debugfs 가 mount된 경로의 tracing/events 위치에 해당 그룹의 directory를 생성

```
root@fslecture-VirtualBox:/sys/k el/debug/tracing# ls events/
block            fence            der_page   kmem     nmi      ras          sched    spi      tlb      xen
compaction       filelock         e          mce      oom      raw_syscalls scsi     swiotlb  udp      xhci-hcd
context_tracking filemap          mu         migrate  pagemap  rcu          signal   syscalls vmscan
enable           ftrace           y          module   power    regmap       skb      task     vsyscall
exceptions       gpio             irq_vectors napi    printk   regulator    snd_pcm  thermal  workqueue
ext4             header_event     jbd2       net      random   rpm          sock     timer    writeback
```


Kernel Tracing Tools

- 커널에서의 Tracepoints 예시
  - ✓ Tracepoint 를 정의하기 위해 정해진 포맷에 맞추어 kernel/include/trace/events 경로에 tracepoint header file을 생성해 주어야 한다.

kernel/include/trace/events/jbd2.h

```
1 #undef TRACE_SYSTEM
2 #define TRACE_SYSTEM jbd2
3
4 #if !defined(_TRACE_JBD2_H) || defined(TRACE_HEADER_MULTI_READ)
5 #define _TRACE_JBD2_H
6
7 #include <linux/jbd2.h>
8 #include <linux/tracepoint.h>
```


```
382 #endif /* _TRACE_JBD2_H */
383
384 /* This part must be outside protection */
385 #include <trace/define_trace.h>
```

- TRACE_EVENT macro 가 속한 group 을 정의
- debugfs 가 mount된 경로의 tracing/events 위치에 해당 그룹의 directory를 생성

- 여러 파일에서 header file을 사용하기 위해 TRACE_HEADER_MULTI_READ를 정의


```
root@fslecture-VirtualBox:/sys/k el/debug/tracing# ls events/
block            fence            der_page   kmem     nmi      ras          sched    spi      tlb      xen
compaction       filelock         e          mce      oom      raw_syscalls scsi     swiotlb  udp      xhci-hcd
context_tracking filemap          mu         migrate  pagemap  rcu          signal   syscalls vmscan
enable           ftrace           y          module   power    regmap       skb      task     vsyscall
exceptions       gpio             irq_vectors napi    printk   regulator    snd_pcm  thermal  workqueue
ext4             header_event     jbd2       net      random   rpm          sock     timer    writeback
```

Kernel Tracing Tools

- 커널에서의 Tracepoints 예시
  - ✓ Tracepoint 를 정의하기 위해 정해진 포맷에 맞추어 kernel/include/trace/events 경로에 tracepoint header file을 생성해 주어야 한다.

kernel/include/trace/events/jbd2.h

```
#undef TRACE_SYSTEM
#define TRACE_SYSTEM jbd2

#if !defined(_TRACE_JBD2_H) || defined(TRACE_HEADER_MULTI_READ)
#define _TRACE_JBD2_H

#include <linux/jbd2.h>
#include <linux/tracepoint.h>
```

```
#endif /* _TRACE_JBD2_H */

/* This part must be outside protection */
#include <trace/define_trace.h>
```

- TRACE_EVENT macro 가 속한 group 을 정의
- debugfs 가 mount된 경로의 tracing/events 위치에 해당 그룹의 directory를 생성

- 여러 파일에서 header file을 사용하기 위해 TRACE_HEADER_MULTI_READ를 정의

- TRACE_EVENT 매크로를 사용하기 위해 tracepoint.h 와 define_traceh.h 를 include
- define_trace 를 #endif 외부에 정의

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# ls events/
block            fence            header_page  kmem     nmi      ras           sched    spi      tlb      xen
compaction       filelock         irqsoff      mce      oom      raw_syscalls  scsi     swiotlb  udp      xhci-hcd
context_tracking filemap          kmem         migrate  pagemap  rcu           signal   syscalls vmscan
enable           ftrace           kvmmmu       module   power    regmap        skb      task     vsyscall
exceptions       gpio             irq_vectors  napi     printk   regulator     snd_pcm  thermal  workqueue
ext4             header_event     jbd2         net      random   rpm           sock     timer    writeback
```


Kernel Tracing Tools

- 기본 경로가 아닌 곳에서의 Tracepoint
  - ✓ TRACE_EVENT 를 생성한 header file이 default 경로 (kernel/include/trace/events) 에 있을 수 있는 커널 모듈 등의 경우 추가적인 작성이 필요하다.
  - ✓ xfs 는 TRACE_EVENT 가 작성된 xfs_trace.h 를 기본 경로가 아닌 xfs source directory 내에 두어 다음과 같이 추가적인 작성이 되어 있다.
    - trace header file 에서의 추가 사항.

kernel/fs/xfs/xfs_trace.h

```
#undef TRACE_INCLUDE_PATH
#define TRACE_INCLUDE_PATH .
#define TRACE_INCLUDE_FILE xfs_trace
#include <trace/define_trace.h>
```

- TRACE_INCLUDE_PATH 를 현재 디렉터리 위치로 지정
- TRACE_INCLUDE_FILE 를 통해 trace header file 의 이름 지정

- Makefile 에서의 추가 사항.

```
ccflags-y += -I$(src)          # needed for trace events
```

- trace header file 경로 지정

Kernel Tracing Tools

- ftrace
  - ✓ Kernel Tracing Tool 들 중, 가장 먼저 도입된 trace framework 로 2.6.27 이후부터 도입되었다.
  - ✓ kprobe, uprobe, tracepoint 등을 기반으로 하여 리눅스 커널을 tracing 하기 위한 다양한 framework 을 제공하며 debugging, performance tuning, data gathering 등에 사용될 수 있다.
  - ✓ mcount() 를 커널 함수의 시작 지점에 삽입하여 커널 내의 각 함수들을 모니터링 한다.
  - ✓ ftrace 기반의 다양한 front-end tool 을 제공한다.
    - kernel-shark, trace-cmd
  - ✓ 다양한 tool 들이 ftrace를 기반으로 동작한다.
    - iosnoop, tpoint, opensnoop, execsnoop, funccount, funcslower …

Kernel Tracing Tools

- ftrace
  - ✓ ftrace interface

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# ls
README                    enabled_functions         options              set_ftrace_pid       trace_clock          tracing_thresh
available_events          events                    per_cpu              set_graph_function   trace_marker         uprobe_events
available_filter_functions free_buffer              printk_formats       set_graph_notrace    trace_options        uprobe_profile
available_tracers         function_profile_enabled  saved_cmdlines      snapshot             trace_pipe
buffer_size_kb            instances                 saved_cmdlines_size  stack_max_size       trace_stat
buffer_total_size_kb      kprobe_events             set_event            stack_trace          tracing_cpumask
current_tracer            kprobe_profile            set_ftrace_filter    stack_trace_filter   tracing_max_latency
dyn_ftrace_total_info     max_graph_depth           set_ftrace_notrace   trace                tracing_on
```

    - available_tracers : 설정 가능한 tracer 목록.
    - current_tracers : 현재 tracer 설정 및 조회.
    - per_cpu : 각 cpu 별 buffer 별 buffer 및 trace 설정.
    - available_filter_functions : tracing 가능한 함수 목록(mcount 설정 가능한 함수)
    - set_ftrace_filter / set_ftrace_notrace : tracing 을 할 함수 지정 및 제외할 함수 지정
    - set_graph_filter / set_graph_notrace : tracing 이 시작될 함수를 지정 및 제외할 함수 지정.
    - trace : tracing 결과가 기록된 ring buffer 내용을 출력
    - events : tracepoint 를 통해 설정한 event
    - trace_on/off : tracing 기능과 무관하게 ring buffer 로의 출력을 관리
    - trace_options : trace 결과에 출력될 값 추가 및 제거
  - ✓ ftrace tracers

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat available_tracers
blk mmiotrace function_graph wakeup_dl wakeup_rt wakeup function nop
```

    - blk : block tracer (blktrace 에서 사용된다.)
    - mmiotrace : binary module 분석 tracer.
    - function_graph ;그래프 형식의 kernel function 분석 tracer
    - wakeup_dl : deadline scheduler 에서의 max latency task tracer.
    - wakeup_rt : real time task 들의 latency tracer.
    - wakeup : highest priority task 의 wake up latency tracer.
    - function : kernel function 분석 tracer

Kernel Tracing Tools

- ftrace
  - ✓ ftrace function tracer
    - 현재 tracer 를 function tracer 로 설정 및 trace 결과 확인

      # echo function > current_tracer

      # cat trace

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo function > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat current_tracer
function
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace | head -20
# tracer: function
#
# entries-in-buffer/entries-written: 205160/1994895   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
          gdbus-2173  [001] d... 109263.629696: account_system_time <-__vtime_account_system
          gdbus-2173  [001] d... 109263.629697: cpuacct_account_field <-account_system_time
          gdbus-2173  [001] d... 109263.629699: acct_account_cputime <-account_system_time
          gdbus-2173  [001] d... 109263.629701: __acct_update_integrals <-acct_account_cputime
          gdbus-2173  [001] d... 109263.629702: jiffies_to_timeval <-__acct_update_integrals
          gdbus-2173  [001] d... 109263.629704: _raw_spin_unlock <-vtime_user_enter
          gdbus-2173  [001] d... 109263.629706: rcu_user_enter <-context_tracking_user_enter
          gdbus-2173  [001] d... 109263.629707: rcu_eqs_enter <-rcu_user_enter
          gdbus-2173  [001] d... 109263.629709: rcu_eqs_enter_common.isra.52 <-rcu_eqs_enter
```

- entries-in-buffer/entries-written : 현재 buffer 에 기록된 trace 수/누적된 trace 결과의 수(차이가 buffer 를 넘어 지워진 결과의 수를 의미)
- P : 실행한 processor 의 수

Kernel Tracing Tools

- ftrace
  - ✓ ftrace function tracer
    - 현재 tracer 를 function tracer 로 설정 및 trace 결과 확인

      # echo function > current_tracer

      # cat trace

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo function > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat current_tracer
function
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace | head -20
# tracer: function
#
# entries-in-buffer/entries-written: 205160/1994895   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
          gdbus-2173  [001] d... 109263.629696: account_system_time <-__vtime_account_system
          gdbus-2173  [001] d... 109263.629697: cpuacct_account_field <-account_system_time
          gdbus-2173  [001] d... 109263.629699: acct_account_cputime <-account_system_time
          gdbus-2173  [001] d... 109263.629701: __acct_update_integrals <-acct_account_cputime
          gdbus-2173  [001] d... 109263.629702: jiffies_to_timeval <-__acct_update_integrals
          gdbus-2173  [001] d... 109263.629704: _raw_spin_unlock <-vtime_user_enter
          gdbus-2173  [001] d... 109263.629706: rcu_user_enter <-context_tracking_user_enter
          gdbus-2173  [001] d... 109263.629707: rcu_eqs_enter <-rcu_user_enter
          gdbus-2173  [001] d... 109263.629709: rcu_eqs_enter_common.isra.52 <-rcu_eqs_enter
```

- entries-in-buffer/entries-written : 현재 buffer 에 기록된 trace 수/누적된 trace 결과의 수(차이가 buffer 를 넘어 지워진 결과의 수를 의미)
- P : 실행한 processor 의 수

- TASK-PID : 해당 함수가 tack 이름 및 process id
- CPU : 함수가 실행된 core number
- latency format 으로 irq 관련 tracer 에서 출력

Kernel Tracing Tools

- ftrace
  - ✓ ftrace function tracer
    - 현재 tracer 를 function tracer 로 설정 및 trace 결과 확인

      # echo function > current_tracer

      # cat trace

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo function > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat current_tracer
function
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace | head -20
# tracer: function
#
# entries-in-buffer/entries-written: 205160/1994895   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
          gdbus-2173  [001] d... 109263.629696: account_system_time <-__vtime_account_system
          gdbus-2173  [001] d... 109263.629697: cpuacct_account_field <-account_system_time
          gdbus-2173  [001] d... 109263.629699: acct_account_cputime <-account_system_time
          gdbus-2173  [001] d... 109263.629701: __acct_update_integrals <-acct_account_cputime
          gdbus-2173  [001] d... 109263.629702: jiffies_to_timeval <-__acct_update_integrals
          gdbus-2173  [001] d... 109263.629704: _raw_spin_unlock <-vtime_user_enter
          gdbus-2173  [001] d... 109263.629706: rcu_user_enter <-context_tracking_user_enter
          gdbus-2173  [001] d... 109263.629707: rcu_eqs_enter <-rcu_user_enter
          gdbus-2173  [001] d... 109263.629709: rcu_eqs_enter_common.isra.52 <-rcu_eqs_enter
```

- entries-in-buffer/entries-written : 현재 buffer 에 기록된 trace 수/누적된 trace 결과의 수(차이가 buffer 를 넘어 지워진 결과의 수를 의미)
- P : 실행한 processor 의 수

- TASK-PID : 해당 함수가 tack 이름 및 process id
- CPU : 함수가 실행된 core number
- latency format 으로 irq 관련 tracer 에서 출력(d : interrupt disable)

Kernel Tracing Tools

- ftrace
  - ✓ ftrace function tracer
    - 현재 tracer 를 function tracer 로 설정 및 trace 결과 확인

      # echo function > current_tracer

      # cat trace

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo function > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat current_tracer
function
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace | head -20
# tracer: function
#
# entries-in-buffer/entries-written: 205160/1994895   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
          gdbus-2173  [001] d... 109263.629696: account_system_time <-__vtime_account_system
          gdbus-2173  [001] d... 109263.629697: cpuacct_account_field <-account_system_time
          gdbus-2173  [001] d... 109263.629699: acct_account_cputime <-account_system_time
          gdbus-2173  [001] d... 109263.629701: __acct_update_integrals <-acct_account_cputime
          gdbus-2173  [001] d... 109263.629702: jiffies_to_timeval <-__acct_update_integrals
          gdbus-2173  [001] d... 109263.629704: _raw_spin_unlock <-vtime_user_enter
          gdbus-2173  [001] d... 109263.629706: rcu_user_enter <-context_tracking_user_enter
          gdbus-2173  [001] d... 109263.629707: rcu_eqs_enter <-rcu_user_enter
          gdbus-2173  [001] d... 109263.629709: rcu_eqs_enter_common.isra.52 <-rcu_eqs_enter
```

- entries-in-buffer/entries-written : 현재 buffer 에 기록된 trace 수/누적된 trace 결과의 수(차이가 buffer 를 넘어 지워진 결과의 수를 의미)
- P : 실행한 processor 의 수

- TASK-PID : 해당 함수가 tack 이름 및 process id
- CPU : 함수가 실행된 core number
- latency format 으로 irq 관련 tracer 에서 출력

- TIME STAMP : 부팅 이후 TIMESTAMP 을 sec.usec 으로 출력
- FUNCTION : trace 된 function 이름 및 해당 함수를 호출한 부모 function 의 이름

Kernel Tracing Tools

- ftrace
  - ✓ ftrace graph function tracer
    - 현재 tracer 를 function graph tracer 로 설정 및 trace 결과 확인

      # echo function_graph > current_tracer

      # cat trace

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo function_graph > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat current_tracer
function_graph
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace | head -20
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 3) + 27.617 us   |        } /* account_system_time */
 3) + 52.156 us   |      } /* __vtime_account_system */
 3)   1.738 us    |      _raw_spin_unlock();
 3) + 71.301 us   |    } /* vtime_user_enter */
 3)               |    rcu_user_enter() {
 3)               |      rcu_eqs_enter() {
 3)   1.561 us    |        rcu_eqs_enter_common.isra.52();
 3)   7.320 us    |      }
 3) + 13.979 us   |    }
 3) + 95.875 us   |  } /* context_tracking_user_enter */
 3)               |  context_tracking_user_exit() {
 3)               |    rcu_user_exit() {
 3)               |      rcu_eqs_exit() {
 3)   1.214 us    |        rcu_eqs_exit_common.isra.50();
 3)   7.812 us    |      }
 3) + 13.977 us   |    }
```

      - CPU : 해당 함수가 실행된 core number
      - DURATION : 함수에서 소요된 시간(누적된 시간을 표시하므로 해당 함수에서만 소요된 시간을 보려면 option 에서 graph-time 을 설정해 주면 된다)
        + : 10 micro second 이상 소요 함수 / ! : 100 micro second 이상 소요 함수
  - ✓ nop function tracer
    - 현재 tracer 를 nop 으로 설정하여 trace 중지

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo nop > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat current_tracer
nop
```


Kernel Tracing Tools

- ftrace
  - ✓ ftrace function filter
    - 커널에서 수행되는 함수들을 모두 tracing 하는 것은 의미가 없다. ftrace 의 set_filter_functions 기능을 통해 tracing 할 함수를 지정 및 tracing 하지 않을 함수 지정을 수행할 수 있다.
    - mcount 를 삽입 가하여 trace 가능한 함수는 available_filtr_function 을 통해 확인 할 수 있다. 32609 개의 trace 가능한 function 을 확인 할 수 있다.
  - ✓

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat available_filter_functions | wc -l
32609
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat available_filter_functions | head -20
run_init_process
try_to_run_init_process
do_one_initcall
match_dev_by_uuid
rootfs_mount
name_to_dev_t
init_linuxrc
init_linuxrc
calibration_delay_done
calibrate_delay
xen_cpuid
xen_set_debugreg
xen_get_debugreg
xen_store_tr
xen_io_delay
xen_get_apic_id
xen_apic_read
xen_apic_icr_read
xen_apic_wait_icr_idle
xen_safe_apic_wait_icr_idle
```

Kernel Tracing Tools

- ftrace
  - ✓ ftrace function filter
    - ext4 의 commit 을 수행하는 함수인 jbd2_journal_commit_transaction 함수를 trace 하기 위해 set_filter_filter 에 해당 함수 trace 설정 후, function tracer 로 확인 한 결과는 아래와 같다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo jbd2_journal_commit_transaction > set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo function > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# fio --output=/home/fs-lecture/lecture/result/test.txt --directo
y=/mnt/ --name=test --rw=randwrite --size=512K
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace
# tracer: function
#
# entries-in-buffer/entries-written: 8/8   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
     jbd2/sda1-8-270   [002] .... 151425.446210: jbd2_journal_commit_transaction <-kjournald2
     jbd2/sda1-8-270   [000] .... 151458.508099: jbd2_journal_commit_transaction <-kjournald2
     jbd2/rxd0-8-20002 [001] .... 151467.621182: jbd2_journal_commit_transaction <-kjournald2
     jbd2/sda1-8-270   [001] .... 151468.502028: jbd2_journal_commit_transaction <-kjournald2
     jbd2/rxd0-8-20002 [002] .... 151487.915748: jbd2_journal_commit_transaction <-kjournald2
     jbd2/sda1-8-270   [002] .... 151488.508176: jbd2_journal_commit_transaction <-kjournald2
     jbd2/rxd0-8-20002 [001] .... 151494.530826: jbd2_journal_commit_transaction <-kjournald2
     jbd2/sda1-8-270   [003] .... 151502.508818: jbd2_journal_commit_transaction <-kjournald2
```


Kernel Tracing Tools

- ftrace
  - ✓ ftrace function filter
    - ext4 의 commit 을 수행하는 함수인 jbd2_journal_commit_transaction 함수를 trace 하기 위해 set_filter_filter 에 해당 함수 trace 설정 후, function tracer 로 확인 한 결과는 아래와 같다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo jbd2_journal_commit_transaction > set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo function_graph > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# fio --output=/home/fs-lecture/lecture/result/test.txt --director
y=/mnt/ --name=test --rw=randwrite --size=512K
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 2)               |  jbd2_journal_commit_transaction() {
 0) # 3689.523 us |  } /* jbd2_journal_commit_transaction */
 0)               |  jbd2_journal_commit_transaction() {
 ------------------------------------------
 0)  jbd2/sd-270   => jbd2/rx-20002
 ------------------------------------------

 0) + 31.724 us   |  jbd2_journal_commit_transaction();
 ------------------------------------------
 0) jbd2/rx-20002  =>  jbd2/sd-270
 ------------------------------------------

 0) # 2887.458 us |  } /* jbd2_journal_commit_transaction */
 ------------------------------------------
 0)  jbd2/sd-270   => jbd2/rx-20002
 ------------------------------------------

 0) + 50.727 us   |  jbd2_journal_commit_transaction();
 1)               |  jbd2_journal_commit_transaction() {
 ------------------------------------------
 0) jbd2/rx-20002  =>  jbd2/sd-270
 ------------------------------------------

 0) # 2522.368 us |  } /* jbd2_journal_commit_transaction */
 2)               |  jbd2_journal_commit_transaction() {
 0) # 2796.308 us |  } /* jbd2_journal_commit_transaction */
 3) + 58.527 us   |  jbd2_journal_commit_transaction();
```

    - 아래와 같이 set_ftrace_filter 에서 trace 할 함수를 지울 수 있다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo nop > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo > set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat set_ftrace_filter
#### all functions enabled ####
```

Kernel Tracing Tools

- ftrace
  - ✓ ftrace function filter
    - set_ftrace_filter 에 등록 함수를 추가 하려 할 시, >> redirection 을 사용하여 추가가 가능하며, 특정 function 을 삭제하려 할 시, ! prefix를 붙여 삭제가 가능 하다.
    - 아래 예시는 commit 이 수행될 시, metadata buffer 내용을 write 하는 함수가 호출되는 모습이다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo nop > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo > set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo jbd2_journal_commit_transaction > set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo jbd2_journal_write_metadata_buffer >> set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo jbd2_journal_start >> set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat set_ftrace_filter
jbd2_journal_start
jbd2_journal_commit_transaction
jbd2_journal_write_metadata_buffer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo '!jbd2_journal_start' >> set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat set_ftrace_filter
jbd2_journal_commit_transaction
jbd2_journal_write_metadata_buffer
```

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace
# tracer: function
#
# entries-in-buffer/entries-written: 19/19   #P:4
#
#                                _-----=> irqs-off
#                               / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth
#                              ||| /     delay
#           TASK-PID    CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |    ||||       |         |
     jbd2/sda1-8-270   [002] .... 161471.561495: jbd2_journal_commit_transaction <-kjournald2
     jbd2/sda1-8-270   [002] .... 161471.561512: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/sda1-8-270   [002] .... 161471.561515: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/sda1-8-270   [002] .... 161471.561518: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/sda1-8-270   [002] .... 161471.561520: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/sda1-8-270   [002] .... 161471.561522: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/sda1-8-270   [002] .... 161471.561523: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/rxd0-8-20002 [001] .... 161476.607045: jbd2_journal_commit_transaction <-kjournald2
     jbd2/rxd0-8-20002 [001] .... 161476.607063: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/rxd0-8-20002 [001] .... 161476.607065: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/rxd0-8-20002 [001] .... 161476.607067: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/rxd0-8-20002 [001] .... 161476.607068: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/rxd0-8-20002 [001] .... 161476.607071: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/sda1-8-270   [002] .... 161477.569338: jbd2_journal_commit_transaction <-kjournald2
     jbd2/sda1-8-270   [002] .... 161477.569360: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/sda1-8-270   [002] .... 161477.569363: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/sda1-8-270   [002] .... 161477.569365: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/rxd0-8-20002 [001] .... 161506.624404: jbd2_journal_commit_transaction <-kjournald2
     jbd2/rxd0-8-20002 [001] .... 161506.624448: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
```


Kernel Tracing Tools

- ftrace
  - ✓ ftrace function filter
    - set_ftrace_filter에 정규 표현식을 사용하여 다수의 trace 함수를 등록할 수 있다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo jbd2* > set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat set_ftrace_filter | wc -l
94
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat set_ftrace_filter | head -10
jbd2_journal_free_reserved
jbd2_journal_destroy_transaction_cache
jbd2_journal_free_transaction
jbd2__journal_restart
jbd2_journal_restart
jbd2__journal_start
jbd2_journal_start
jbd2_journal_extend
jbd2_journal_lock_updates
jbd2_journal_unlock_updates
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo function > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# fio --output=/home/fs-lecture/lecture/result/test.txt --directory=/
mnt/ --name=test --rw=randwrite --size=512K
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace | head -20
# tracer: function
#
# entries-in-buffer/entries-written: 5420/5420   #P:4
#
#                                _-----=> irqs-off
#                               / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth
#                              ||| /     delay
#           TASK-PID    CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |    ||||       |         |
    kworker/u8:0-20283 [000] .... 159209.180902: jbd2_journal_blocks_per_page <-ext4_writepages
    kworker/u8:0-20283 [000] .... 159209.180904: jbd2__journal_start <-__ext4_journal_start_sb
    kworker/u8:0-20283 [000] .... 159209.180909: jbd2_journal_free_transaction <-start_this_handle
    kworker/u8:0-20283 [000] .... 159209.180917: jbd2_journal_get_write_access <-__ext4_journal_get_write_access
    kworker/u8:0-20283 [000] .... 159209.180918: jbd2_journal_add_journal_head <-jbd2_journal_get_write_access
    kworker/u8:0-20283 [000] .... 159209.180920: jbd2_journal_grab_journal_head <-__jbd2_journal_file_buffer
    kworker/u8:0-20283 [000] .... 159209.180921: jbd2_journal_cancel_revoke <-do_get_write_access
    kworker/u8:0-20283 [000] .... 159209.180922: jbd2_journal_put_journal_head <-jbd2_journal_get_write_access
    kworker/u8:0-20283 [000] .... 159209.180923: jbd2_journal_get_write_access <-__ext4_journal_get_write_access
```

Kernel Tracing Tools

- ftrace
  - ✓ ftrace function filter
    - 표현식을 사용하여 filter 된 function 의 동작을 세부적으로 제어가 가능하다.

      function : command [: parameter]

      function 이 시작 할때, command 를 수행
      - function : trace filter function
      - command : traceon, traceoff, stackstace, snapshot, enable_event,disable_event 설정 가능.
      - parameter : parameter를 통해 function hit 시 수행되는 command 의 수를 설정 가능, command 가 event 관련 function 일 경우(enable_event, disable_event) parameter 를 아래와 같이 구성하여 function 에 도달 할 시, 특정 event 를 실행하거나 해제하도록 할 수 있다.

        <event sub system>:<event name>[:count]
    - 아래는 commit 이 발생했을 시, stacktrace command 를 수행하여, 어떤 함수들을 통해 commit 함수가 호출 되었는지를 확인 한 것이다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 'jbd2_journal_commit_transaction:stacktrace:1' > set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# fio --output=/home/fs-lecture/lecture/result/test.txt --directory=/mnt/ --n
ame=test --rw=randwrite --ioengine=sync --size=512K
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace
# tracer: nop
#
# entries-in-buffer/entries-written: 1/1   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
     jbd2/rxd0-8-20524 [000] .... 177015.836952: <stack trace>
 => jbd2_journal_commit_transaction
 => kjournald2
 => kthread
 => ret_from_fork
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo '!jbd2_journal_commit_transaction:stacktrace:1' > set_ftrace_filter
```


Kernel Tracing Tools

- ftrace
  - ✓ ftrace function trace start point
    - filter 를 통해 특정 함수만 filtering 외에 trace 의 시작 함수를 set_graph_function 함수로 지정하여 해당 함수에서 호출되는 함수들만 모니터링이 가능하다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo jbd2_journal_commit_transaction > set_graph_function
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo function_graph > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace | head -30
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 3)               |  jbd2_journal_commit_transaction() {
 3)   9.283 us    |    journal_tag_bytes();
 3)   4.803 us    |    _raw_write_lock();
 3)   4.951 us    |    _raw_spin_lock();
 3)   4.055 us    |    _raw_spin_unlock();
 3)   4.046 us    |    _raw_spin_lock();
 3)               |    __jbd2_journal_clean_checkpoint_list() {
 3)   4.400 us    |      journal_clean_one_cp_list();
 3)   4.005 us    |      journal_clean_one_cp_list();
 3) + 40.752 us   |    }
 3)   4.081 us    |    _raw_spin_unlock();
 3)   5.656 us    |    jbd2_clear_buffer_revoked_flags();
 3)   5.208 us    |    jbd2_journal_switch_revoke_table();
 3)   7.934 us    |    ktime_get();
 3)               |    __wake_up() {
 3)   3.986 us    |      _raw_spin_lock_irqsave();
 3)   3.988 us    |      __wake_up_common();
 3)   4.010 us    |      _raw_spin_unlock_irqrestore();
 3) + 52.935 us   |    }
 3)   3.956 us    |    _raw_spin_lock();
 3)   3.964 us    |    _raw_spin_unlock();
 3)   4.111 us    |    blk_start_plug();
 3)   5.946 us    |    jbd2_journal_write_revoke_records();
 3)   3.947 us    |    _raw_write_lock();
 3)               |    jbd2_journal_get_descriptor_buffer() {
 3)               |      jbd2_journal_next_log_block() {
```

Kernel Tracing Tools

- ftrace
  - ✓ ftrace function trace start point per core
    - ftrace 는 ring buffer 에 기록된 전체 tracing 의 결과 외에 core 별 buffer 를 통해 각 core 에서의 trace 결과와 snapshot, 통계등을 확인 할 수 있다.
    - per_cpu 를 통해 각 core 를 확인 할 수 있다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# ls per_cpu/
cpu0  cpu1  cpu2  cpu3
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# ls per_cpu/cpu0/
buffer_size_kb  snapshot  snapshot_raw  stats  trace  trace_pipe  trace_pipe_raw
```

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat per_cpu/cpu3/trace | head -20
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 3)               |  jbd2_journal_commit_transaction() {
 3)   9.283 us    |    journal_tag_bytes();
 3)   4.803 us    |    _raw_write_lock();
 3)   4.951 us    |    _raw_spin_lock();
 3)   4.055 us    |    _raw_spin_unlock();
 3)   4.046 us    |    _raw_spin_lock();
 3)               |    __jbd2_journal_clean_checkpoint_list() {
 3)   4.400 us    |      journal_clean_one_cp_list();
 3)   4.005 us    |      journal_clean_one_cp_list();
 3) + 40.752 us   |    }
 3)   4.081 us    |    _raw_spin_unlock();
 3)   5.656 us    |    jbd2_clear_buffer_revoked_flags();
 3)   5.208 us    |    jbd2_journal_switch_revoke_table();
 3)   7.934 us    |    ktime_get();
 3)               |    __wake_up() {
 3)   3.986 us    |      _raw_spin_lock_irqsave();
```


Kernel Tracing Tools

- ftrace
  - ✓ ftrace event interface (tracepoint 기반)
    - EVENT_TRACE 에서 생성한 tracepoint 를 기반으로 커널 로직이 특정 event 에 도달하였을 때, event 의 내용을 모니터링 하거나, 표현식을 사용하여 특정 명령(traceon, traceoff, event_enable, event_disable) 등을 수행하도록 설정 할 수 있다.
    - 기존 커널에서 정의된 tracepoint 는 각 tracepoint header file 에 의해 디렉터리 생성되고 디렉터리 내에 련 tracepoint 들이 관리된다.
    - 커널에 이미 정의된 1048 개의 event 목록을 available_event 를 통해 확인 할 수 있다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat available_events | wc -l
1048
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat available_events | head -10
fat_lecture:fat_ent_new_write
snd_pcm:hwptr
snd_pcm:xrun
snd_pcm:hw_ptr_error
xen:xen_cpu_set_ldt
xen:xen_cpu_write_gdt_entry
xen:xen_cpu_load_idt
xen:xen_cpu_write_idt_entry
xen:xen_cpu_write_ldt_entry
xen:xen_mmu_write_cr3
```

    - 각 event 디렉터리에는 event group(tracepoint group) 별로 event 들이 관리된다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# ls events/
block            fence        header_page  kmem     nmi      ras          sched    spi      tlb      xen
compaction       filelock     i2c          mce      oom      raw_syscalls scsi     swiotlb  udp      xhci-hcd
context_tracking filemap      iommu        migrate  pagemap  rcu          signal   syscalls vmscan
enable           ftrace       irq          module   power    regmap       skb      task     vsyscall
exceptions       gpio         irq_vectors  napi     printk   regulator    snd_pcm  thermal  workqueue
ext4             header_event jbd2         net      random   rpm          sock     timer    writeback
```

Kernel Tracing Tools

- ftrace
  - ✓ ftrace event interface (tracepoint 기반)
    - jbd2 의 경우 아래와 같은 tracepoint 들을 확인 할 수 있다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# ls events/jbd2/
enable                jbd2_commit_flushing   jbd2_end_commit     jbd2_lock_buffer_stall  jbd2_update_log_tail
filter                jbd2_commit_locking    jbd2_handle_extend  jbd2_run_stats          jbd2_write_superblock
jbd2_checkpoint       jbd2_commit_logging    jbd2_handle_start   jbd2_start_commit
jbd2_checkpoint_stats jbd2_drop_transaction  jbd2_handle_stats   jbd2_submit_inode_data
```

    - TRACE_EVENT 를 통해 생성된 event fomat 을 확인 할 수 있으며 각각의 event 들은 커널 코드 내에서 사용되는 것을 확인 할 수 있다.

```
trace_jbd2_start_commit(journal, commit_transaction);
```

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing/events/jbd2# ls jbd2_start_commit/
enable  filter  format  id  trigger
root@fslecture-VirtualBox:/sys/kernel/debug/tracing/events/jbd2# cat jbd2_start_commit/format
name: jbd2_start_commit
ID: 847
format:
        field:unsigned short common_type;       offset:0;       size:2; signed:0;
        field:unsigned char common_flags;       offset:2;       size:1; signed:0;
        field:unsigned char common_preempt_count;       offset:3;       size:1; signed:0;
        field:int common_pid;   offset:4;       size:4; signed:1;

        field:dev_t dev;        offset:8;       size:4; signed:0;
        field:char sync_commit; offset:12;      size:1; signed:1;
        field:int transaction;  offset:16;      size:4; signed:1;

print fmt: "dev %d,%d transaction %d sync %d", ((unsigned int) ((REC->dev) >> 20)), ((unsigned int) ((REC->dev) & ((1
U << 20) - 1))), REC->transaction, REC->sync_commit
```

      - enable : event 설정 해제
      - filter : ring buffer 에 기록된 common parameter, event specific parameter 로 event 선별 가능
      - *format* : TRACE_EVENT 를 통해 생성된 *해당 event 의 구조체 형식*
      - trigger : 커널 코드 내에 해당 event 에 도달할 시, 수행할 command 등 지정.

Kernel Tracing Tools

- ftrace
  - ✓ ftrace event interface (tracepoint 기반)
    - jbd2 commit 함수의 경우, 4개의 tracepoint 가 사용되고 있다.
      - trace_jbd2_start_commit, trace_jbd2_commit_locking, trace_jbd2_commit_flushing, trace_jbd2_commit_logging, trace_jbd2_run_stats, trace_jbd2_end_commit
    - set_function_filter 를 통해 filter function 을 jbd2_journal_commit_transaction, jbd2_journal_write_metadata_buffer 로 설정하고, event 확인을 통해 transaction 에서 journal 되는 handle 의 수, logging 되는 block 의 수 등을 확인해 볼 수 있다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo nop > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo > set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo jbd2_journal_commit_transaction > set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo jbd2_journal_write_metadata_buffer >> set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat set_ftrace_filter
jbd2_journal_commit_transaction
jbd2_journal_write_metadata_buffer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 1 > events/jbd2/jbd2_start_commit/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 1 > events/jbd2/jbd2_commit_locking/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 1 > events/jbd2/jbd2_commit_flushing/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 1 > events/jbd2/jbd2_commit_logging/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 1 > events/jbd2/jbd2_run_stats/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 1 > events/jbd2/jbd2_end_commit/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo function > current_tracer
```

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 > events/jbd2/j
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 > events/jbd2/j
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 > events/jbd2/j
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 > events/jbd2/j
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 > events/jbd2/j
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 > events/jbd2/j
root@fslecture-VirtualBox:/sys/kernel/debug/tracing#
root@fslecture-VirtualBox:/sys/kernel/debug/tracing#
```

Kernel Tracing Tools

- ftrace
  - ✓ ftrace event interface (tracepoint 기반)
    - jbd2 commit 함수의 경우, 4개의 tracepoint 가 사용되고 있다.
      - trace_jbd2_start_commit, trace_jbd2_commit_locking, trace_jbd2_commit_flushing, trace_jbd2_commit_logging, trace_jbd2_run_stats, trace_jbd2_end_commit
    - 아래와 같이 각 event 가 동작하도록 설정할 수 있다.

      # echo 1 > events/jbd2/jbd2_start_commit/enable

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo nop > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo > set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo jbd2_journal_commit_transaction > set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo jbd2_journal_write_metadata_buffer >> set_ftrace_filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat set_ftrace_filter
jbd2_journal_commit_transaction
jbd2_journal_write_metadata_buffer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 1 > events/jbd2/jbd2_start_commit/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 1 > events/jbd2/jbd2_commit_locking/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 1 > events/jbd2/jbd2_commit_flushing/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 1 > events/jbd2/jbd2_commit_logging/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 1 > events/jbd2/jbd2_run_stats/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 1 > events/jbd2/jbd2_end_commit/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo function > current_tracer
```

    - event 별로 각 event 동작을 멈출 수 있다.

      # echo 0 > events/jbd2/jbd2_start_commit/enable
    - jbd2 subsystem directory 에서 jbd2 의 모든 event 를 멈추도록 설정 할 수 있다.
    - ***# echo 0 > events/jbd2/enable***

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 > events/jbd2/jbd2_start_commit/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 > events/jbd2/jbd2_commit_locking/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 > events/jbd2/jbd2_commit_flushing/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 > events/jbd2/jbd2_commit_logging/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 > events/jbd2/jbd2_run_stats/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 > events/jbd2/jbd2_end_commit/enable
root@fslecture-VirtualBox:/sys/kernel/debug/tracing#
root@fslecture-VirtualBox:/sys/kernel/debug/tracing#
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 >  events/jbd2/enable
```


Kernel Tracing Tools

- ftrace
  - ✓ ftrace event interface (tracepoint 기반)
    - tracer 를 function 으로 설정한 결과는 아래와 같다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# fio --output=/home/fs-lecture/lecture/result/test.txt --directory=/
mnt/ --name=test --rw=randwrite --size=512K
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace
# tracer: function
#
# entries-in-buffer/entries-written: 23/23   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
     jbd2/rxd0-8-20524 [003] .... 164192.466572: jbd2_journal_commit_transaction <-kjournald2
     jbd2/rxd0-8-20524 [003] .... 164192.466574: jbd2_start_commit: dev 251,0 transaction 6 sync 0
     jbd2/rxd0-8-20524 [003] .... 164192.466576: jbd2_commit_locking: dev 251,0 transaction 6 sync 0
     jbd2/rxd0-8-20524 [003] .... 164192.466578: jbd2_commit_flushing: dev 251,0 transaction 6 sync 0
     jbd2/rxd0-8-20524 [003] .... 164192.466581: jbd2_commit_logging: dev 251,0 transaction 6 sync 0
     jbd2/rxd0-8-20524 [003] .... 164192.466590: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/rxd0-8-20524 [003] .... 164192.466592: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/rxd0-8-20524 [003] .... 164192.466594: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/rxd0-8-20524 [003] .... 164192.466596: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/rxd0-8-20524 [003] .... 164192.466597: jbd2_journal_write_metadata_buffer <-jbd2_journal_commit_transaction
     jbd2/rxd0-8-20524 [003] .... 164192.466611: jbd2_run_stats: dev 251,0 tid 6 wait 0 request_delay 0 running 5012 loc
ked 0 flushing 0 logging 0 handle_count 133 blocks 5 blocks_logged 6
     jbd2/rxd0-8-20524 [003] .... 164192.466614: jbd2_end_commit: dev 251,0 transaction 6 sync 0 head 2
```

Kernel Tracing Tools

- ftrace
  - ✓ ftrace event interface (tracepoint 기반)
    - tracer 를 function_graph 으로 설정한 결과는 아래와 같다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 1)               |  jbd2_journal_commit_transaction() {
 1)               |  /* jbd2_start_commit: dev 251,0 transaction 17 sync 0 */
 1)               |  /* jbd2_commit_locking: dev 251,0 transaction 17 sync 0 */
 1)               |  /* jbd2_commit_flushing: dev 251,0 transaction 17 sync 0 */
 1)               |  /* jbd2_commit_logging: dev 251,0 transaction 17 sync 0 */
 1)   7.119 us    |    jbd2_journal_write_metadata_buffer();
 1)               |  /* jbd2_run_stats: dev 251,0 tid 17 wait 0 request_delay 0 running 5004 locked 0 flushing 0 logging
 0 handle_count 131 blocks 1 blocks_logged 2 */
 1)               |  /* jbd2_end_commit: dev 251,0 transaction 17 sync 0 head 2 */
 1) ! 189.410 us  |  }
```


Kernel Tracing Tools

- ftrace
  - ✓ ftrace event interface (tracepoint 기반)
    - 커널이 event 에 도달 했을 시, stacktrace 를 볼 수 있도록 trigger 를 사용하여 아래와 같이 설정이 가능하며 stack trace 를 통해 현재 event 가 수행되는 부분에서 함수 호출 순서를 확인 할 수 있다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo '!stacktrace' > events/jbd2/jbd2_start_commit/trigger
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 'stacktrace' > events/jbd2/jbd2_start_commit/trigger
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# fio --output=/home/fs-lecture/lecture/result/test.txt --directory=/mnt/ --n
ame=test --rw=randwrite --size=512K
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace
# tracer: nop
#
# entries-in-buffer/entries-written: 5/5   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
     jbd2/rxd0-8-20524 [000] .... 178165.115189: <stack trace>
 => ftrace_raw_event_jbd2_commit
 => jbd2_journal_commit_transaction
 => kjournald2
 => kthread
 => ret_from_fork
```

Kernel Tracing Tools

- ftrace
  - ✓ ftrace profiling interface (tracepoint 기반)
    - ftrace 의 function_profile 기능을 통해 trace_stat directory 내에서 function prefix 로 명시된 파일들을 통해 각 core 별 커널에서 수행되는 함수들의 수행 시간 통계를 확인할 수 있다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo nop > current_tracer
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 1 > function_profile_enabled
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# ls trace_stat/
function0  function1  function2  function3
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace_stat/function0 | head -10
  Function                            Hit    Time            Avg             s^2
  --------                            ---    ----            ---             ---
  __schedule                        11279    1007278358 us     89305.64 us     1776443 us
  schedule                          10819    1006466186 us     93027.65 us     14404955 us
  poll_schedule_timeout              4262    697229019 us     163591.9 us     18530577 us
  schedule_hrtimeout_range           4184    697212170 us     166637.7 us     46802841 us
  schedule_hrtimeout_range_clock     4184    697195008 us     166633.6 us     664692.9 us
  do_sys_poll                        6216    669252911 us     107666.1 us     934447.7 us
  SyS_poll                           3532    630557919 us     178527.1 us     1533535 us
  arch_cpu_idle                      3733    39465540 us     10572.07 us     6220094 us
```

    - 함수의 누적 수행시간이 아닌 해당 함수에서만의 수행 시간을 확인하려 할 경우, option/graph-time 설정을 disable 하여 확인할 수 있다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 > options/graph-time
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 0 > function_profile_enabled
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# echo 1 > function_profile_enabled
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat trace_stat/function0 | head -10
  Function                            Hit    Time            Avg             s^2
  --------                            ---    ----            ---             ---
  __schedule                          297    195583560 us     658530.5 us     38188204 us
  default_idle                         87    4564091 us     52460.82 us     40689989 us
  __fdget                            5058    24039.94 us     4.752 us        7215.363 us
  __fget_light                       5084    20852.51 us     4.101 us        186314.0 us
  do_select                            53    16960.01 us     320.000 us      229714.5 us
  sock_poll                          3642    14878.79 us     4.085 us        3790.728 us
  get_vtime_delta                    4126    14854.37 us     3.600 us        894.385 us
  acct_account_cputime               3890    13585.25 us     3.492 us        77.220 us
```


Kernel Tracing Tool
실습

Kernel Tracing Tool 실습

- Kernel Tracing 실습 시나리오
  - ✓ 파일시스템 내에 자신이 원하는 위치에 tracepoint 를 생성하여 모니터링 하고 싶은 위치에 대해 debugfs 내에 event를 생성한다.
  - ✓ 현재 커널코드에서 FAT 파일시스템에는 tracepoint 가 없다. 따라서 FAT 모듈 내에 tracepoint 를 삽입하여 event 를 생성해 본다.
  - ✓ 실습의 진행 순서는 다음과 같다.
    1. 예제로 작성된 trace_fat_ent_new_write event를 통해 tracepoint 생성 방법을 숙지하고 ftrace tool 을 통해 분석해 본다.
    2. 예제를 참고하여 두번째 실습인 FAT 파일시스템 logging 실습에서 자신이 직접 구현한 함수에 tracepoint 를 삽입하고 ftrace 를 통해 분석해 본다.
       (또는 실습에서 제공된 FAT 파일시스템 모듈이 아니더라도 분석중인 다른 파일시스템이 있을 경우, 위치에 맞게 trace header file을 생성하여 같은 실습을 진행한다.)

Kernel Tracing 실습

- Kernel Tracing 실습 예제
  - ✓ 실습 예제는 두번째 실습의 참고자료파일과 동일한 위치에서 확인할 수 있다.
    - 실습 파일 위치 : /home/fs-lecture/lecture/fat_practice/fat_log_module_lecture
  - ✓ 실습 예제 파일 구성 및 사용할 파일들은 아래와 같다.

```
root@fslecture-VirtualBox:/home/fs-lecture/lecture/fat_practice/fat_log_module_lecture# ls
Kconfig         dir.o      fat_log.c    file.o         mount.sh       tags                    vfat_log_lecture.mod.c
Makefile        fat.h      fat_log.o    inode.c        namei_msdos.c  test                    vfat_log_lecture.mod.o
Module.symvers  fat.ko     fat_trace.h  inode.o        namei_vfat.c   tracepoint_example.sh   vfat_log_lecture.o
cache.c         fat.mod.c  fatent.c     misc.c         namei_vfat.o   tracepoint_log.sh       xxd.sh
cache.o         fat.mod.o  fatent.o     misc.o         nfs.c          umount.sh
dir.c           fat.o      file.c       modules.order  nfs.o          vfat_log_lecture.ko
```

    - fat_trace.h : FAT 모듈에 tracepoint 를 생성하기 위해 TRACE_EVENT 가 정의된 header file .
    - fatten.c : fat_ent_write 함수 내에 생성한 callback 함수가 삽입되어 있다.
    - Makefile : tracepoint 가 정의된 header file 의 경로가 추가적으로 명시되어 있다.
    - mount.sh : ramdisk 생성, fat 포맷, mount 등을 진행하는 간단한 스크립트 파일
    - tracepoint_example.sh : fat_ent_write 에 삽입한 tracepoint 를 ftrace로 확인하기 위한 간단한 스크립트 파일
    - xxd.sh : xxd 명령어를 통해 fat 영역의 hexdump 를 보기 위한 간단한 스크립트 파일
  - ✓ 기존의 FAT 모듈은 tracepoint 가 없으므로, event 목록에 FAT 모듈을 확인 할 수 없는 것을 볼 수 있다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# ls events/
block            fence        header_page  kmem     nmi      ras          sched    spi      tlb      xen
compaction       filelock     i2c          mce      oom      raw_syscalls scsi     swiotlb  udp      xhci-hcd
context_tracking filemap      iommu        migrate  pagemap  rcu          signal   syscalls vmscan
enable           ftrace       irq          module   power    regmap       skb      task     vsyscall
exceptions       gpio         irq_vectors  napi     printk   regulator    snd_pcm  thermal  workqueue
ext4             header_event jbd2         net      random   rpm          sock     timer    writeback
```

Kernel Tracing 실습

- Kernel Tracing 실습 예제
  - ✓ fatent.c 내의 fat_ent_write 함수가 동작 할 때, 기록되는 FAT cluster number 를 monitoring 하기 위한 trace_fat_ent_new_write 호출이 추가되어 있다.

/home/fs-lecture/fat_practice/fat_log_module_lecture/Makefile

```
#define CREATE_TRACE_POINTS
#include "fat_trace.h"
```

```
int fat_ent_write(struct inode *inode, struct fat_entry *fatent,
          int new, int wait)
{
    struct super_block *sb = inode->i_sb;
    struct fatent_operations *ops = MSDOS_SB(sb)->fatent_ops;
    int err;

    ops->ent_put(fatent, new);
    /* add tracepoint to log fatent number */
    trace_fat_ent_new_write(fatent->fat_inode, new, wait);
    if (wait) {
        err = fat_sync_bhs(fatent->bhs, fatent->nr_bhs);
        if (err)
            return err;
    }
    return fat_mirror_bhs(sb, fatent->bhs, fatent->nr_bhs);
}
```

  - ✓ trace_fat_ent_new_write 함수가 정의된 위치를 참조하기 위해 아래와 같이 tracepoint 를 사용하는 파일을 명시해 주거나 주석 처리된 방법처럼 전체 소스 파일에 적용되도록 할 수 있다.

```
CFLAGS_fatent.o = -I$(src)
#EXTRA_CFLAGS = -I$(src)
```


Kernel Tracing 실습

- Kernel Tracing 실습 예제
  - ✓ TRACE_EVENT 매크로를 통해 생성할, tracepoint 가 받은 args 를 통해 ring buffer 에 기록될 구조체 형태를 생성하며, 초기화될 방법, 출력방법등을 기술한다.

/home/fs-lecture/fat_practice/fat_log_module_lecture/fat_trace.h

```
#undef TRACE_SYSTEM
#define TRACE_SYSTEM fat_lecture

#if !defined(_TRACE_FAT_LECTURE_H) || defined(TRACE_HEADER_MULTI_READ)
#define _TRACE_FAT_LECTURE_H

#include <linux/tracepoint.h>

TRACE_EVENT(fat_ent_new_write,
    TP_PROTO(struct inode *inode, int new_cls, int wait),

    TP_ARGS(inode, new_cls, wait),

    TP_STRUCT__entry(
        __field(    unsigned long   , t_ino         )
        __field(    int             , t_new         )
        __field(    int             , t_wait        )
    ),

    TP_fast_assign(
        __entry->t_ino  = inode->i_ino;
        __entry->t_new  = new_cls;
        __entry->t_wait = wait;
    ),

    TP_printk("ino %lu , cluster : %d , wait : %d ", __entry->t_ino, __entry->t_new ,__entry->t_wait)
);

#endif /* _TRACE_FAT_LECTURE_H  */


#undef TRACE_INCLUDE_PATH
#define TRACE_INCLUDE_PATH .
#define TRACE_INCLUDE_FILE fat_trace
#include <trace/define_trace.h>
```

Kernel Tracing 실습

- Kernel Tracing 실습 예제 실행
  - ✓ 예제 파일 실행은 mount.sh -> tracepoint_example.sh -> xxd.sh -> umount.sh 순서로 진행한다.
    - ./mount.sh
      mount 이후 fat_log_lecture 모듈 관련 event directory 가 생성되는 것을 확인할 수 있다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# ls events/
block            fat_log_lecture  header_event  jbd2     net      random       rpm      sock     timer      writeback
compaction       fence            header_page   kmem     nmi      ras          sched    spi      tlb        xen
context_tracking filelock         i2c           mce      oom      raw_syscalls scsi     swiotlb  udp        xhci-hcd
enable           filemap          iommu         migrate  pagemap  rcu          signal   syscalls vmscan
exceptions       ftrace           irq           module   power    regmap       skb      task     vsyscall
ext4             gpio             irq_vectors   napi     printk   regulator    snd_pcm  thermal  workqueue
```

또한 TRACE_EVENT 에서 정의한 내용에 따라 event format 이 생성된 것을 확인할 수 있다.

```
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# ls events/fat_log_lecture/
enable  fat_ent_new_write  fat_log_check  filter
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# ls events/fat_log_lecture/fat_ent_new_write/
enable  filter  format  id  trigger
root@fslecture-VirtualBox:/sys/kernel/debug/tracing# cat events/fat_log_lecture/fat_ent_new_write/format
name: fat_ent_new_write
ID: 1091
format:
        field:unsigned short common_type;       offset:0;       size:2; signed:0;
        field:unsigned char common_flags;       offset:2;       size:1; signed:0;
        field:unsigned char common_preempt_count;       offset:3;       size:1; signed:0;
        field:int common_pid;   offset:4;       size:4; signed:1;

        field:unsigned long t_ino;      offset:8;       size:8; signed:0;
        field:int t_new;        offset:16;      size:4; signed:1;
        field:int t_wait;       offset:20;      size:4; signed:1;

print fmt: "ino %lu , cluster : %d , wait : %d ", REC->t_ino, REC->t_new ,REC->t_wait
```


Kernel Tracing 실습

- Kernel Tracing 실습 예제 실행
  - ✓ 예제 파일 실행은 mount.sh -> tracepoint_example.sh -> xxd.sh -> umount.sh 순서로 진행한다.
    - ./tracepoint_example.sh
      fat_ent_write 를 filter function 으로 두고, ftrace tracer 를 functiongraph 로 설정한 스크립트를 실행한 결과 아래와 같이 file.c 를 복사하는 과정에서 fat_ent_write 가 호출 될 때 4 번 cluster 와 5 번 cluste를 할당한 것을 확인 할 수 있다.

```
root@fslecture-VirtualBox:/home/fs-lecture/lecture/fat_practice/fat_log_module_lecture# ./tracepoint_example.sh
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 2)               |  fat_ent_write [fat]() {
 2)               |  /* fat_ent_new_write: ino 0 , cluster : 4 , wait : 0  */
 2)   2.944 us    |  }
 2)               |  fat_ent_write [fat]() {
 2)               |  /* fat_ent_new_write: ino 0 , cluster : 5 , wait : 0  */
 2)   2.211 us    |  }
```

Kernel Tracing 실습

- Kernel Tracing 실습 예제 실행
  - ✓ 예제 파일 실행은 mount.sh -> tracepoint_example.sh -> xxd.sh -> umount.sh 순서로 진행한다.
    - ./xxd.sh

      xxd 스크립트를 통해 hexdump 확인 결과 실제로 root directory 인 3번 cluster 를 제외하고, 4번, 5번이 생성된 것을 볼 수 있다.

```
root@fslecture-VirtualBox:/home/fs-lecture/lecture/fat_practice/fat_log_module_lecture# ./xxd.sh
fat1 area - 0x4000
0004000: f8ffff0f ffffff0f f8ffff0f 04000000  ................
0004010: 05000000 ffffff0f 00000000 00000000  ................
0004020: 00000000 00000000 00000000 00000000  ................
0004030: 00000000 00000000 00000000 00000000  ................
0004040: 00000000 00000000 00000000 00000000  ................
0004050: 00000000 00000000 00000000 00000000  ................
0004060: 00000000 00000000 00000000 00000000  ................
0004070: 00000000 00000000 00000000 00000000  ................
fat2 area - 0x203000
0203000: f8ffff0f ffffff0f f8ffff0f 04000000  ................
0203010: 05000000 ffffff0f 00000000 00000000  ................
0203020: 00000000 00000000 00000000 00000000  ................
0203030: 00000000 00000000 00000000 00000000  ................
0203040: 00000000 00000000 00000000 00000000  ................
0203050: 00000000 00000000 00000000 00000000  ................
0203060: 00000000 00000000 00000000 00000000  ................
0203070: 00000000 00000000 00000000 00000000  ................
log area - 0x402000
0402000: 46494c45 20202020 43202000 114b4d10  FILE    C  ..KM.
0402010: 00000000 00000000 00000000 00000000  ................
0402020: 00000000 00000000 00000000 00000000  ................
0402030: 00000000 00000000 00000000 00000000  ................
0402040: 00000000 00000000 00000000 00000000  ................
0402050: 00000000 00000000 00000000 00000000  ................
0402060: 00000000 00000000 00000000 00000000  ................
0402070: 00000000 00000000 00000000 00000000  ................
data area - 0x601000
0601000: 41660069 006c0065 002e000f 00856300  Af.i.l.e......c.
0601010: 0000ffff ffffffff ffff0000 ffffffff  ................
0601020: 46494c45 20202020 43202020 00004d10  FILE    C   ..M.
0601030: 114b114b 00004d10 114b0000 00000000  .K.K..M..K......
0601040: 00000000 00000000 00000000 00000000  ................
0601050: 00000000 00000000 00000000 00000000  ................
0601060: 00000000 00000000 00000000 00000000  ................
0601070: 00000000 00000000 00000000 00000000  ................
```

    - ./umount

      umount 를 통해 모듈 해제, ramdisk 해제 등을 수행한다.

Kernel Tracing 실습

- Kernel Tracing 실습 구현
  - ✓ 실습 예제 파일을 참고하여 두번째 FAT logging 실습에서 구현한 log 영역에 파일 이름, 접근 시간 등을 기록하는 함수에 tracepoint 를 삽입하거나 자신이 분석중인 파일시스템의 필요한 부분에 tracepoint를 직접 삽입하여 ftrace tool 을 통해 monitoring 해본다.
  - ✓ 실습 위치
    - /home/fs-lecture/lecture/fat_practice/fat_log_module_practice/
  - ✓ 실습 결과.
    - 두번째 실습인 FAT logging 실습에서 구현한 함수에 대하여 file.c 파일과 inode.c 파일을 mount directory 에 복사하는 과정에서의 monitoring 결과 예시는 다음과 같다.

```
root@fslecture-VirtualBox:/home/fs-lecture/lecture/fat_practice/fat_log_module_lecture# ./tracepoint_log.sh
# tracer: nop
#
# entries-in-buffer/entries-written: 2/2   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
             cp-15661 [000] .... 17592.326377: fat_log_check: logged content[file name : FILE    C  , ino : 24,
date : 2017.8.17, time : 3:15:30, blocknr : 12296, sectornr : 8208
             cp-15661 [000] .... 17592.326460: fat_log_check: logged content[file name : INODE   C  , ino : 25,
date : 2017.8.17, time : 3:15:30, blocknr : 12296, sectornr : 8208
```

    - 실습 시 진행한 간단한 FAT 파일시스템 외에 다양한 복잡한 파일시스템에서 tracepoint, kprobe 등에 기반한 ftrace, perf 등의 다양한 tool 들은 강력한 프로파일링 및 디버깅 도구로 활용 될 수 있다.

3. Performance Benchmarking Tools

Performance Benchmarking 실습

- Performance evaluation 실습의 목표
  - ✓ Benchmark를 이해한다.
  - ✓ Performance evaluation을 위해 다양한 benchmark tool들의 사용법을 익힌다.
  - ✓ 자신이 구현한 system의 performance를 직접 측정함으로서 자신의 system의 정당성을 입증할 수 있는 능력을 기른다.
  - ✓ Kernel의 요소들의 performance를 직접 측정해 봄으로써 performance에 관한 issue들을 발견할 수 있는 능력과 분석력을 기른다.
- Performance evaluation 실습의 내용
  - ✓ 다양한 benchmark tool들의 소개한다.
  - ✓ 다양한 benchmark tool들의 사용법을 익힌다.
  - ✓ F2FS 실습시간에 수정한 source code의 performance를 측정해 본다.

Performance Benchmarking Tools

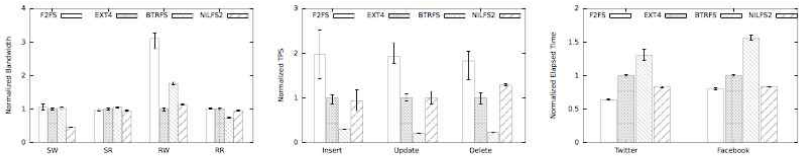
- Benchmark
  - ✓ 벤치마킹이란 측정의 기준이 되는 대상을 설정하고 그 대상과 비교 분석을 통해 장점을 따라 배우는 행위를 말한다.
  - ✓ 시스템에서의 벤치마킹이란 측정의 기준이 되는 대상을 설정하고 그 대상과 비교 분석을 하는행위를 말한다.
  - ✓ 특정한 작업의 환경에 따라 어떠한 file system을 선택하느냐가 system performance에 많은 영향을 미칠 수 있다.
    - 이를 위해 benchmark를 통한 file system의 성능 측정은 매우 중요하다.
  - ✓ 본 실습에서는 다양한 benchmark를 소개하고 이를 이용하여 다양한 file system의 성능을 측정하는 방법을 알아본다.

Performance Benchmarking Tools

- 다양한 Benchmark의 종류
  - ✓ IOzone
  - ✓ Fio
  - ✓ Filebench
  - ✓ Postmark
  - ✓ Dbench
  - ✓ Bonnie++
  - ✓ Phronix
  - ✓ Iometer
  - ✓ Fs-mark
  - ✓ Lmbench
  - ✓ …. etc

Performance Benchmarking Tools

- benchmark의 사용 사례



(a) iozone (b) SQLite (c) Facebook-app and Twitter-app

Figure 3: Performance results on the mobile system.

<Source: fast'15 F2FS: A new File System for Flash Storage>

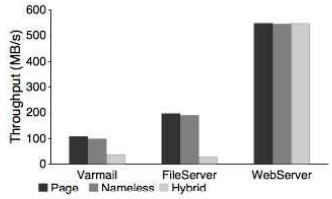


Figure 2: **Throughput of Filebench.** *Throughput of varmail, fileserver, and webmail macro-benchmarks with page-level, nameless-writing, and hybrid FTLs.*

<Source: fast'12 Implication of New Storage Technology>

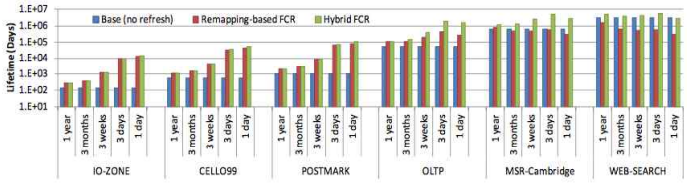
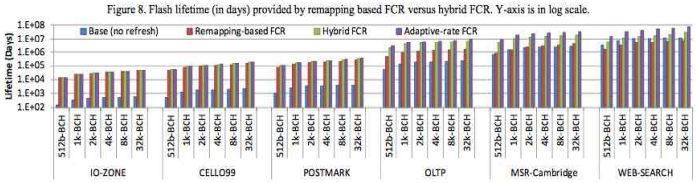
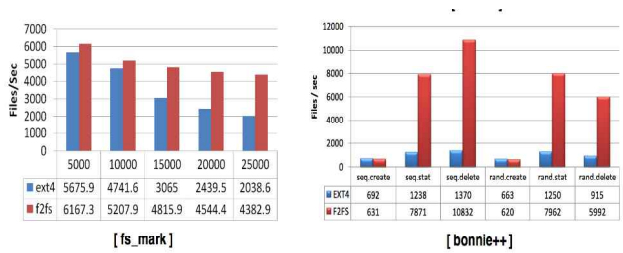


Figure 8. Flash lifetime (in days) provided by remapping based FCR versus hybrid FCR. Y-axis is in log scale.



<Source: flash correct-and-refresh: retention-aware error Management for increased flash memory life time>



<Source: J. Hwang, "Flash-Friendly File system", Feb. 2013>

Performance Benchmarking Tools

- benchmark의 사용 사례

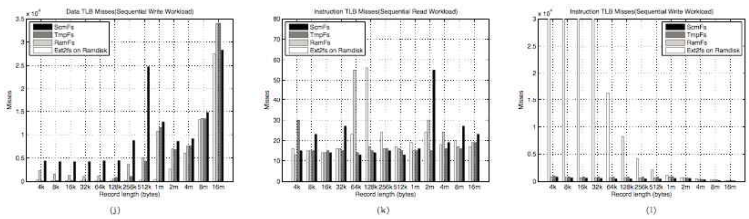


Figure 6: IOZONE results(Sequential workloads)

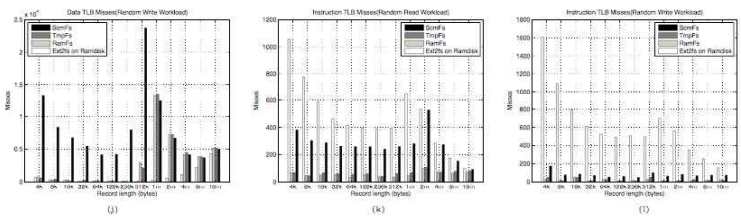


Figure 7: IOZONE results(Random workloads)

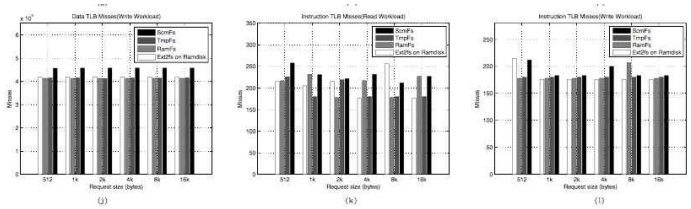
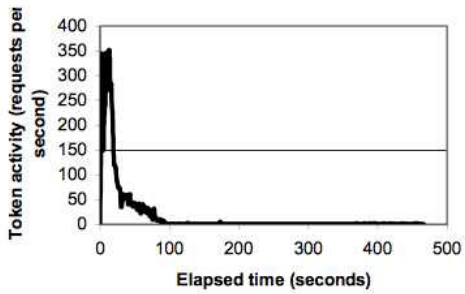


Figure 11: Postmark results. SCMFS-1, original SCMFS. SCMFS-2, SCMFS-1 with space pre-allocation.

<Source: SCMFS: a file system for storage class memory>

Figure 5: Token Activity During Dbench Run



<Source: GPFS: a shared-disk file system for large computing clusters>

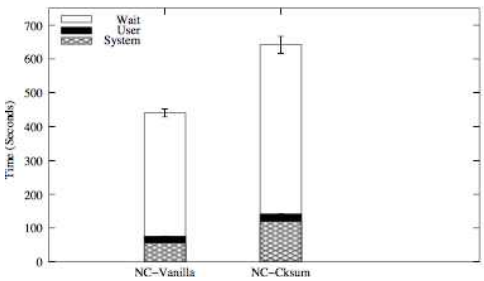


Figure 1: Postmark results

<Source: enhancing file system integrity through checksums>

Performance Benchmarking Tools

- benchmark의 사용 사례

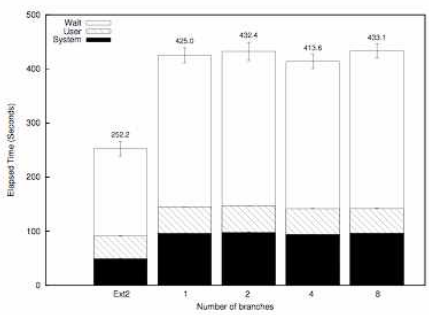


Figure 7: Postmark: 20,000 files and 200,000 transactions.

<Source: Unionfs: User-and community-oriented development of a unification filesystem>

System	Untar	Delete	PostMark	Andrew
Featherstitch ext2				
soft updates	**6.4 [1.3]**	**0.8 [0.1]**	**38.3 [10.3]**	**36.9 [4.1]**
meta journal	**5.8 [1.3]**	**1.4 [0.5]**	**48.3 [14.5]**	**36.7 [4.2]**
full journal	**11.5 [3.0]**	**1.4 [0.5]**	**82.8 [19.3]**	**36.8 [4.2]**
async	4.1 [1.2]	0.7 [0.2]	37.3 [6.1]	36.4 [4.0]
full journal	10.4 [3.7]	1.1 [0.5]	74.8 [23.1]	36.5 [4.2]
Linux				
ext3 writeback	**16.6 [1.0]**	**4.5 [0.3]**	**38.2 [3.7]**	**36.8 [4.1]**
ext3 full journal	**12.8 [1.1]**	**4.6 [0.3]**	**69.6 [4.5]**	**38.2 [4.0]**
ext2	4.4 [0.7]	4.6 [0.1]	35.7 [1.9]	36.9 [4.0]
ext3 full journal	10.6 [1.1]	4.4 [0.2]	61.5 [4.5]	37.2 [4.1]

<Source: Generalized file system dependencies>

Type	Suite/Benchmark	Resource	Unit
SI	LMbench/all [24]	Many	Many
SI	Bonnie/all [25], [26]	Disk	MBps
SI	CacheBench/all [27]	Memory	MBps
MI	HPCC/HPL [28], [29]	CPU	GFLOPS
MI	HPCC/DGEMM [30]	CPU	GFLOPS
MI	HPCC/STREAM [30]	Memory	GBps
MI	HPCC/RandomAccess [31]	Network	MUPS
MI	HPCC/b_{eff}(lat.,bw.) [32]	Comm.	µs, GBps

<Source: Performance Analysis of cloud computing services for many-tasks scientific computing>

Type	Suite / Benchmark	Resource	Unit
Single instance	LMbench / all [12]	Many	Many
Single instance	Bonnie / all [11, 85]	Disk	MBps
Single instance	CacheBench / all [159]	Memory	MBps
Multi instance	HPCC / HPL [93, 139]	CPU	GFLOPS
Multi instance	HPCC / DGEMM [43]	CPU	GFLOPS
Multi instance	HPCC / STREAM [43]	Memory	GBps
Multi instance	HPCC / RandomAccess [4]	Network	MUPS
Multi instance	HPCC / b_{eff}(lat.,bw.) [16]	Comm.	μs, GBps

<Source: resource management for scientific application in hybrid cloud computing enviroment>

Performance Benchmarking Tools

- iozone
  - ✓ "# apt-get install iozone3"를 통해 iozone을 install한다.
  - ✓ iozone을 통해 측정할 수 있는 것들은 다음과 같다.
    - Read
    - Write
    - Re-read
    - Re-write
    - Random read
    - Random write
    - Backward read
    - Record re-write
  - ✓ iozone option
    - -a : 모드를 자동으로해서 실행한다. 기록하는 사이즈는 4K부터 16M까지 증가하며, 파일의 크기는 64K에서 512M까지 반복하며 테스트한다.
    - -n : 자동모드의 최소파일크기를 설정한다.
    - -r : 사용자가 지정해서 테스트 기록사이즈를 결정한다. 단위로는 k,m,g 등을 사용한다.
    - -i : 테스트할 항목들을 별도로 지정해서 테스트할 때 사용한다.
      - (0=write/rewrite, 1=read/re-read, 2=random-read/write, 3=Read-backwards, 4=Re-write-record, 5=stride-read, 6=fwrite/re-fwrite, 7=fread/Re-fread, 8=random mix, 9=pwrite/Re-pwrite, 10=pread/Re-pread, 11=pwritev/Re-pwritev, 12=preadv/Re-preadv).
    - 부가적인 옵션은 "# iozone –h"를 통해 확인가능하다.

Performance Benchmarking Tools

- -a option
  - ✓ -a 옵션의 경우에는 iozone을 자동으로 수행한다.
  - ✓ -a 옵션을 주고 수행하는 경우에 test file의 크기는 64K~512M, record size의 크기는 4K~16M의 크기로 random으로 수행된다.

 # iozone –a (mount point)

```
Iozone: Performance Test of File I/O
        Version $Revision: 3.397 $
        Compiled for 64 bit mode.
        Build: linux-AMD64

Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
             Al Slater, Scott Rhine, Mike Wisner, Ken Goss
             Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
             Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
             Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
             Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root,
             Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer.
             Ben England.

Run began: Thu Aug  6 20:18:44 2015

Auto Mode
Command line used: iozone -a /mnt/fat_test/
Output is in Kbytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 Kbytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
                                                              random  random    bkwd   record   stride
      KB  reclen   write rewrite    read    reread    read    write    read  rewrite     read   fwrite frewrite   fread  freread
      64       4  286849  603489 1638743  1734015 1599680   673098  852702   712397  1204796   561808   598110  969761  1392258
      64       8  329852  659861 1879725  1991276 1553394   760859  955947   866463  1279447   695778   686877 1188789  1638743
      64      16  735831  673098 1879725  2067979 1638743   397796  983980   852702  1119387   673098   727850 1124074  1330167
      64      32  271200  718114 2298136  2358717 1638743   528618  998622   901377  1049372   659861   955947 1049372  1210227
      64      64  353760  712397 2133730  2222043 1648808   852702  998622   855419   733819   712397   542506  743988   821391
     128       4  776042 1082825 1684006  1684006 1467639   499970 1123617   826202  1185654   630273   673778 1319723  1526043
     128       8  360605  794416 1967960  1997245 1755594   546819 1254941  1059322  1561553   776042   774922 1487977  1684006
     128      16  396271  831319 2066432  2166499 1967960   940549 1206978  1093856  1603529   795593   848397 1539168  1755594
     128      32  870405  847059 2090572  2202044 1967960   976473 1319723  1074159  1642783   895074   711270 1504659  1618028
     128      64  895074 2027414 2035099  2066432 2211113   985435 1291158  1030846  1508887   680612  1074159 1455701  1172704
     128     128  400110  830034 2276751  2326073 2248149   990891  743788   612303  1436229   889145   895074 1471662  1858934
```

File size

record size

Performance Benchmarking Tools

- -b option
  - ✓ -b 옵션은 iozone test의 수행결과를 spreadsheet 형태의 파일로 저장해주는 옵션이다.
  - ✓ 수행 결과물인 xls 파일은 엑셀형태로서 생성된다.

 # iozone –a -b (xls 파일 결과물 생성 위치) (mount point)

<테스트 결과 파일>

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	iozone -a -b /home/donghwa/result.xls /mnt/fat_test/													
2														
3	The top row is records sizes, the left column is file sizes													
4	Writer Report													
5		4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
6	64	283217	353760	348253	351444	357529								
7	128	345075	375485	397444	870405	0								
8	Re-writer Report													
9		4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
10	64	615952	666414	703068	710511	695778								
11	128	700139	766076	1939522	821147	0								
12	Reader Report													
13		4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
14	64	1734015	1933893	2133730	2203800	2052169								
15	128	1732927	1939522	5122535	1975201	0								
16	Re-reader Report													
17		4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
18	64	1828508	1933893	2133730	2203800	2133730								
19	128	1778862	2066432	2132084	2066432	0								
20	Random Read Report													
21		4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
22	64	1599680	1526887	1599680	1599680	2067979								
23	128	1471662	1802755	1967960	1911894	0								
24	Random Write Report													
25		4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
26	64	659861	496362	520419	533875	852702								
27	128	752124	864797	942199	581781	0								
28	Backward Read Report													
29		4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
30	64	942521	969761	1002351	1013707	674790								
31	128	1105114	1220700	1332828	1294270	0								

Performance Benchmarking Tools

- -i option
  - ✓ -i 옵션은 특정한 방식으로 test를 수행하고 싶을 때 사용한다.

    # iozone –i (test-type) (mount point)

  - ✓ -i 옵션의 test-type
    - 0 : write / rewrite
    - 1 : random-read / write
    - 2 : read-backwards
    - 4 : re-write-record
    - 5 : stride-read
    - 6 : fwrite/re-fwrite
    - 7 : fread/re-fread
    - 8 : random mix
    - 9 : pwrite/re-pwrite
    - 10 : pread/re-pread
    - 11 : pwritev/re-pwritev
    - 12 : preadv/re-preadv

  - ✓ -i 옵션의 test-type은 복합적으로 사용 가능하다.

    # iozone –i 0 –i 1 (mount point)

Performance Benchmarking Tools

- -s, -r option
  - ✓ -s 옵션의 경우 iozone에서 test로 사용하는 파일의 크기를 지정하는 옵션이다.
  - ✓ 지정을 해주지 않는 경우 default로 64K~512M의 다양한 크기로 test를 한다.

 # iozone –a –i 0 –s 1024 (mount point)

```
  KB  reclen   write rewrite
1024       4 1257748  781072
1024       8  418661  915223
1024      16  444254 2375790
1024      32  451303  983548
1024      64  447727  939446
1024     128  454503  957884
1024     256  446564  849176
1024     512  435957  854752
1024    1024  424202  888894
```

  - ✓ -r 옵션의 경우 record size를 지정하는 옵션이다.

 # iozone –a –i 0 –r 32 (mount point)

```
   KB  reclen   write rewrite
   64      32  316623  703068
  128      32  399812  646205
  256      32  436870  907792
  512      32  449094  937435
 1024      32  435957  864734
 2048      32  456333  940764
 4096      32  451997  845390
 8192      32  706571 1279415
```

Performance Benchmarking Tools

- -t option
  - ✓ -t 옵션의 경우 multi thread를 사용하여 test를 수행하게 하는 옵션이다.
  - ✓ -a와 –t 옵션을 함께 사용할 수 없다는 점을 주의해야한다.

 # iozone –i 0 –t 2 (mount point)

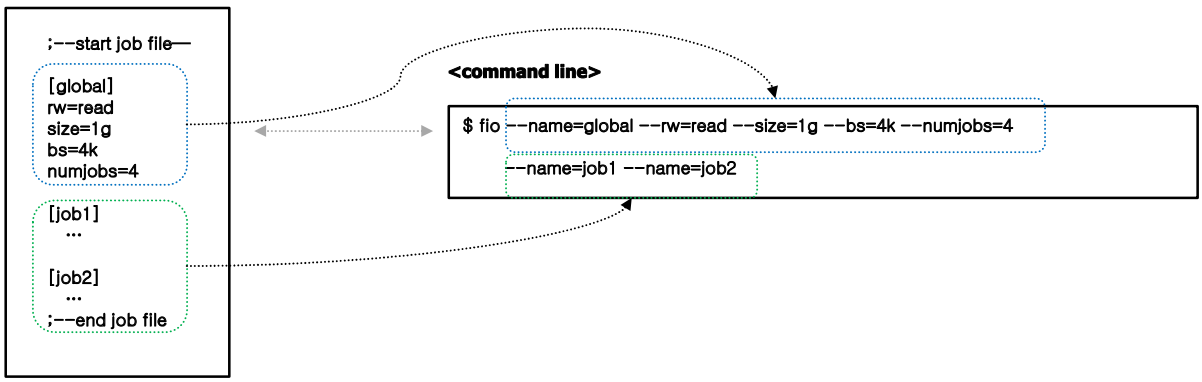
```
Children see throughput for  2 initial writers  =  647309.09 KB/sec
Parent sees throughput for  2 initial writers   =   17801.60 KB/sec
Min throughput per process                      =  315743.84 KB/sec
Max throughput per process                      =  331565.25 KB/sec
Avg throughput per process                      =  323654.55 KB/sec
Min xfer                                        =     496.00 KB

Children see throughput for  2 rewriters        = 1209506.94 KB/sec
Parent sees throughput for  2 rewriters         =   17382.40 KB/sec
Min throughput per process                      =  599219.31 KB/sec
Max throughput per process                      =  610287.62 KB/sec
Avg throughput per process                      =  604753.47 KB/sec
Min xfer                                        =     504.00 KB
```

- iozone의 더욱 상세한 옵션은 다음의 웹사이트에서 확인 가능하다.
  - ✓ http://www.iozone.org/docs/IOzone_msword_98.pdf

Performance Benchmarking Tools

- fio
  - ✓ "# apt-get install fio"를 통해 fio를 install한다.
  - ✓ fio는 직접 만든 job file 혹은 command line을 통해 원하는 benchmarking을 진행할 수 있다.

<job file>



- fio option
  - ✓ Fio의 option은 크게 I/O type, Block size, I/O size, I/O engine, I/O depth, num files, num threads로 나뉘며, 이러한 option들을 이용하여 job file을 만들거나 command line에서 직접 명령어를 입력하여 benchmarking을 수행한다.

Performance Benchmarking Tools

- I/O type option
  - ✓ I/O type 옵션은 fio에서 수행하는 I/O의 pattern을 결정하는 옵션이며, 가능한 parameter들은 다음과 같다.
    - read : sequential read
    - write : sequential write
    - randread : random read
    - randwrite : random write
    - readwrite/rw : sequential read write mix
    - randrw : random read write mix
    - read/write mix의 경우 --rwmixread를 통해 비율을 조정할 수 있다.
      - ex) –rwmixread=70 , 70%read, 30%write
- Block size option
  - ✓ Block size 옵션은 fio에서 수행하는 I/O의 block 크기를 조절하는 옵션이다. size (m,k,g) 입력
- I/O size option
  - ✓ I/O size 옵션은 fio에서 read/write할 data의 크기를 조절하는 옵션이다. . size (m,k,g) 입력

Performance Benchmarking Tools

- I/O engine option
  - ✓ I/O engine 옵션은 fio에서 I/O를 어떤 식으로 수행하는지를 나타내는 옵션이다.
  - ✓ 사용 가능한 I/O engine은 다음과 같다.
    - sync
    - psync
    - vsync
    - psyncv
    - libaio
    - posixaio
    - windowsaio
    - mmap
    - …etc
- Num files, num jobs option
  - ✓ Num files/jobs 옵션은 각각 지정된 크기의 파일을 몇 개나 read/write할지, 몇 개의 job을 돌려서 bench marking을 수행할지를 결정해주는 옵션이다.
- fio의 상세한 옵션은 다음의 웹사이트에서 확인가능하다.
  - ✓ http://www.bluestop.org/fio/HOWTO.txt

Performance Benchmarking Tools

- fio example – command line

fio --name=fio_ex --filename=(mount point)/test --rw=read --blocksize=4k --size=2g --numjobs=2

```
fio_ex: (g=0): rw=read, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1
fio_ex: (g=0): rw=read, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1
fio 1.59
Starting 2 processes
Jobs: 2 (f=2)
fio_ex: (groupid=0, jobs=1): err= 0: pid=6499
  read : io=2048.0MB, bw=1299.6MB/s, iops=332670 , runt=  1576msec
    clat (usec): min=0 , max=305 , avg= 2.16, stdev= 9.98
     lat (usec): min=0 , max=305 , avg= 2.23, stdev= 9.98
    bw (KB/s) : min=832944, max=1565208, per=49.56%, avg=1318960.00, stdev=420914.67
  cpu          : usr=18.16%, sys=81.33%, ctx=460, majf=0, minf=4
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued r/w/d: total=524288/0/0, short=0/0/0
     lat (usec): 2=68.62%, 4=29.99%, 10=0.14%, 20=0.30%, 50=0.21%
     lat (usec): 100=0.22%, 250=0.52%, 500=0.01%

fio_ex: (groupid=0, jobs=1): err= 0: pid=6500
  read : io=2048.0MB, bw=1299.6MB/s, iops=332670 , runt=  1576msec
    clat (usec): min=0 , max=273 , avg= 2.16, stdev=10.06
     lat (usec): min=0 , max=273 , avg= 2.23, stdev=10.06
    bw (KB/s) : min=832928, max=1565880, per=49.57%, avg=1319130.67, stdev=421079.66
  cpu          : usr=20.19%, sys=79.37%, ctx=413, majf=0, minf=5
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued r/w/d: total=524288/0/0, short=0/0/0
     lat (usec): 2=69.15%, 4=29.45%, 10=0.14%, 20=0.30%, 50=0.21%
     lat (usec): 100=0.22%, 250=0.53%, 500=0.01%


Run status group 0 (all jobs):
   READ: io=4096.0MB, aggrb=2598.2MB/s, minb=1330.7MB/s, maxb=1330.7MB/s, mint=1576msec, maxt=1576msec
```

Parallel하게 **benchmarking** 진행

Performance Benchmarking Tools

- fio example – job file
  - ✓ Job file은 “xxxx.fio”의 형태로 만들고 작성한다.
  - ✓ Job file은 “# fio xxxx.fio”명령어로 실행할 수 있다.

```
[globla]
filename=/mnt/fat_test/test
rw=read
blocksize=4k
size=2g
numjobs=2

[fio_job_1]

[fio_job_2]

[fio_job_3]
```

- 모든 **parameter**들은 공유된다.
- 각각의 **job**에 특정 **parameter**들을 설정하는것도 가능하다.

```
fio_ex: (g=0): rw=read, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1
fio_ex: (g=0): rw=read, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1
fio 1.59
Starting 2 processes
Jobs: 2 (f=2)
fio_ex: (groupid=0, jobs=1): err= 0: pid=6499
  read : io=2048.0MB, bw=1299.6MB/s, iops=332670 , runt=  1576msec
    clat (usec): min=0 , max=305 , avg= 2.16, stdev= 9.98
     lat (usec): min=0 , max=305 , avg= 2.23, stdev= 9.98
    bw (KB/s) : min=832944, max=1565208, per=49.56%, avg=1318960.00, stdev=420914.67
  cpu          : usr=18.16%, sys=81.33%, ctx=460, majf=0, minf=4
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued r/w/d: total=524288/0/0, short=0/0/0
     lat (usec): 2=68.62%, 4=29.99%, 10=0.14%, 20=0.30%, 50=0.21%
     lat (usec): 100=0.22%, 250=0.52%, 500=0.01%

fio_ex: (groupid=0, jobs=1): err= 0: pid=6500
  read : io=2048.0MB, bw=1299.6MB/s, iops=332670 , runt=  1576msec
    clat (usec): min=0 , max=273 , avg= 2.16, stdev=10.06
     lat (usec): min=0 , max=273 , avg= 2.23, stdev=10.06
    bw (KB/s) : min=832928, max=1565880, per=49.57%, avg=1319130.67, stdev=421079.66
  cpu          : usr=20.19%, sys=79.37%, ctx=413, majf=0, minf=5
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued r/w/d: total=524288/0/0, short=0/0/0
     lat (usec): 2=69.15%, 4=29.45%, 10=0.14%, 20=0.30%, 50=0.21%
     lat (usec): 100=0.22%, 250=0.53%, 500=0.01%


Run status group 0 (all jobs):
   READ: io=4096.0MB, aggrb=2598.2MB/s, minb=1330.7MB/s, maxb=1330.7MB/s, mint=1576msec, maxt=1576msec
```

fio --filename=(mount point)/test --rw=read --blocksize=4k --size=2g --numjobs=2 **--file=fio_job_1 --file=fio_job_2 ....**

Command line에서 **job**을 여러개 주고 **benchmarking**하는 방법

Performance evaluation
실습

Performance Benchmarking 실습

- Performance evaluation 실습 시나리오
  - ✓ 이전의 실습에서 수정되지 않은 원본 f2fs, garbage collection을 수정한 f2fs 두 개의 모듈을 각각 적재하고 mount 한 후, 원본과 수정본의 performance를 benchmark tool을 이용하여 비교한다.
  - ✓ 현재 실습환경에서는 ramdisk를 사용하기 때문에 정확한 실험결과는 얻을 수 없다는 점을 유의하여야 한다.

```
drwxr-xr-x  7 root     root       4096  8월  11 16:17 .
drwxr-xr-x 45 donghwa donghwa   12288  8월  11 14:22 ..
-rw-r--r--  1 root     root      52736  8월   8 23:48 f2fs-tools_1.4.1.orig.tar.xz
drwxr-xr-x  3 root     root       4096  8월  11 16:17 f2fs_module
drwxr-xr-x  3 root     root       4096  8월  11 13:37 f2fs_module_1
drwxr-xr-x  2 root     root       4096  8월   8 23:49 fat_module
drwxr-xr-x 12 root     root       4096  8월  11 14:08 mtd_module
-rw-r--r--  1 root     root    2097920  8월   8 23:48 rxdsk-3.0.tar
drwxr-xr-x  6 root     root       4096  8월   9 16:26 test
```

  - ✓ 실습의 진행 순서는 다음과 같다.
    1. 수정한 f2fs를 iozone, bonnie, ... 등의 benchmark tool을 이용하여 성능을 측정하고 모듈을 내린다.
    2. 수정하지 않은 f2fs_origin 모듈을 컴파일, 적재, mount 후 iozone, bonnie, ... 등의 benchmark tool을 이용하여 성능을 측정하고 모듈을 내린다.
    3. 측정 결과를 비교한다.

Performance Benchmarking 실습

- Performance evaluation 실습 시나리오
  - ✓ 본 실습의 예제에서는 iozone을 이용한다.

```
File size set to 524288 KB
Record Size 512 KB
Command line used: iozone -i 1 -i 0 -s 512m -r 512 -t 4
Output is in Kbytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 Kbytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Throughput test with 4 processes
Each process writes a 524288 Kbyte file in 512 Kbyte records

Children see throughput for  4 initial writers  =   75084.74 KB/sec
Parent sees throughput for  4 initial writers   =   60251.62 KB/sec
Min throughput per process                      =   16646.29 KB/sec
Max throughput per process                      =   19995.79 KB/sec
Avg throughput per process                      =   18771.18 KB/sec
Min xfer                                        =  436736.00 KB

Children see throughput for  4 rewriters        =   75550.53 KB/sec
Parent sees throughput for  4 rewriters         =   64278.85 KB/sec
Min throughput per process                      =   17635.54 KB/sec
Max throughput per process                      =   19354.78 KB/sec
Avg throughput per process                      =   18887.63 KB/sec
Min xfer                                        =  478208.00 KB

Children see throughput for  4 readers          = 11095355.25 KB/sec
Parent sees throughput for  4 readers           = 9745425.74 KB/sec
Min throughput per process                      = 2739737.50 KB/sec
Max throughput per process                      = 2851109.75 KB/sec
Avg throughput per process                      = 2773838.81 KB/sec
Min xfer                                        =  504320.00 KB

Children see throughput for 4 re-readers        = 11083118.25 KB/sec
Parent sees throughput for 4 re-readers         = 11005437.96 KB/sec
Min throughput per process                      = 2734238.50 KB/sec
Max throughput per process                      = 2835781.50 KB/sec
Avg throughput per process                      = 2770779.56 KB/sec
Min xfer                                        =  505856.00 KB
```

<수정하지 않은 F2FS>

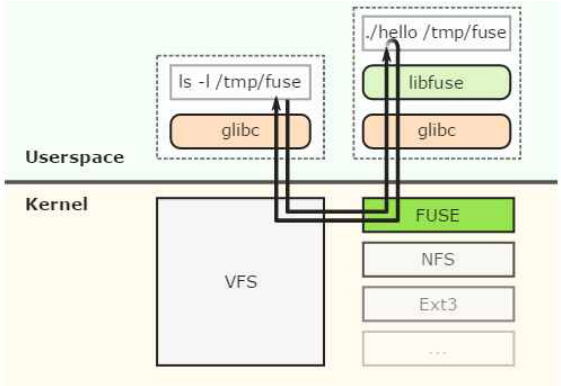
```
File size set to 524288 KB
Record Size 512 KB
Command line used: iozone -i 1 -i 0 -s 512m -r 512 -t 4
Output is in Kbytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 Kbytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
Throughput test with 4 processes
Each process writes a 524288 Kbyte file in 512 Kbyte records

Children see throughput for  4 initial writers  =   75149.21 KB/sec
Parent sees throughput for  4 initial writers   =   63583.29 KB/sec
Min throughput per process                      =   18225.45 KB/sec
Max throughput per process                      =   19045.26 KB/sec
Avg throughput per process                      =   18787.30 KB/sec
Min xfer                                        =  502272.00 KB

Children see throughput for  4 rewriters        =   75374.93 KB/sec
Parent sees throughput for  4 rewriters         =   64017.41 KB/sec
Min throughput per process                      =   17403.95 KB/sec
Max throughput per process                      =   19430.78 KB/sec
Avg throughput per process                      =   18843.73 KB/sec
Min xfer                                        =  470016.00 KB

Children see throughput for  4 readers          = 11300954.25 KB/sec
Parent sees throughput for  4 readers           = 10169463.82 KB/sec
Min throughput per process                      = 2812975.50 KB/sec
Max throughput per process                      = 2832778.75 KB/sec
Avg throughput per process                      = 2825238.56 KB/sec
Min xfer                                        =  520704.00 KB

Children see throughput for 4 re-readers        = 11340136.50 KB/sec
Parent sees throughput for 4 re-readers         = 11258636.20 KB/sec
Min throughput per process                      = 2825140.00 KB/sec
Max throughput per process                      = 2844846.75 KB/sec
Avg throughput per process                      = 2835034.12 KB/sec
Min xfer                                        =  520704.00 KB
```

<수정 한 F2FS>

FUSE File System
추가 실습

FUSE File System 추가 실습

- FUSE 파일 시스템
  - ✓ 개발자들이 Kernel Level 에서가 아닌 User Level 에서 다양한 언어로 파일시스템을 개발 할 수 있도록 API를 제공
  - ✓ User Level daemon, Kernel Module 로 구성.
    - FUSE User Level daemon : 연산의 핵심 동작 구현부로 커널 모듈의 큐에 요청이 있음이 확인 될 시, 커널 모듈의 요청을 읽어와 처리 한 후, 커널 레벨로 요청 결과를 전송.
      장치에 직접 접근 하거나 stackable file system 으로 구현 가능.
    - FUSE Kernel Module : 사용자 프로그램의 요청을 관리하는 다양한 큐를 가지고 있으며, VFS를 거친 사용자의 요청을 User Level Daemon 에 전달 및 User Level Daemon 의 결과를 사용자 프로그램에 전달.



FUSE File System 추가 실습

- FUSE 파일 시스템
  - ✓ FUSE 파일 장단점
    - 사용하기 쉬운 interface 를 제공
    - Kernel Level 에서의 구현 보다 낮은 장벽
  - ✓ FUSE 파일 시스템의 단점
    - libfuse 설치 필요.
    - low-level implementation 보다 낮은 성능.
      최근 다양한 옵션 제공 및 최적화로 성능 향상 기대.
      - ref. Vangoor, Bharath Kumar Reddy, Vasily Tarasov, and Erez Zadok. "To FUSE or Not to FUSE: Performance of User-Space File Systems." *FAST*. 2017.
  - ✓ FUSE 파일 시스템 예시
    - LoggedFS, GlusterFS, SSHFS, SMailFS

- UNIX 파일 시스템
  - ✓ UNIX operating system 출시 당시 UNIX 개발자인 Ken Tompson 이 개발한 파일 시스템으로 inode, bitmap, multi-level indexing 구조를 가진 파일 시스템이다.

FUSE File System 추가 실습

- FUSE 기반 Simple UFS 구현
  - ✓ 과제 구현 목표
    - FUSE 를 활용하여 ramisk 를 기반으로 동작하는 간단한 UNIX file system 을 구현한다.
    - 다중 user 를 지원하기 위해 pthread 에서 제공되는 condition variable, spinlock, rwlock 등을 사용하여 구현한다.
  - ✓ 실습 과제 구성

    본 과제에서는 user level file system interface 인 FUSE를 기반으로 간단한 UNIX 파일시스템을 구현한다. 실습 관련 직접 수정 및 구현 해야 하는 파일은 아래와 같다.

```
root@fslecture-VirtualBox:/home/fs-lecture/lecture/fuse# ls
Makefile  bmap.c  dir.c  example  file.c  fuse_main.c  include  inode.c  mkfs  mount.c
```

    - inode.c : inode, bitmap, data bitmap 함수 관련 소스 파일 구현
    - file.c : write, read, create, delete 등 file 연산 관련 파일
    - dir.c : mkdir, rmdir, rename 등 directory 연산 관련 파일
    - inode.c : inode table, inode 관련 연산 파일
  - ✓ 실습 관련 매뉴얼
    - http://embedded.dankook.ac.kr/~choijm/course/201701OSI/Lab3_FileSystem.pdf

Appendix

Performance Benchmarking Tools

- Dbench
  - ✓ "# apt-get install dbench"를 통해 dbench를 install한다.
  - ✓ Dbench는 I/O workload를 생성하여 benchmarking을 진행하며 file system, NFS server, SCSI를 target으로 테스트할 수 있다.
  - ✓ 이 전의 iozone, fio와 달리 간단한 명령어와 option만으로 사용가능하다는 특징이 있다.

- Dbench option
  - ✓ -B : "fileio", "sockio", "nfs", "scsi", "iscsi", "smb"등의 target I/O를 설정하는 옵션. (default : fileio)
  - ✓ -c : loadfile(fio의 job file 같은 개념)을 통해 dbench benchmarking을 진행하는 옵션.
  - ✓ -t : dbench benchmark의 수행 시간을 정하는 옵션. (default : 600)
  - ✓ -D : benchmarking의 target이 되는 mount point directory를 설정하는 옵션.
  - ✓ -R : load file을 사용하는 경우 rate를 조절하여 load file의 수행속도를 빠르게 하는 옵션.

- 이 외에 B 옵션에 따라 추가할 수 있는 다양한 옵션들이 존재한다.
  - ✓ https://dbench.samba.org/doc/dbench.1.html

Performance Benchmarking Tools

- Dbench example

 # debench –D (mounting point) –t (run time) ....options... (num of process)

```
dbench version 4.00 - Copyright Andrew Tridgell 1999-2004

Running for 10 seconds with load '/usr/share/dbench/client.txt' and minimum warmup 2 secs
0 of 1 processes prepared for launch   0 sec
1 of 1 processes prepared for launch   0 sec
releasing clients
   1       1632    26.17 MB/sec  warmup   1 sec  latency 41.605 ms
   1       6324    10.67 MB/sec  execute   1 sec  latency 49.044 ms
   1       7914     9.65 MB/sec  execute   2 sec  latency 43.207 ms
   1      10438    11.07 MB/sec  execute   3 sec  latency 44.466 ms
   1      12137    10.74 MB/sec  execute   4 sec  latency 41.928 ms
   1      14928    11.47 MB/sec  execute   5 sec  latency 49.490 ms
   1      16572    10.67 MB/sec  execute   6 sec  latency 41.723 ms
   1      18549    10.95 MB/sec  execute   7 sec  latency 43.159 ms
   1      20780    11.10 MB/sec  execute   8 sec  latency 44.157 ms
   1      22248    10.68 MB/sec  execute   9 sec  latency 42.886 ms
   1  cleanup  10 sec
   0  cleanup  10 sec

 Operation      Count    AvgLat    MaxLat
 ----------------------------------------
 NTCreateX       3682     0.016     0.112
 Close           2704     0.003     0.068
 Rename           155     0.050     0.084
 Unlink           743     0.030     0.165
 Qpathinfo       3335     0.010     0.108
 Qfileinfo        580     0.001     0.006
 Qfsinfo          609     0.005     0.015
 Sfileinfo        298     0.013     0.035
 Find            1288     0.036     0.104
 WriteX          1809     0.043     0.224
 ReadX           5780     0.009     0.098
 LockX             12     0.005     0.009
 UnlockX           12     0.003     0.004
 Flush            253    37.989    49.477

Throughput 10.6793 MB/sec  1 clients  1 procs  max_latency=49.490 ms
```

Performance Benchmarking Tools

- Postmark
  - ✓ "# apt-get install postmark"를 통해 postmark를 install한다.
  - ✓ Postmark는 기존의 대용량 파일을 읽고 쓰고 속도를 측정하는 방법이 아닌 대량의 파일을 생성/읽기/삭제를 수행하면서 작은 파일들에 대한 처리능력을 파악하기위한 benchmark tool 이다.
    - Mail server와 같은 작은 파일들이 지속적으로 읽고 쓰여지는 환경 혹은 DB server와 비슷한 환경에서 주된 performance를 측정할 수 있는 benchmark이다.

- Postmark option
  - ✓ Postmark의 경우 수행을 하기위해서 이전의 benchmark tool들과는 달리 command line에 "#postmark"를 입력하여 수행한 후, option을 설정하고 run하는 방식이다.

```
root@donghwa-com:/home/donghwa# postmark
PostMark v1.51 : 8/14/01
pm>help
set size - Sets low and high bounds of files
set number - Sets number of simultaneous files
set seed - Sets seed for random number generator
set transactions - Sets number of transactions
set location - Sets location of working files
set subdirectories - Sets number of subdirectories
set read - Sets read block size
set write - Sets write block size
set buffering - Sets usage of buffered I/O
set bias read - Sets the chance of choosing read over append
set bias create - Sets the chance of choosing create over delete
set report - Choose verbose or terse report format
run - Runs one iteration of benchmark
load - Read configuration file
show - Displays current configuration
help - Prints out available commands
quit - Exit program
```

postmark benchmark tool을 수행한 후 **help** 명령어 입력시 **postmark**가 지원하는 다양한 **option**들을 확인할 수 있다.

Performance Benchmarking Tools

- Postmark example

```
PostMark v1.51 : 8/14/01
pm>set size 10240
pm>set number 500
pm>set location /mnt/fat_test
pm>set read 512
pm>set write 1024
pm>show
Current configuration is:
The base number of files is 500
Transactions: 500
Files are 10.00 kilobytes in size
Working directory:
        /mnt/fat_test (weight=1)
Block sizes are: read=512 bytes, write=1.00 kilobytes
Biases are: read/append=5, create/delete=5
Using Unix buffered file I/O
Random number generator seed is 42
Report format is verbose.
```

Benchmark를 하기위한 **option** 설정

자신이 설정한 **benchmark option**을 확인하는 "show" option을 통해 **benchmark configuration** 확인

```
pm>run
Creating files...Done
Performing transactions.........Done
Deleting files...Done
Time:
        1 seconds total
        1 seconds of transactions (500 per second)

Files:
        764 created (764 per second)
                Creation alone: 500 files (500 per second)
                Mixed with transactions: 264 files (264 per second)
        262 read (262 per second)
        0 appended (0 per second)
        764 deleted (764 per second)
                Deletion alone: 528 files (528 per second)
                Mixed with transactions: 236 files (236 per second)

Data:
        2.56 megabytes read (2.56 megabytes per second)
        7.46 megabytes written (7.46 megabytes per second)
```

Benchmarking result

Performance Benchmarking Tools

- Bonnie
  - ✓ "# apt-get install bonnie++"를 통해 bonnie++를 install한다.
  - ✓ Bonnie는 disk I/O를 benchmarking하는 프로그램으로 c로 제작되었으며 이를 기반으로 c++로 만든것이 Bonnie++이다. Bonnie는 HDD와 file system의 성능의 테스트 목적으로 만들어졌다.
  - ✓ 프로그램 실행을 통해 sequential/random에 대한 자료가 함께 나오므로 서로 다른 file system을 간단히 비교하는데 도움이 될 수 있다.
  - ✓ 그렇지만 여러 개의 테스팅을 동시에 진행하므로 하나의 시스템에서 특정한 옵션을 바꾸어가면서 성능을 테스트하기에는 적합하지 않다.

Performance Benchmarking Tools

- Bonnie option
  - ✓ Command line에서 "# bonnie"를 입력하면 bonnie에 존재하는 옵션들 확인 가능.

```
usage:
bonnie++ [-d scratch-dir] [-c concurrency] [-s size(MiB)[:chunk-size(b)]]
       [-n number-to-stat[:max-size[:min-size][:num-directories[:chunk-size]]]]
       [-m machine-name] [-r ram-size-in-MiB]
       [-x number-of-tests] [-u uid-to-use:gid-to-use] [-g gid-to-use]
       [-q] [-f] [-b] [-p processes | -y] [-z seed | -Z random-file]
       [-D]
```

  - ✓ -d option : 테스트에 사용될 directory를 지정
  - ✓ -s option : I/O performance 처리를 위한 MB단위의 파일 크기. 만약 파일의 크기가 1G 보다 크다면 여러개의 파일로 저장이된다. Chunk-size는 byte 단위의 크기를 측정한다.
  - ✓ -n option : 생성되는 파일의 수. Default로 *1024개의 파일이 생성된다.
  - ✓ -m option : 장비의 이름. 단순히 보여주기 위한 목적으로 사용된다.
  - ✓ -r option : memory size를 의미한다.
  - ✓ -x option : 테스트의 실행횟수를 의미하며 여러 번의 테스트를 하는 경우 유용하다.
  - ✓ -u/g option : user/group id 옵션.
  - ✓ -f option : 빠른 실행 옵션으로써, character I/O test를 수행하지 않는다.
  - ✓ -p option : 실행을 위한 프로세스의 수.

Performance Benchmarking Tools

- Bonnie example

 # bonnie –d (mounting point) –m (test_name) –s (file_size:block_size) –n (number_of_files) –r (memory size) –u (uid:gid)

```
Version  1.96       ------Sequential Output------ --Sequential Input- --Random-
Concurrency   1     -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
Machine    Size:chnk K/sec %CP K/sec %CP K/sec %CP K/sec %CP K/sec %CP  /sec %CP
bonnie_test  2G:512  1761  99 433671  99 390131  99  5036  99 1168301  99 11694 139
Latency              1162us       42us       66us      113us       20us       35us
Version  1.96       ------Sequential Create------ --------Random Create--------
bonnie_test         -Create-- --Read--- -Delete-- -Create-- --Read--- -Delete--
               files  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP
                  10   266  99 +++++ +++  3819  99   420  99 +++++ +++   991  99
Latency               107ms     2930us    1889us    6700us      31us    3401us
```

✓ Bonnie benchmark의 결과 값은 한 눈에 알아보기 힘든 형식으로 출력된다. 이를 위해 redirection을 통해 text 파일을 만든 하더라도 한눈에 들어오는 명확한 형태로 보이지 않는다.

```
Version  1.96       ------Sequential Output------ --Sequential Input- --Random-
Concurrency   1     -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
Machine    Size:chnk K/sec %CP K/sec %CP K/sec %CP K/sec %CP K/sec %CP  /sec %CP
bonnie_test  2G:512  1780  99 428471  99 308453  99  4989  99 877876  99 +++++ +++
Latency              1176us      583us    15204us      244us      123us     6414us
Version  1.96       ------Sequential Create------ --------Random Create--------
bonnie_test         -Create-- --Read--- -Delete-- -Create-- --Read--- -Delete--
               files  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP
                  10   267  99 +++++ +++  3745  99   416  99 +++++ +++   998  99
Latency               135ms     1820us    1786us    7111us      18us    3365us
```


Performance Benchmarking Tools

- Bonnie example
  - ✓ Bonnie의 benchmarking 결과를 명확히 한 눈에 보기위해 bon_csv2html, bon_csv2txt를 사용할 수 있다.
  - ✓ Bonnie의 benchmarking 결과의 마지막줄에 일련의 숫자와 문자열들이 조합된 문장이 출력된다. 이는 benchmarking 결과 값들만을 나열한 [RESULT]이다.
  - ✓ [RESULT] 값을 복사한 후 다음의 명령어를 통하여 좀 더 정돈된 결과물을 확인할 수 있다.

 # echo [RESULT] | bon_csv2html or bon_csv2txt > (결과물이 생성될 디렉토리)/(결과물의 이름)

```
Version      1.96   ------Sequential Output------ --Sequential Input- --Random-
                    -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
Machine        Size K/sec %CP K/sec %CP K/sec %CP K/sec %CP K/sec %CP  /sec %CP
bonnie_test  2G:512  1747  99 428719  99 381653  99  5029  99 1124582  99 +++++ +++
Latency              1169us      41us      41us     127us      19us    7129us
                    ------Sequential Create------ --------Random Create--------
                    -Create-- --Read--- -Delete-- -Create-- --Read--- -Delete--
files:max:min        /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP
bonnie_test      10   274  99 +++++ +++  3801  99   420  99 +++++ +++  1011  99
Latency              107ms    1715us    2069us    7337us      12us    3391us
```

Version 1.96			Sequential Output						Sequential Input		Random Seeks			Sequential Create						Random Create					
	Size	Chunk Size	Per Char		Block		Rewrite		Per Char		Block				Num Files	Create		Read		Delete		Create		Read		Delete	
			K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	/sec	% CPU		/sec	% CPU	/sec	% CPU	/sec	% CPU	/sec	% CPU	/sec	% CPU	/sec	% CPU
bonnie_test	2G	512	1747	99	428719	99	381653	99	5029	99	1124582	99	+++++	+++	10	274	99	+++++	+++	3801	99	420	99	+++++	+++	1011	99
	Latency		1169us		41us		41us		127us		19us		7129us		Latency	107ms		1715us		2069us		7337us		12us		3391us	