OS Project 3
Demand Paging

Advisor: Tei-Wei Kuo
Speaker: Che-Wei Tsao

Outline

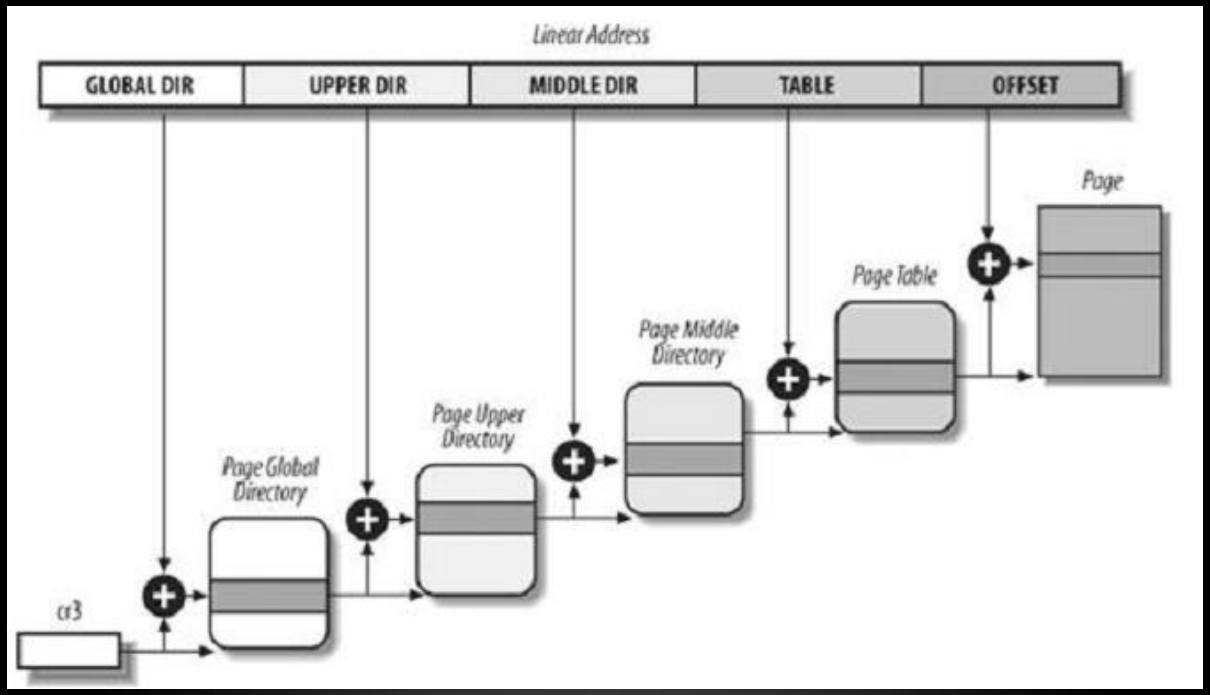
- Introduction
- Project Requirements
- Submission Rules
- References

Memory Management in Linux kernel

- Page frame management
  - memory architecture, page replacement strategy, … etc
- Kernel object management
  - Slab, buddy, … etc
- Process address space management
  - Page table handling, memory region , … etc

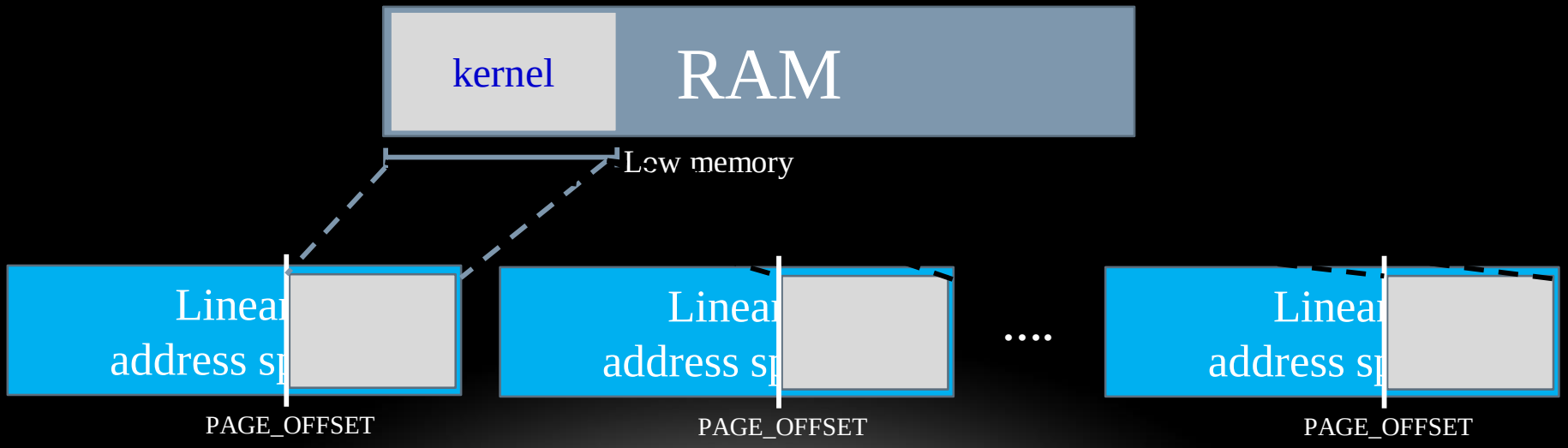
Paging in Linux

- Page table structure
  - A common paging model that supports both 32-bit and 64-bit architecture



Paging in Linux

- Process page table
  - The macro PAGE_OFFSET = 0xC000 0000
  - This is the offset in the linear address space of a process where the kernel lives



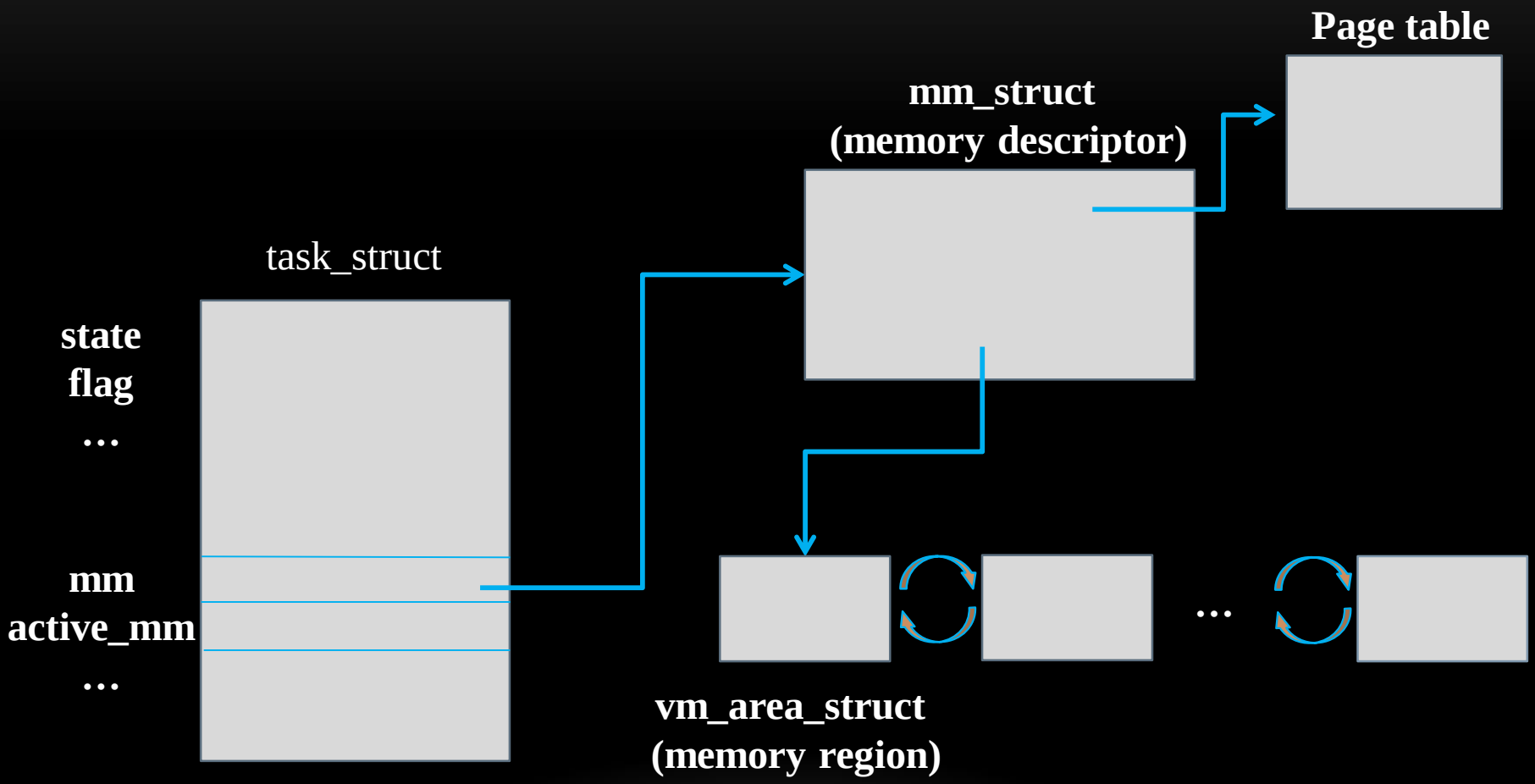
Memory Request

- Requested by kernel
  - Kernel is the highest component of the OS
  - Kernel trusts itself
- Requested by user processes
  - The requests are considered non-urgent
  - User program cannot be trusted
    - Error handling

Process Address Space

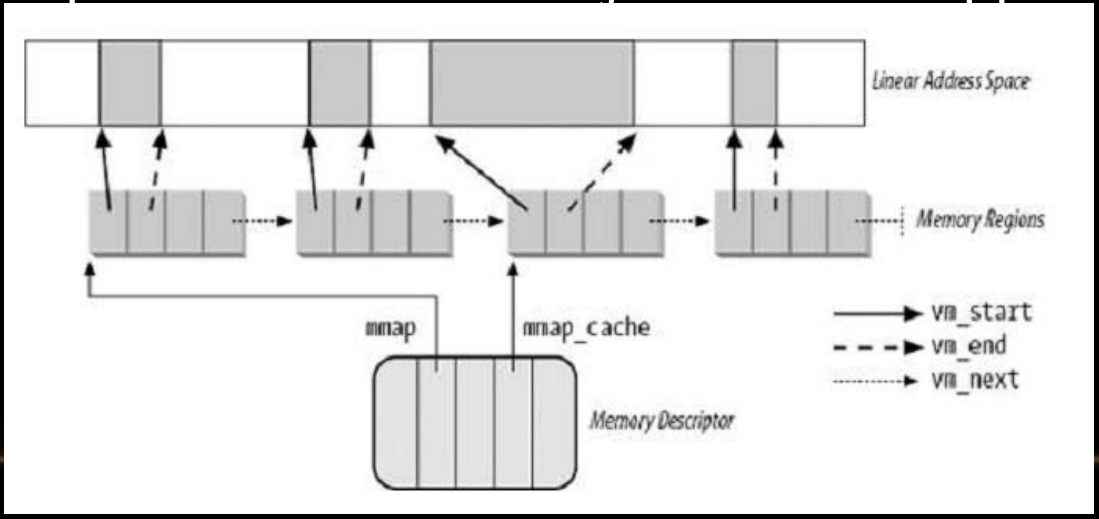
- Each process has its own linear addresses
- The full address space of a process is rarely used
- Instead of getting page frames directly, it gets the right to use a new range of linear addresses (Memory Region)

Process Address Space



The Data Structure of Memory Region

- vm_start – first linear address inside the region
- vm_end – first linear address after the region
- vm_flags – the access rights of the region
- vm_ops (vm_operations_struct) – pointer to the methods of the region
- vm_file – pointer to the file object of the mapped file, if any



Memory Region Operations

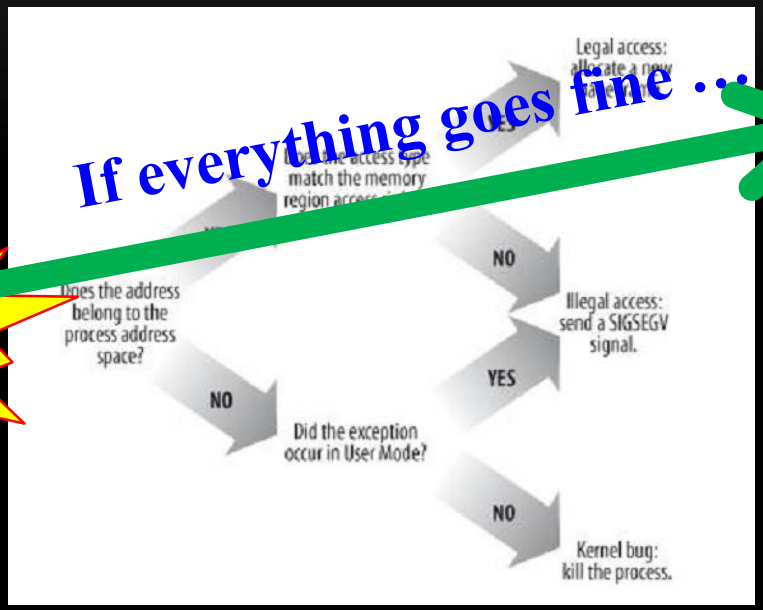
- vm_operations_struct //include/linux/mm.h
  - void (*open)(struct vm_area_struct* area)
  - void (*close)(struct vm_area_struct* area)
  - int (*fault)(struct vm_area_struct* area, struct vm_fault* vmf);

- File-backed memory regions will use a generic memory region operation
  - generic_file_vm_ops //mm/filemap.c
    - .fault = filemap_fault

```
struct vm_fault {
        unsigned int flags;
        pgoff_t pgoff;
        void __user *virtual_address;

        struct page *page;
```

Page Fault Handling Overview



Memory Region Handler

File-backed

Non-file-backed

Readahead algorithm

Page cache

disk

Anonymous page handling

Demand paging

- On memory efficiency perspective
  - Page contents that will not be accessed should not be loaded into memory
  - Thus, it favors small page loading on page fault
    - An extreme case: pure demand paging
- On runtime performance perspective
  - Disk I/O access is very time-consuming
  - Thus, it favors large page loading on page fault

Memory efficiency v.s. runtime performance

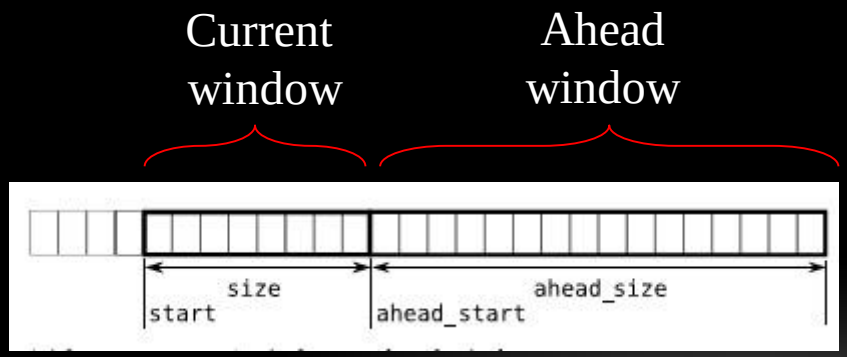
Readahead scheme (kernel 2.6)

- A widely deployed technique to bridge the huge gap between disk access and the memory usage of applications
  - Disk drives suffers from seek latencies and are better utilized by large accesses
  - Applications tend to do lots of tiny sequential reads
- 3 major benefits
  - I/O delays are effectively hidden from the applications
  - Disks are better utilized with the large prefetching requests
  - It helps to amortize processing overheads in the I/O path

Readahead scheme (kernel 2.6)

- On **sequential access**
  - Ahead window size = 2* Current window
    (the size growing is stopped when reaching max size)
  - Whenever a request is crossing ahead window,
    it becomes current window and new request I/O
    is triggered to make new ahead window
- On **random access**
  - Ignore the ahead window
  - The size of the next ahead window is shrinked

- ***Sequential***:
  file pointer = 0 or
  the page is the next page
  of last page fault



- Cache hit – minor page fault
- Cache miss – major page fault

Testing Flow

1. Add additional kernel parameter in boot loader
   - loglevel=2
2. Instrument message in mm/filemap.c/filemap_fault()
   - printk(KERN_CRIT, “%s, %X\n”, current->comm, vmf->virtual_address);
3. Clear page cache
   - Echo 1 | sudo tee /proc/sys/vm/drop_caches
4. Run test.c process
5. Collect syslog and program output

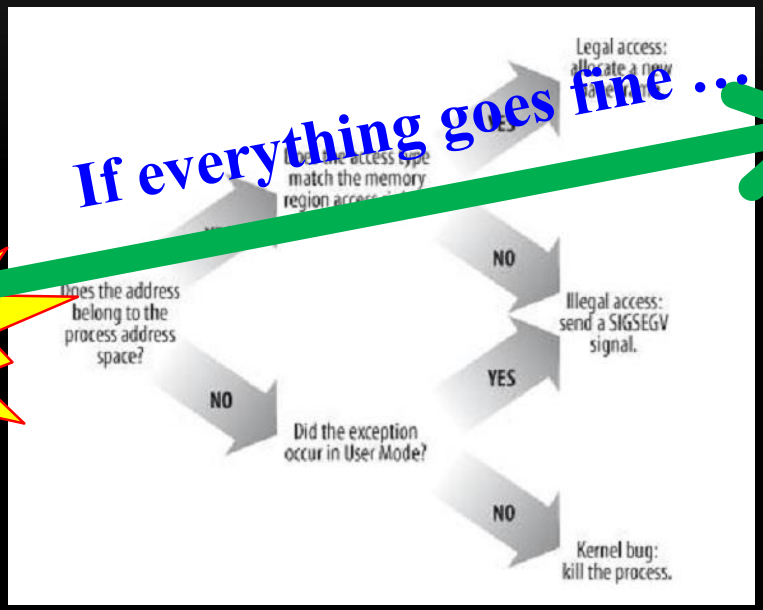
Scoring of Project 3

- Revise mm/filemap.c/filemap_fault() for pure demand paging (60%)
- Report (40%)
  - At most 4 pages
  - Trace code
  - Compare pure demand paging with the readahead algorithm by a case study
    - A test program is given

Test Program

- Input.log
  - A random generated file
  - 128 MB
- test.c test.h
  - Map input.log into process address space
  - Read the first integer of a page specified by an index array
- Syslog.sh
  - Write message to system log

Page Fault Handling Overview



Page Fault

Memory Region Handler

File-backed

Non-file-backed

Readahead algorithm

Page cache

disk

Anonymous page handling

Scoring of Project 3 (cont)

- Bonus (at most 20%)
  - Implement your own readahead algorithm (15%)
    - Either one of the following conditions must be satisfied
      - # of page faults < # of default page faults
      - # of RSS (Resident Set Size) < # of default RSS
  - Report (5%)
    - Additional 2 pages at most

Submission Rules

- Project deadline: 2015/06/17 23:59
  - Delayed submissions yield severe point deduction
- Upload your team project to the FTP site.
  - FTP server: 140.112.28.132 (SFTP)
  - 請用anonymous (匿名)方式登入
- The team project should
  - Contain the whole "**linux3.2.54/**" directory
  - Contain your page fault syslog
  - Contain your test program syslog
  - Contain your report (PDF, within 6 pages)
  - Be packed as one file named "OSPJ3_Group##.tar.xz"
- **DO NOT COPY THE HOMEWORK**

References

- Understanding the Linux Virtual Memory Manager
- Understanding the Linux kernel, 3rd
- LinuxMM, http://linux-mm.org/
- Linux Cross Reference, http://lxr.free-electrons.com/
- Kernel Parameters, http://lxr.free-electrons.com/source/Documentation/kernel-parameters.txt
- Debugging by printing, http://elinux.org/Debugging_by_printing

Contact TAs

- If you have any problem about the projects, you can contact TAs by the following ways:
- Facebook: NTU OS2015 Spring Group

  - https://www.facebook.com/groups/920624997989865/
- E-mail:
  - Che-Wei Tsao: d02944011@csie.ntu.edu.tw