OS Project 3
Readahead Algorithm

Advisor: Tei-Wei Kuo

Speaker: Chih-Hsuan Yen

Outline

- Introduction
- Project Requirements
- Submission Rules
- References

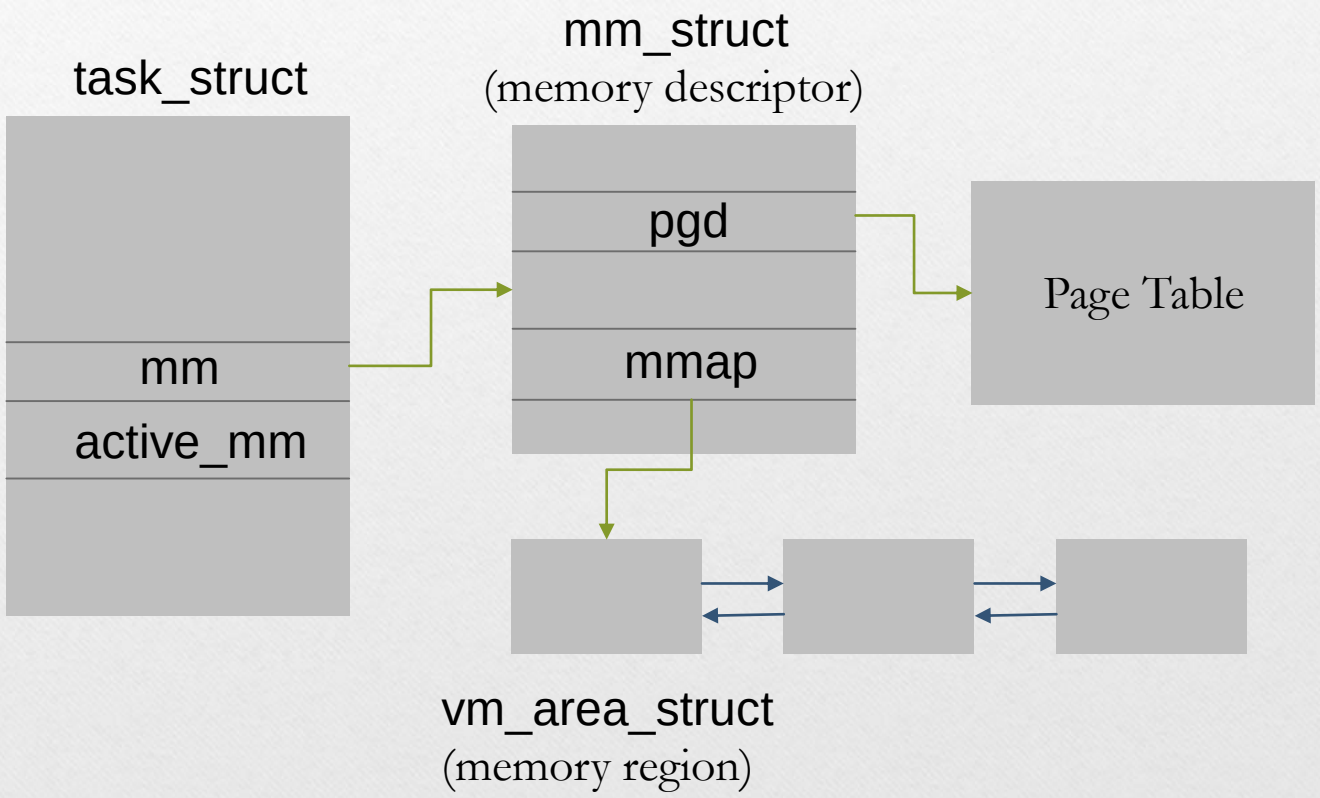
Memory Management in Linux

- Page frame management
  - Memory architecture, page replacement policy, … etc.
- Kernel object management
  - Slab allocator, buddy system, … etc.
- Process address space management
  - Page table handling, memory region, … etc.

Memory request

- Requested by kernel – no point to defer it
  - Kernel is the highest component of the OS
  - Kernel trusts itself
- Requested by user processes – deferred allocation
  - Instead of getting page frames directly, it gets the right to use a new range of linear addresses (Memory Region)
  - The requests are considered non-urgent
  - User program cannot be trusted – error handling

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

Process Memory Regions

sudo cat /proc/1/maps

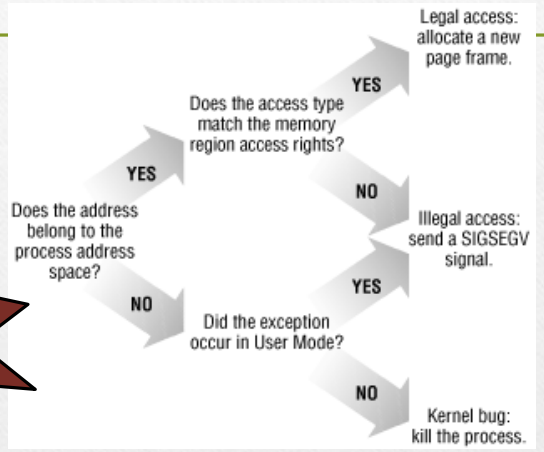
```
b76bf000-b76c7000 r-xp 000                                -gnu/libnih-dbus.so.1.0.0
b76c7000-b76c8000 r--p 00007000 08:01 132115     /lib/i386-linux-gnu/libnih-dbus.so.1.0.0
b76c8000-b76c9000 rw-p 00008000 08:01 132115     /lib/i386-linux-gnu/libnih-dbus.so.1.0.0
b76c9000-b76e0000 r-xp 00000000 08:01 132117     /lib/i386-linux-gnu/libnih.so.1.0.0
b76e0000-b76e1000 r--p 00016000 08:01 132117     /lib/i386-linux-gnu/libnih.so.1.0.0
b76e1000-b76e2000 rw-p 00017000 08:01 132117     /lib/i386-linux-gnu/libnih.so.1.0.0
b76f2000-b76f4000 rw-p 00000000 00:00 0
b76f4000-b76f5000 r-xp 00000000 00:00 0          [vdso]
b76f5000-b7715000 r-xp 00000000 08:01 132232     /lib/i386-linux-gnu/ld-2.15.so
b7715000-b7716000 r--p 0001f000 08:01 132232     /lib/i386-linux-gnu/ld-2.15.so
b7716000-b7717000 rw-p 00020000 08:01 132232     /lib/i386-linux-gnu/ld-2.15.so
b7717000-b7745000 r-xp 00000000 08:01 32658      /sbin/init
b7745000-b7746000 r--p 0002e000 08:01 32658      /sbin/init
b7746000-b7747000 rw-p 0002f000 08:01 32658      /sbin/init
b8948000-b89cc000 rw-p 00000000 00:00 0          [heap]
bfd5d000-bfd72000 rw-p 00000000 00:00 0          [stack]
```

vm_start vm_end vm_flags vm_file

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 //mm/filemap.c
  - vma->vm_ops = generic_file_vm_ops
  - .fault = filemap_fault

Page Fault Handling Overview



Page Fault

Memory region fault handler

Non-file-backed

File-backed

Anonymous Page Handling

Page Cache

Readahead

Disk

Readahead scheme (2.4.13+)

- Applications tend to do lots of tiny sequential reads
  - 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
- 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

How much to read?

- 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

Requirements of Project 3

- Revise mm/filemap.c/filemap_fault() for pure demand paging (60%)
- Report (40%)
  - At most 4 pages
  - Trace code: mmap() and filemap_fault() (15%)
  - Compare pure demand paging with the readahead algorithm by the test program (25%)

Testing Flow

- Add additional kernel parameter in boot loader
  - Modify /etc/default/grub
  - loglevel=2
- Instrument message in **mm/filemap.c, filemap_fault()**
  - printk(KERN_CRIT, "%s, %X\n", current->comm, vmf->virtual_address);
- Clear page cache
  - echo 3 | sudo tee /proc/sys/vm/drop_caches
- Run test.c process
- Collect syslog (dmesg) and program output

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 (dmesg)

Bonus of Project 3

- Implement your own readahead policy (15%)
- Either one of the following conditions must be satisfied
  - # of major page faults <# of default major page faults
  - # of RSS (Resident Set Size) < 1.1 * # of default RSS
- Report (5%)
  - Additional 2 pages at most
  - Implementation, discussion & experiments

Submission Rules

- Project deadline: 2016/06/15 (Wed.) 23:59
  - Delayed submissions yield severe point deduction
- Upload your team project to the FTP site.
  - FTP server: 140.112.28.132:5566 (SFTP)
  - Account/password: os2016/ktw2016os
- The team project should contain
  - Any modified files
  - Your report (PDF, within 6 pages)
- Packed as "OSPJ3_Group##.tar.gz"

```
OSPJ3_Group##/
    Report.pdf
    Baseline/
        xxx.c
    Bonus/
        yyy.c
```

References

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

More References

- brk() & sbrk()
  - http://man7.org/linux/man-pages/man2/sbrk.2.html
- Virtual Memory Areas
  - http://www.makelinux.net/books/lkd2/ch14lev1sec2
- Page Tables in Linux kernel
  - https://www.kernel.org/doc/gorman/html/understand/understand006.html
- Linux Cross Reference http://lxr.free-electrons.com/

Contact TAs

- If you have any problem about the projects, you can contact TAs by the following ways:
- Facebook: NTU OS2016 Spring Group
  - https://www.facebook.com/groups/1683988081869980/
- E-Mail
  - Chih-Hsuan Yen: r04922036@ntu.edu.tw