一. Scheduling 每时每刻决定哪些线程可以访问资源（一般为 cpu time）

1. Scheduling assumptions:One program per user;One thread per program;Programs are independent

2. Execution model:programs 在 CPU and I/O 之间交替（我们的目标主要集中在 use cpu 的时刻）

3. Scheduling goals:Minimize Response Time（减少去做某一项工作的运行时间，在用户视角表现为在编辑器中敲键的时间或是编译程序的时间）;Maximize Throughput（实现：最小化开销比如 context switch 或是更高效的使用资源。吞吐量与响应时间相关，当我想最大化吞吐量时，那么 short jobs 将会变多那么 context switch 就会变多，但前者所导致的文本切换小于后者）;Fairness（以公平的方式在用户之间共享 CPU,但并不能最小化平均响应时间）

4. Scheduling 算法

（1）FCFS/FIFO（First-Come, First Served 不可抢占算法）Pros: simple；Cons: short jobs get stuck behind long ones（Convoy Effort: short process behind long process）当从小到大执行或 jobs 等长时执行，均为最优解

（2）Round-Robin（可抢占算法 with a time quantum q）:若有 n 个 process,No process waits more than (n-1)q time units.Pros: Better for short jobs,Fair;Cons:Context switching overhead for long jobs.当 q really large->FCFS,将会影响 response time;small->interleaved,将会影响吞吐量;并且因为 context switch 的开销,q 要足够大。

Cache state must be shared among all jobs with RR but can be devoted to each job with FCFS.当所有 jobs 等长，那么 RR 的 performance 要更差

（3）Priority（优先运行优先级更高的 jobs），但存在饥饿（低优先级得不到执行）和死锁（优先级反转）的问题，可用动态优先级来进行调整同时改善 fairness 情况/给每个队列分配对应比例的 cpu 资源（pintos project1）

> Every thread has a *nice* value between -20 and 20 directly under its control. Each thread also has a priority, between 0 (PRI_MIN) through 63 (PRI_MAX), which is recalculated using the following formula every fourth tick:
>
> *priority* = PRI_MAX - (*recent_cpu* / 4) - (*nice* * 2).
>
> *recent_cpu* measures the amount of CPU time a thread has received "recently." On each timer tick, the running thread's *recent_cpu* is incremented by 1. Once per second, every thread's *recent_cpu* is updated this way:
>
> *recent_cpu* = (2*load_avg)/(2*load_avg + 1) * *recent_cpu* + *nice*.
>
> *load_avg* estimates the average number of threads ready to run over the past minute. It is initialized to 0 at boot and recalculated once per second as follows:
>
> *load_avg* = (59/60)*load_avg + (1/60)*ready_threads.
>
> where *ready_threads* is the number of threads that are either running or ready to run at time of update (not including the idle thread).

（4）Lottery Scheduling（用以实现 fairness）给每个 job 都分配至少一张彩票以避免饥饿，short 分配更多，long 分配更少，每个 time slice 选择一个赢家，最终以实现 cpu time 成比例的分配给每个 job。当 load changes 时表现也要优于优先级调度因为成比例的原因。当太多 short jobs 时会影响 response time 时，让一些 users 登出

（5）SJF/STCF(Shortest Time to Completion First,FCFS 的最优 ordering)

（6）SRTF/SRTCF(Shortest Remaining Time First,SJF 的可抢占版本)但会引起饥饿问题，因为 short 优先执行。Pros: Optimal (average response time) -Cons: Hard to predict future -Cons: Unfair

当已知未来的情况下（5）（6）两种方法将会表现出极优的 performance 来降低 average response time

（7）Multilevel Queue:Processes assigned to one queue permanently;Scheduling must be done between the queues:Fixed priority and Time Slice

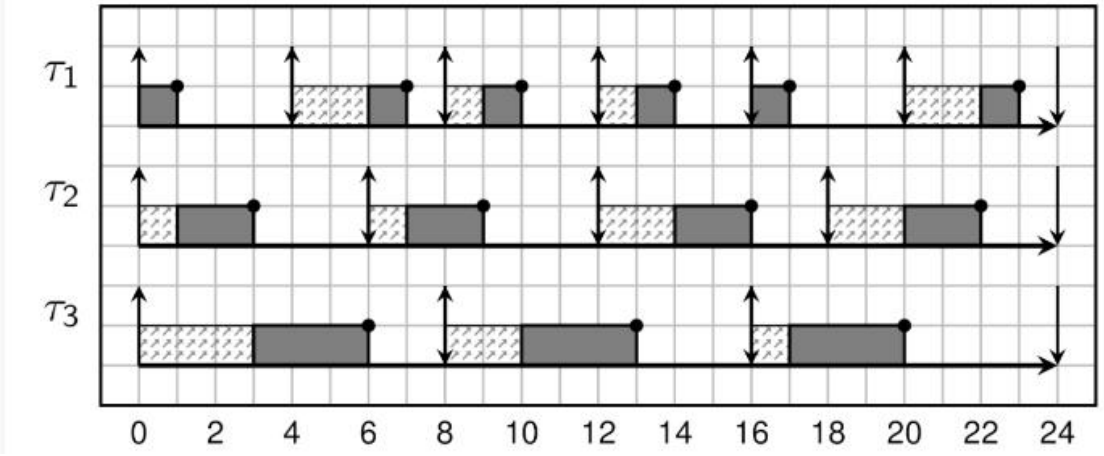
（8）Multilevel Feedback Queue:Multilevel queue, each with different priorities;Each queue has its own scheduling algorithm;A process can move between the queues;Scheduling must be done between the queues:Fixed priority scheduling and Time Slice(queue 的优先级高则表示为“foreground task”反之为“background task”)

（9）CFS（Completely Fair Scheduler）priorities reflected by weights such that increasing a task’s priority by

1 always gives the same fractional increase in CPU time - regardless of current priority

（10）Real-Time Scheduling,predictability is essential. Hard real-time computing(Attempt to meet all deadlines):EDF and RMS/DM;Soft real-time computing(Attempt to meed deadlines with high probability):CBS

（11）EDF(Earliest Deadline First),Tasks periodic with period P and computation C in each period for each task i 一般来说 P 即为对应的周期性 ddl，C 是其完成所需要的时间，ddl 优先



（12）如何评估一个调度算法：•Deterministic modeling –Takes a predetermined workload and compute the performance of each algorithm for that workload •Queueing models –Mathematical approach for handling stochastic workloads (Distributions of CPU and I/O bursts -> formula: n = lamda×W – n: average queue length;lamda: average arrival rate;W: average waiting time in queue)•Implementation/ Simulation –Build system which allows actual algorithms to be run against actual data - most flexible/general

二. Memory management

- Big-endian: **高位**字节存入**低地址**，**低位**字节存入**高地址**
- Little-endian: **低位**字节存入**低地址**，**高位**字节存入**高地址**

例如，将 12345678h 写入 1000h 开始的内存中，以**大端序**和**小端序**模式存放结果如下

表 1.2 以 Big-endian 和 Little-endian 模式存放的结果

存放顺序	1000h	1001h	1002h	1003h
Big-endian	12h	34h	56h	78h
Little-endian	78h	56h	34h	12h

如图 1.2 所示是一种更加直观的描述。

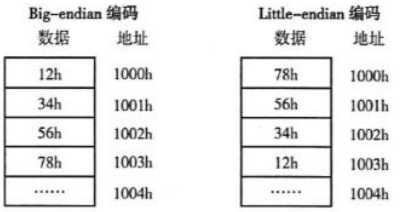


图 1.2 Big-endian 与 Little-endian 内存存储方式

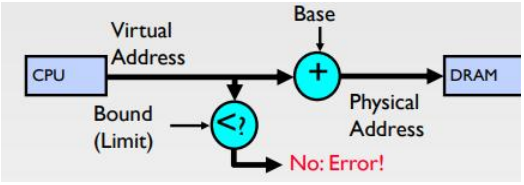
1. 一个程序执行的准备:Compile time (i.e. “gcc”) –Link/Load time (UNIX “ld”) –Execution time (e.g. dynamic libs)

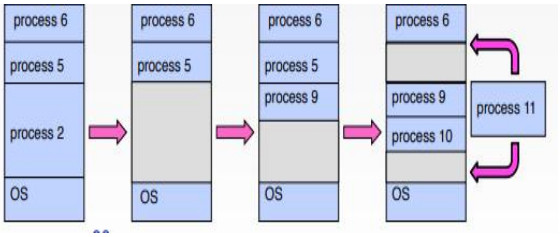
2. Protection ways for multi-programming:

Without traslation:

(1) base and bound:use two special registers BaseAddr and LimitAddr to prevent user from straying outside designated area;During switch, kernel loads new base/limit from PCB and users are not allowed.

但是 BB method 存在很多缺点:碎片化 fragmentation problem;Missing support for sparse address space;Hard to do inter-processsharing



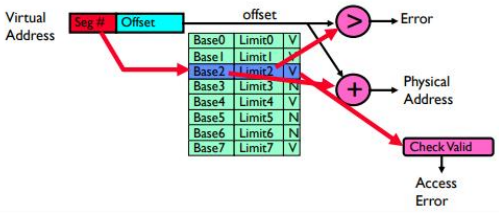


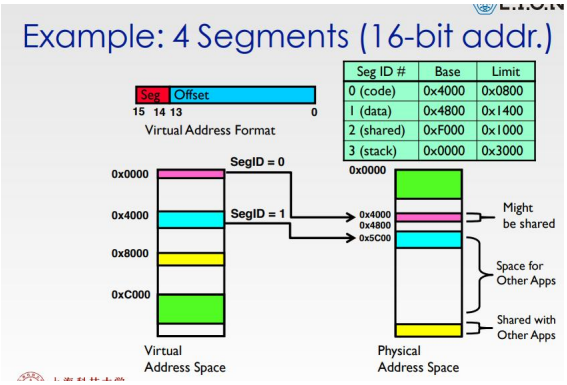
With address translation(much easier to implement protection/isolation and easy to manage 比如可以进行 sharing):every program can be linked/ loaded into same region of user address space

(2) Segmentation:多个分割的 segments 且每个 segment is given region of contiguous memory.

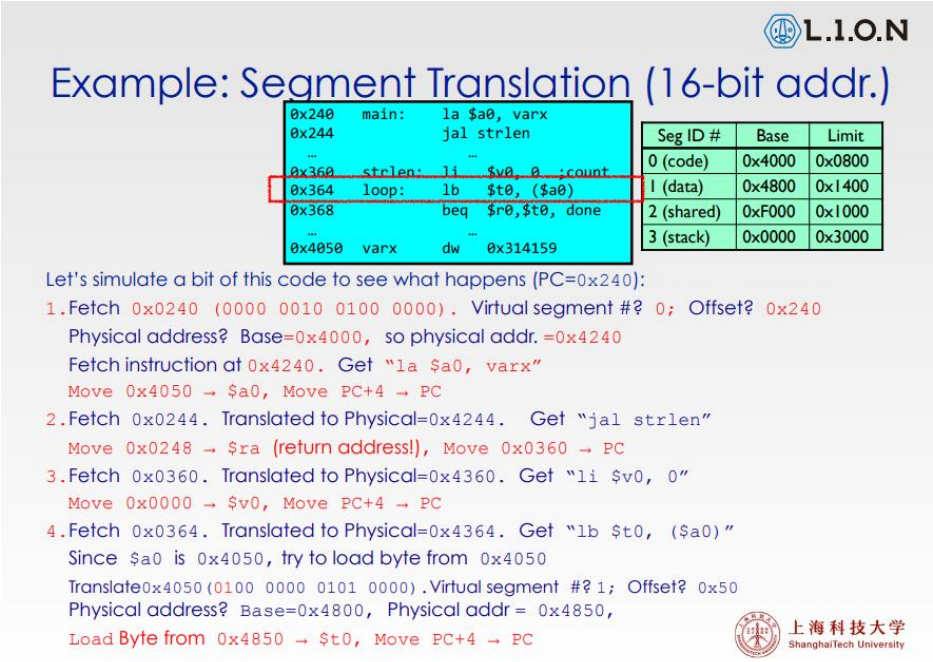
实现具体细节：Segment map resides in processor －Segment number mapped into base/limit pair －Base added to offset to generate physical address －Error check catches offset out of range.

表现：Virtual address space has holes 碎片化;Need protection mode in segment table;当 context switch 需要存取哪些部分:Segment table stored in CPU, not in memory and might store all of process' memory onto disk(called "swapping", 进一步提升开销)



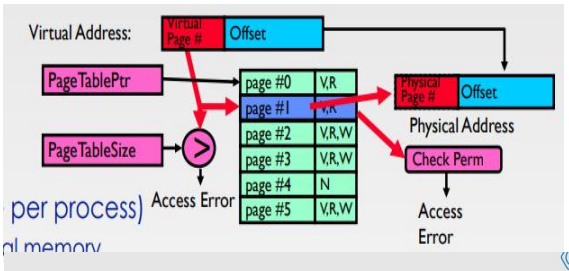


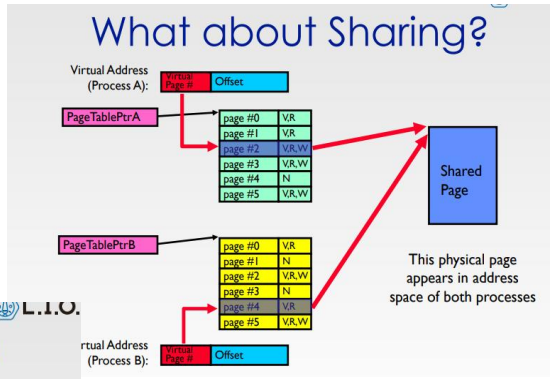
但 segmentation 也存在许多问题:Must fit variable-sized chunks into physical memory;May move processes multiple times to fit everything;Limited options for swapping to disk;Fragmentation: wasted space －External: free gaps between allocated chunks －Internal: don't need all memory within allocated chunks

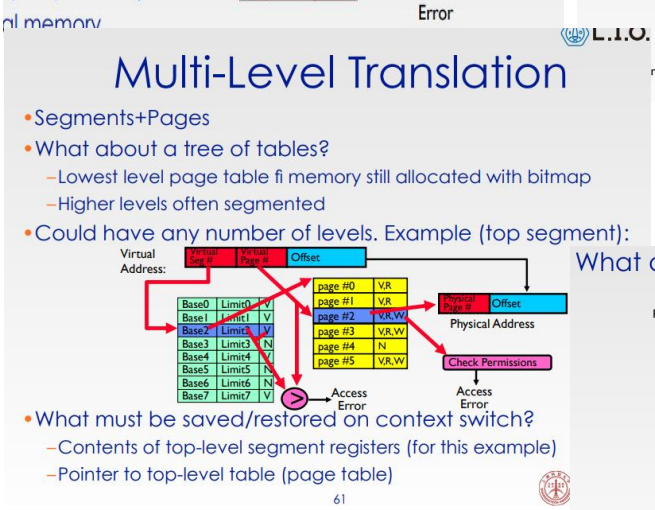


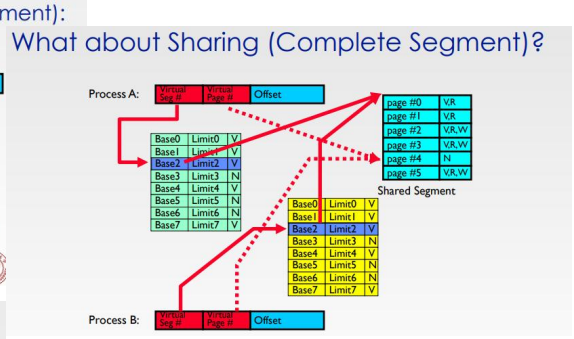
(3) Page:Physical Memory in Fixed Size Chunks. Cons:访存速度慢（当存在多级时需要多次访问）;hardware complex;能够解决 external fragmentation 但是只能缓解 internal fragmentation .Pros:Simple memory allocation and Easy to share. 当 context switch 时需要切换什么:Page table pointer and limit

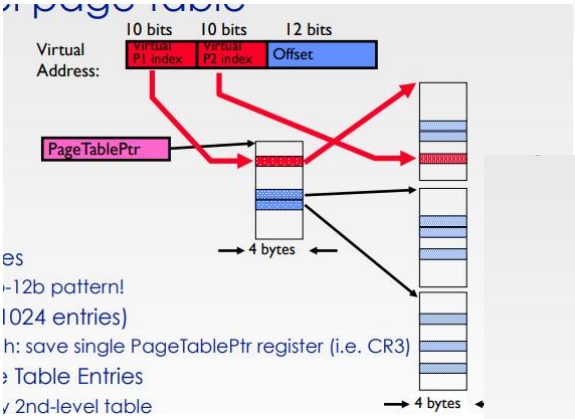
-》Multilevel translation:Pros: Only need to allocate as many page table entries as we need for application;Easy memory allocation;Easy Sharing.Cons: One pointer per page;Page tables need to be contiguous;－Two (or more, if >2 levels) lookups per reference.

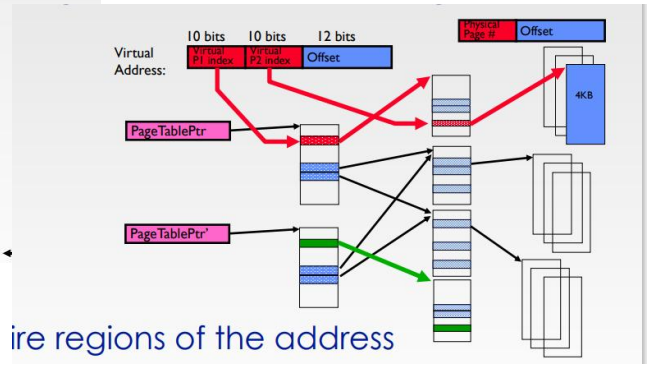








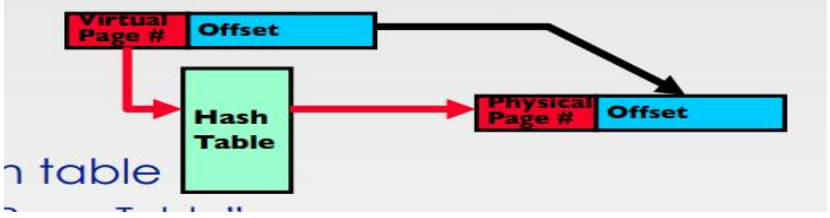




-》inverted page table（哈希表）

反置页表的 entry 数量与物理内存的页框数量相同，而与 vm 无关

Cons: －Complexity of managing hash chains: Often in hardware! －Poor cache locality of page table



L.I.O.N

Address Translation Comparison

	Advantages	Disadvantages
Simple Segmentation	Fast context switching: Segment mapping maintained by CPU	External fragmentation
Paging (single-level page)	No external fragmentation, fast easy allocation	Large table size ~ virtual memory Internal fragmentation
Paged segmentation	Table size ~ # of pages in virtual memory, fast easy allocation	Multiple memory references per page access
Two-level pages		
Inverted Table	Table size ~ # of pages in physical memory	Hash function more complex No cache locality of page table

3. 如何使用 pte:(1)Demand Paging －Keep only active pages in memory －Place others on disk and mark their PTEs invalid(2)Copy on Write(when fork)－ Make copy of parent’s page tables (point at same memory) － Mark entries in both sets of page tables as read only － Page fault on write creates two copies

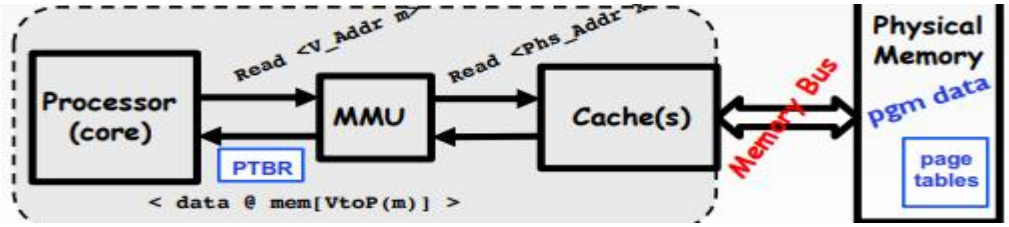
(3)Zero Fill On Demand－New data pages must carry no information (say be zeroed) －Mark PTEs as invalid; page fault on use gets zeroed page －Often, OS creates zeroed pages in background

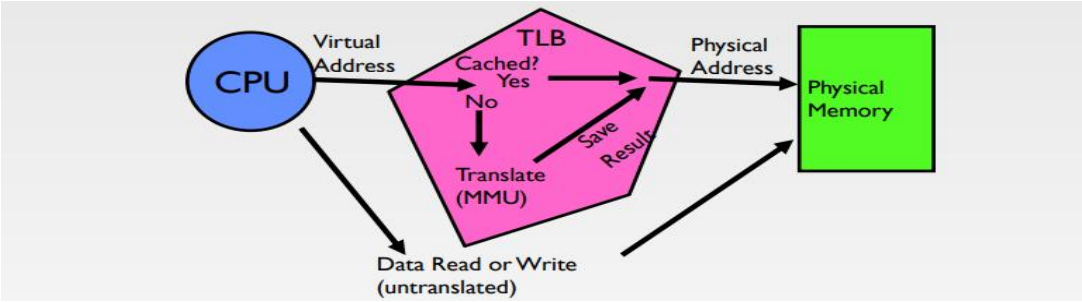
三. Cache and TLB and related disk(secondary memory)

1.MMU(Memory management unit):On every reference (I-fetch, Load, Store) read (multiple levels of) page table entries to get physical frame or FAULT －Through the caches to the memory －Then read/write the physical location

2.Cache:Speedup the address referencing,最重要的特性是时间与空间 locality

Average Memory Access time（AMAT） = (Hit Rate x Hit Time) + (Miss Rate x Miss Time)



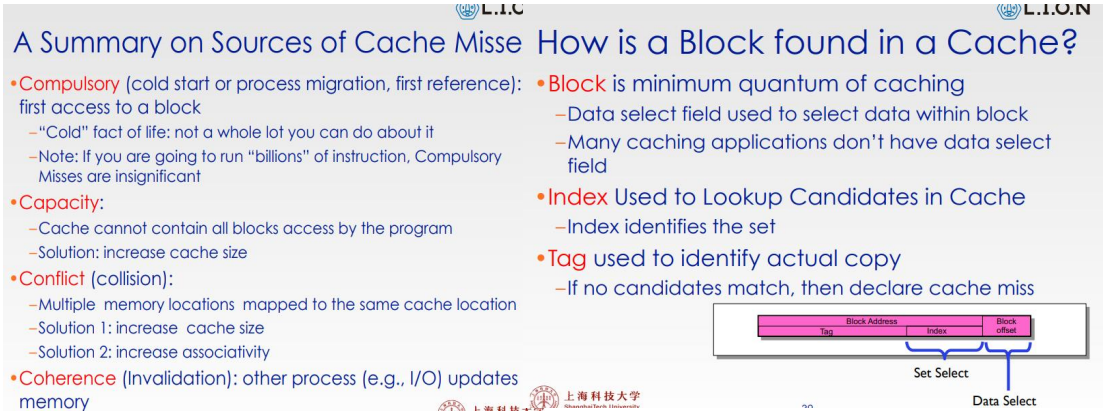


3.TLB（Translation Look-Aside Buffer）:访存更快，且冲突变少，否则 miss time 是很高的因为 TLB 的引入。On a TLB miss,

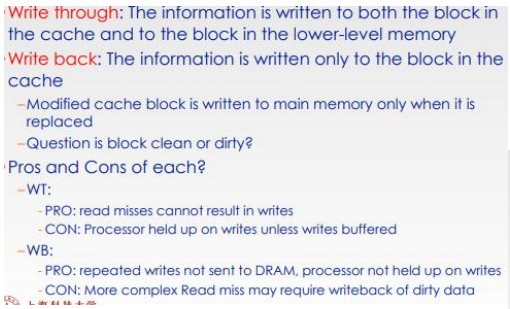
the page tables may be cached, so only go to memory when both miss.

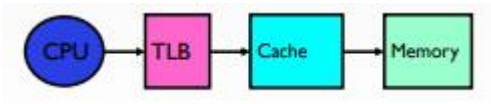
->Thrashing: continuous conflicts between accesses（If a process does not have “enough” pages, the page-fault rate is very high）进而导致 a process is busy swapping pages in and out with little or no actual progress，最终表现为低 cpu 使用率和 Os 花费很多的时间在 swapping to disk

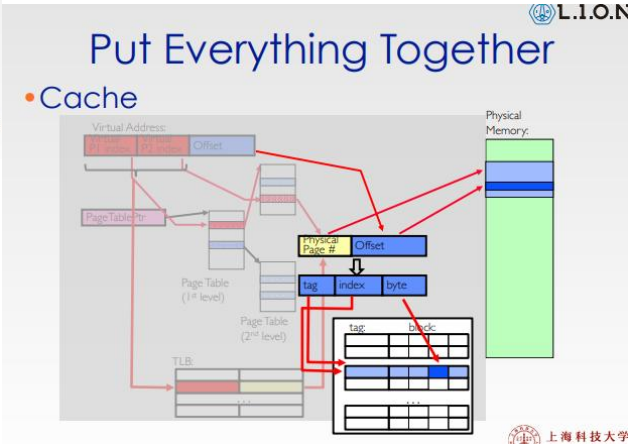
->TLB consistency:当文本切换时,因为 address space 发生改变,所以 tlb entry 变为 invalid,通过多加 bits 来存储 processid 在 tlb 中来实现. 当 translation tables change，也要 invalidate.



Cache Organizations: -Direct Mapped: single block per set -Set associative: more than one block per set -Fully associative: all entries equivalent







4.Page fault：Virtual-to-Physical Translation fails 具体过程为当 process 处理 instruction 时传入虚拟地址到 mmu 中去，再到物理地址时 fails 将由 mmu 引起 page fault，接着引起 exception 让 Os 跳转至 page fault handler 再 load page from disk 去 update 物理页表 entry，结束后跳转至 scheduler 去 retry 这条指令。

-》demand paging(one of use of pte)：有着多种用途

•Extend the stack -Allocate a page and zero it

•Extend the heap (sbrk of old, today mmap)

•Process Fork

-Create a copy of the page table

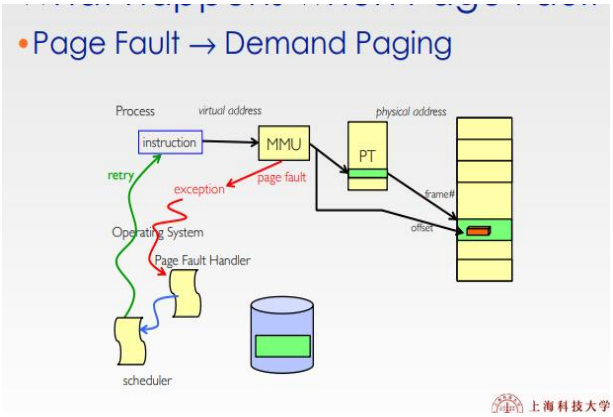
-Entries refer to parent pages

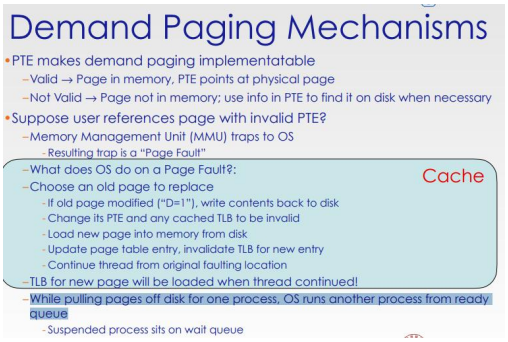
- NO-WRITE -Shared read-only pages remain shared

-Copy page on write

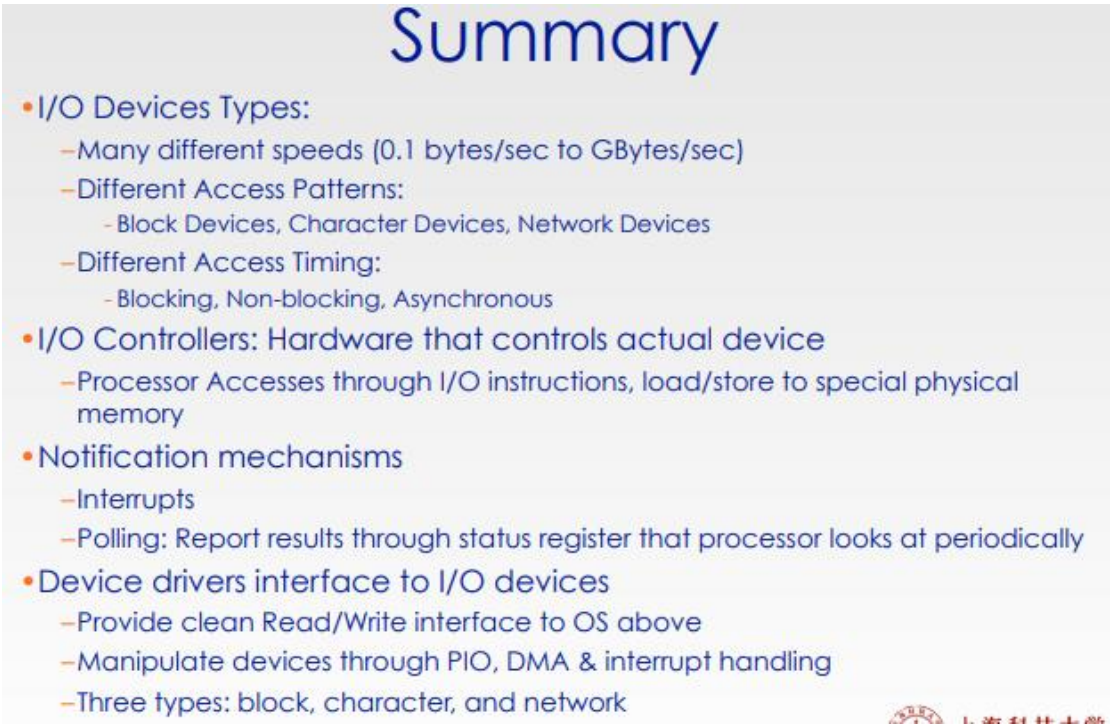
•Exec -Only bring in parts of the binary in active use -Do this on demand

-》额外的一种 miss for page :Policy Misses: Caused when pages were in memory, but kicked out prematurely because of the replacement policy





四. I/O



1.设备的标准接口

•Block Devices: e.g. disk drives, tape drives, DVD-ROM －Access blocks of data －Commands include open(), read(), write(), seek() －Raw I/O or file-system access －Memory-mapped file access possible

•Character Devices: e.g. keyboards, mice, serial ports, some USB devices －Single characters at a time －Commands include get(), put() －Libraries layered on top allow line editing

•Network Devices: e.g. Ethernet, Wireless, Bluetooth －Different enough from block/character to have own interface －Unix and Windows include socket interface －Separates network protocol from network operation －Includes select() functionality －Usage: pipes, FIFOs, streams, queues, mailboxe

2.用户如何处理时间

•Blocking Interface: “Wait” －When request data (e.g. read() system call), put process to sleep until data is ready －When write data (e.g. write() system call), put process to sleep until device is ready for data

•Non-blocking Interface: “Don’t Wait” －Returns quickly from read or write request with count of bytes successfully transferred －Read may return nothing, write may write nothing

•Asynchronous Interface: “Tell Me Later” –When request data, take pointer to user’s buffer, return immediately; later kernel fills buffer and notifies user –When send data, take pointer to user’s buffer, return immediately; later kernel takes data and notifies user

3.I/O 设备如何通知 OS 通过 I/O interrupt 或 polling 来实现

->I/O interrupt: –Device generates an interrupt whenever it needs service –Pro: handles unpredictable events well –Con: interrupts relatively high overhead

->polling:OS periodically checks a device-specific status register – I/O device puts completion information in status register –Pro: low overhead –Con: may waste many cycles on polling if infrequent or unpredictable I/O operations

五. File System 文件系统

1.File(permanent storage,Os 抽象出来的概念):含有 data(blocks on disk)和 metadata（attributes）

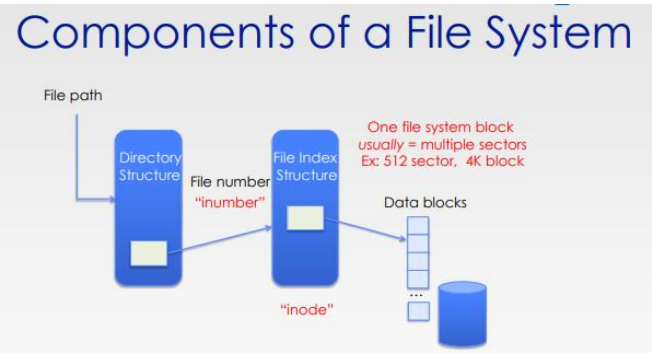
(1)File Attributes:•Name •Identifier •Type •Location •Size •Protection •Time, Date, User Identification •Information about files are kept in the directory structure, maintained on disk

(2)File operations:-Create –Write –Read –Reposition within file – file seek –Delete –Truncate –Open/Close

2.Directory:每个 dir entry 由 file 和 dir 组成，且有 name 和 attribute

(1)information:•File name •File type •Address or location •Current length •Maximum length •Date created, last accessed, last updated •Owner ID, Protection Information

(2)operation:•Search for a file •Create a file •Delete a file •List a directory •Rename a file •Traverse file system

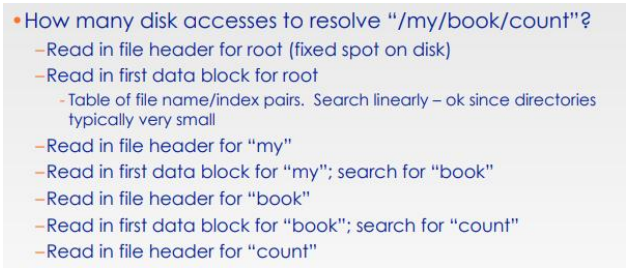


Open performs Name Resolution:首先将 pathname 解析为 filenumber 即 index 去索引对应的 inode 进而对应的 sector；在 kernel 的 PCB 中创建一个 file descriptor;最后返回一个 handle 到用户进程

Read, Write, Seek, and Sync operate on handle:Mapped to file descriptor and to blocks

-》如何解析一个路径

-》Current working directory: Per-address-space pointer to a directory (inode) used for resolving file names，这也就引入了相对路径概念



课本给出的一个解析路径的例子

(b) File System (10pts)

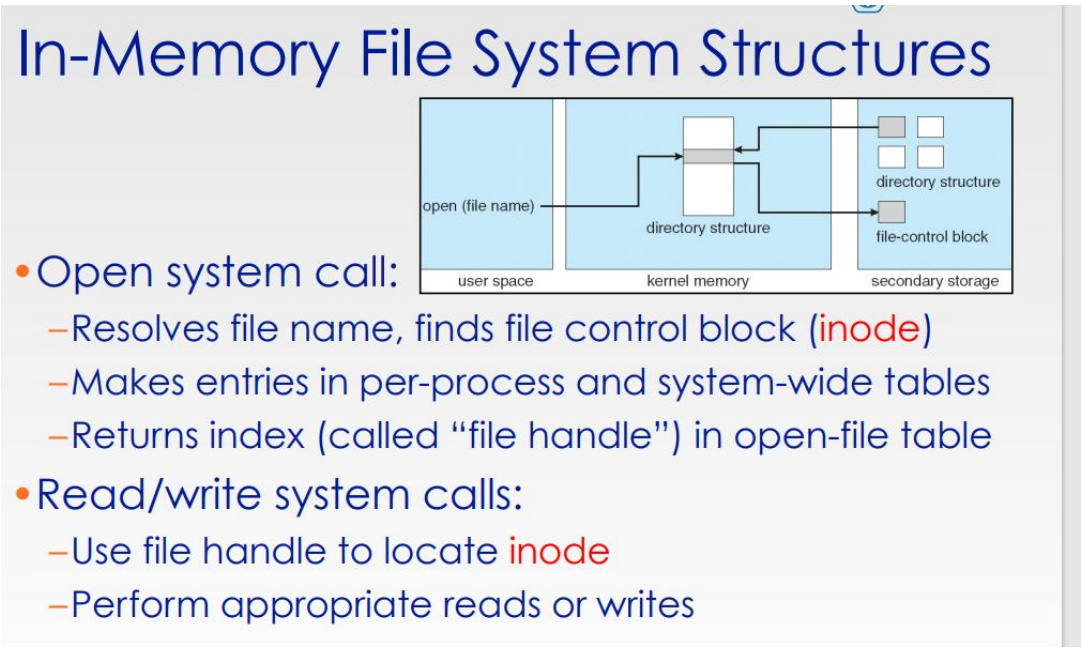
Consider a Unix File System. It uses inodes to index directories and files. Assume only the address of the data block for root directory (not the inode of root directory) is cached. All other contents stored in the File System are not cached. Assume the content of directories can be stored in only one data block.

1. How many disk operations are needed to read the first byte of a file with absolute path: /home/usr/myfile.txt? Please explain your answer.

7 operations are needed:
Read / requires no disk operation (cached).
1.Read the first data block of ”/” for the inode number of ”home”
2.Read the inode of ”home”
3.Read the first data block of ”home” for the inode number of ”usr”
4.Read the inode of ”usr”
5.Read the first data block of ”usr” for the inode number of ”myfile.txt”
6.Read the inode of ”myfile.txt” for the address of the first data block
7.Read the first byte

HW2 中给出的一个解析路径并且读取数据的例子

值得注意的是该题 root dir 的 data block 地址已经被 cache 所以我们无需读取 root 的 inode 去获得相关信息而是直接从 cache 读取即可



3.File system:

(1)FAT（File Allocation Table，链表结构需遍历，其中 file 是 diskblocks 的集合）:一般存储在 Disk！！！/on boot cache in memory,

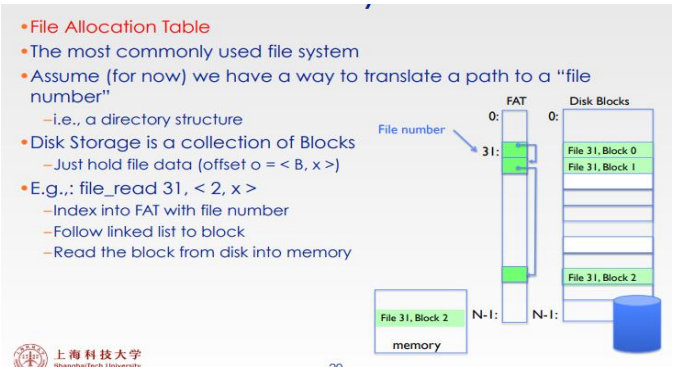
->利用 name 转成的 number 对对应 file 的 blocklist 来进行索引，再通过 offset 来遍历找到对应的 block

->没有用到的 block 标记为 free，当写回到/创建新的 block 时，抓取一个 free block 然后将其与 file 连接

->当格式化 disk 时，将 fat 的所有 entry 标记为 free，并将 block 变为 0

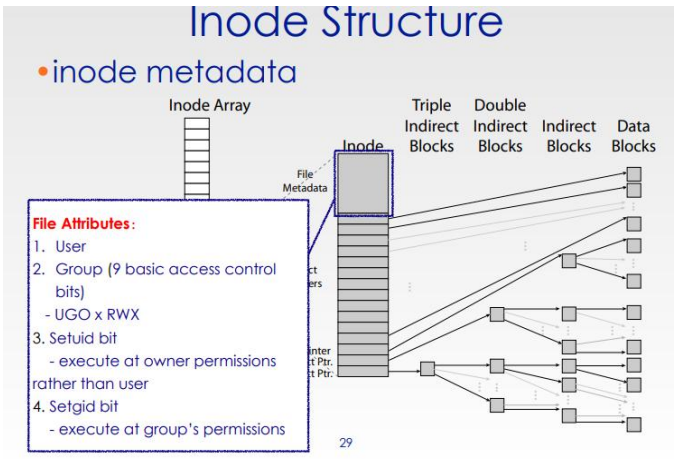
->如何找到根目录"/":At well-defined place on disk/For FAT, this is at block 2 (there are no blocks 0 or 1)

->FAT 存在很大的安全漏洞•FAT has no access rights•FAT has no header in the file blocks•Just gives an index into the FAT to read data



4.Inode（file control block）（记住一定不含 filename！！！）

For inode, it will include three parts one is file metadata and others are direct and indirect pointers.



2. Assume the disk block size is 8KB, and the disk block pointers require 4 bytes. Within the inode there are 12 direct pointers, as well as one single and one double indirect disk pointers. What is the maximum size of a file that this file system can support? Please write down the full calculating expressions.

There can be 8KB/4B = 2048 pointers in a page.

Direct pointers: 12 * 8KB = 96KB

Single indirect pointer: 1 * 2048 * 8KB = 16384KB = 16MB

Double indirect pointer: 1 * 2048 * 2048 * 8KB = 33554432KB = 32768MB = 32GB

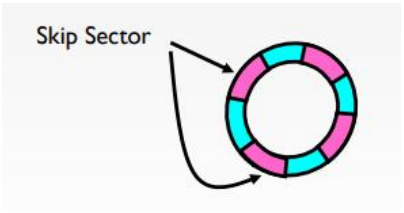
So the max file size = 12 * 8KB + 2048 * 8KB + 2048 * 2048 * 8KB = 33570912KB = 96KB + 16MB + 32GB

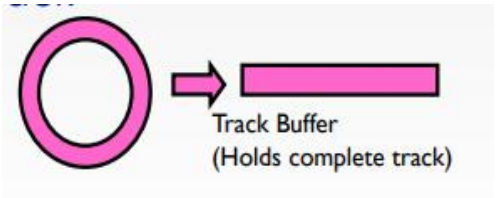
->但存在两个问题：

->Problem 1: When create a file, don’t know how big it will become

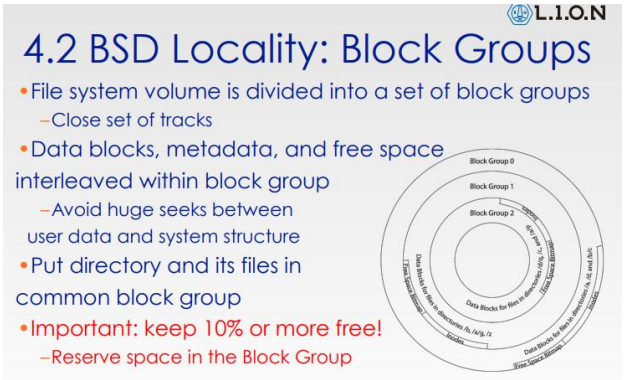
->Problem 2: Missing blocks due to rotational delay

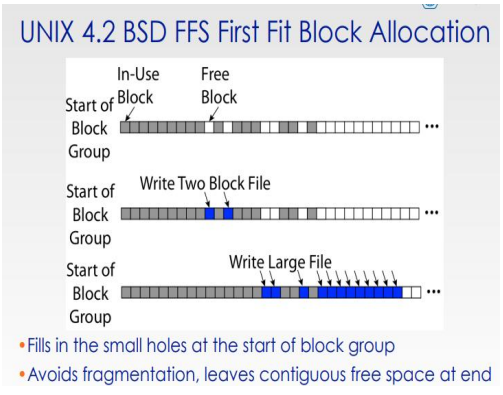
->Solution1: Skip sector positioning (“interleaving”) •Place the blocks from one file on every other block of a track: give time for processing to overlap rotation •Can be done by OS or in modern drives by the disk controller





->solution2:Read ahead: read next block right after first, even if application hasn’t asked for it yet •This can be done either by OS (read ahead) •By disk itself (track buffers) - many disk controllers have internal RAM that allows them to read a complete track





->什么时候可以删除文件：Maintain ref-count of links to the file -Delete after the last reference is gone （In pintos open_cnt == 0）

4.软连接和硬链接：

Hard link（不可跨越文件系统）: No different from regular files, inodes all point to the same block of the file on the hard drive.

Sets another directory entry to contain the file number for the file;Creates another name (path) for the file;Each is “first class”.

-》多个 dir entry 指向同一个 inode，全部删除才能彻底删除

Soft link（可跨越）：Saves the absolute path of the file it represents, which is another type of file with independent blocks on the hard drive that replace its own path when accessed

Directory entry contains the path and name of the file;Map one name to another name

-》重新建立一个 inode 但是内容是另一个文件的 path，当源文件删除，则变为死链接

6. NTFS:•可变范围长度•所有都是序列对•direct 和 indirect 自由混合使用•目录默认为 B-tree structure

习题总结：

1. 如何计算需要多少级 level 使得 page table fits into XXX 比如 page 或是 4KB memory

(Virtual bits - offset bits)/ num of page table entry = XXX/page table entry size 向上取整

2.When we acquire a large amount of empty memory (i.e., 32GB at once) in an OS with demand paging, its first access will likely be much slower than its second access (even if we have lots of free memory). Briefly explain the cause and your idea to improve the first access performance

->Demand paging only allocates pages when it is first referenced, not at virtual memory allocation time (especially for large amounts of memory).

->Pre-fault these pages to prevent page faults the first time you access them; Or use larger page sizes to reduce the number of page faults and the levels of page table needed to configure when first accessing them.

4. (T) In Pintos, the directory path /../../../ is valid.

5. (F) Directory entries are stored on the directory's inode on disk.实际上 dir entry 存在对应的 dir structure 缓存在 kernel memory 中，通过该 dir entry 便可以索引到对应的 inode 中，再通过 inode 去索引 disk 中的 data block 来获取相关数据

6. (F) On Unix file systems, the inodes for regular files have different formats than the inodes for directories 实际上在 inode 内部会记录类型来区分是 file 还是 dir

7. (F) Inodes are organized in an array and placed on disk at a random location (or locations).其一 inode 在 array 中是 sequential 的，其二该 array 被存在 disk 上一个连续的特别位置

以 ext2 文件系统为基础

8. 如何计算 How large of a disk can this file system support（与多大 file 不同！！！！！！）

32bits 系统意味着 block 的索引有 2^32，而一个 block 为 4KB，那么所需 disk 容量为 2^32 * 2^12 = 16TB

不同block大小情况下，文件系统的最大容量计算

原创 云疏不知数 于 2021-04-22 13:36:27 发布 阅读量1.4k 收藏 1 点赞数 版权

分类专栏: Linux

Linux 专栏收录该内容 2 订阅 34 篇文章 订阅专栏

在操作系统中，文件系统都是针对分区而言的，一个磁盘必须先分区才能格式化文件系统(即使你将磁盘所有容量划分一个分区).格式文件系统后才能挂载使用，此时就必须知道一个文件系统到底支持多大的分区大小。

在ext2、ext3文件系统中，采用32bit的块索引空间，且其采用int的无符号整型，因此

- **当block为4KiB时，一个分区的最大空间为：2^32*8KByte=32TiB**
- **当block为4KiB时，一个分区的最大空间为：2^32*4KByte=16TiB**
- **当block为2KiB时，一个分区的最大空间为：2^32*2KByte=8TiB**
- **当block为1KiB时，一个分区的最大空间为：2^32*1KByte=4TiB,但ext2文件系统本身的限制，ext2文件系统限制单个文件最大容量为2TiB**

9. 根目录存在第二个 block 即 inode2 中，同时也被称为 master inode

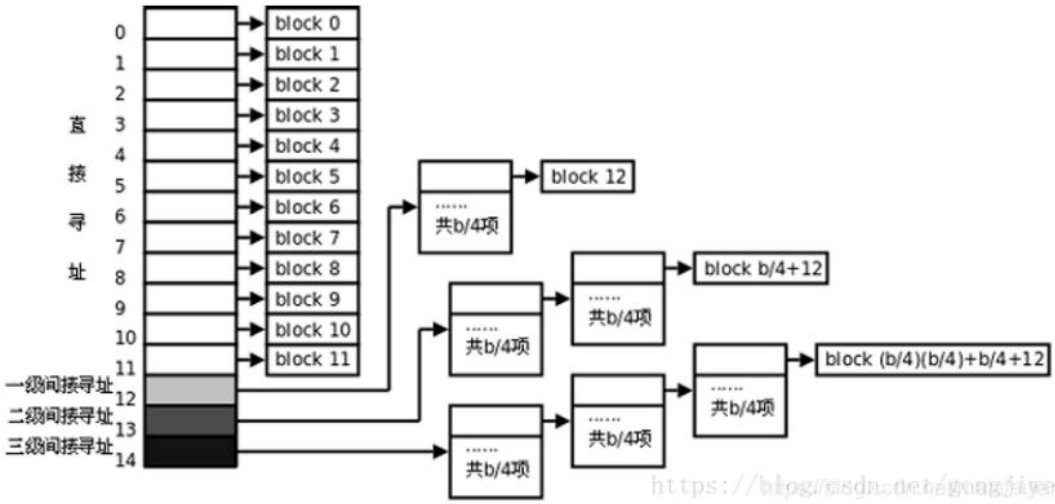
10. 如何在 ext2 文件系统中读取 file 对应的 block

8 operations

- Read inode 2 for "/" (cached)
- Read the first data block of "/" for the inode number of "home"
- Read the inode of "home"
- Read the first data block of "home" for the inode number of "study"
- Read the inode of "study"
- Read the first data block of "study" for the inode number of "cs130.txt"
- Read the inode of "cs130" for the address of indirect block
- Read the indirect block for the 2nd pointer (adress of block 12)
- Read block 12

Eg How many disk read operations are required to open and read the block 12 of the file "/home/study/cs130.txt"

Block12 对应的为一级 indirect pointer，所以在找到 file 的 inode 时我们需要额外的一级操作从 indirect pointer 来找到 direct pointer 才能读取 block12



11. What data makes sense for an inode to store (multiple choice): A. The size of a file; B. User number & group number; C. Access mode (r/w/x permissions); D. File descriptor.

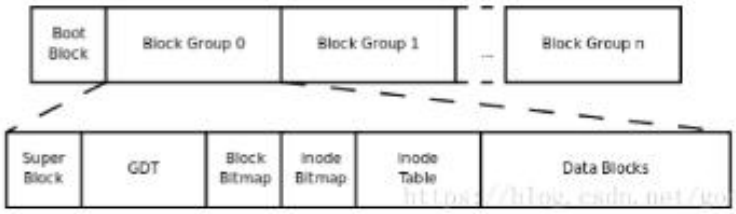
->>>>>>>ABC

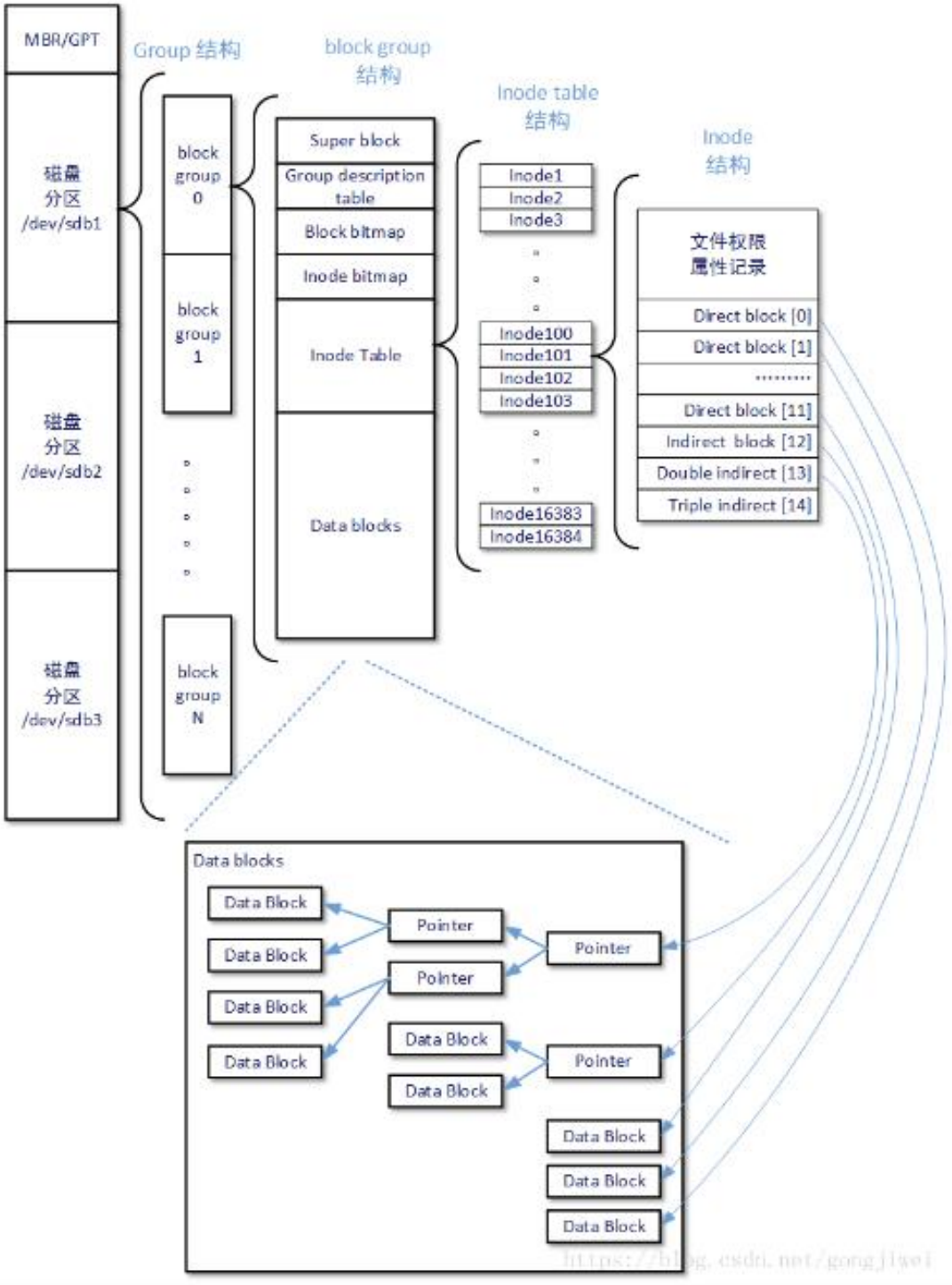
12. In a well-known Unix-like operating system, Linux, one flag of the open() system call is O_APPEND, which specifies that the file is opened in append mode i.e., the subsequent write() system calls take place at the current end of the file. Explain how the data listed in 11.can be accessed and updated when you open and append to a file.

->>>

When opening the file, before each write() the file location (in the file descriptor) where the write takes place is read from the size field in the inode. After each write() the file location and the size are updated with the offset of the new end of the file. Access permissions are checked when the file is opened (not on each write of the file).

ext2文件系统


Boot Block
Block Group 0
Block Group 1
...
Block Group n
Super Block
GDT
Block Bitmap
Inode Bitmap
Inode Table
Data Blocks


MBR/GPT
磁盘分区 /dev/sdb1
磁盘分区 /dev/sdb2
磁盘分区 /dev/sdb3
Group 结构
block group 0
block group 1
block group N
block group 结构
Super block
Group description table
Block bitmap
Inode bitmap
Inode Table
Data blocks
Inode table 结构
Inode1
Inode2
Inode3
Inode100
Inode101
Inode102
Inode103
Inode16383
Inode16384
Inode 结构
文件权限
属性记录
Direct block [0]
Direct block [1]
Direct block [11]
Indirect block [12]
Double indirect [13]
Triple indirect [14]
Data blocks
Data Block
Pointer
Pointer
Pointer
Pointer
Data Block
Data Block
Data Block
Data Block
Data Block
Data Block
Data Block

12. Kernel booting is a complex procedure and you may find some words appear frequently when diving into the booting process. Please explain briefly what are BIOS, UEFI, GPT and MBR? What's the use of GRUB? And what's the relationship between them? Please explain the questions above briefly.

->BIOS (basic input/output system) is the program a computer's microprocessor uses to start the computer system after it is powered on.

->The Unified Extensible Firmware Interface (UEFI) is a publicly available specification that defines a software interface between an operating system and platform firmware.

->Both UEFI and BIOS are low-level software that starts when you boot your PC before booting your operating system, but UEFI is a more modern solution, supporting larger hard drives, faster boot times, more security features such as graphics and mouse cursors.

->The GUID Partition Table (GPT) is a standard for the layout of partition tables of a physical computer storage device.

->A master boot record (MBR) is a special type of boot sector at the very beginning of partitioned computer mass storage devices.

->GPT is a newer partitioning standard than MBR and doesn't have as many limitations.

->BIOS+MBR: Most traditional booting methods (disk < 2TB).

->BIOS+GPT: Can't booting operating systems but can be used as storage devices.

->UEFI+MBR: Can achieve the same goal as BIOS+MBR.

->UEFI+GPT: Can booting operating system on large disk (> 2TB). The os must be 64bits.

->GRUB (Grand Unified Bootloader) is a complete program for loading and managing the boot process. The bootloader transfers the control to the operating system kernel.