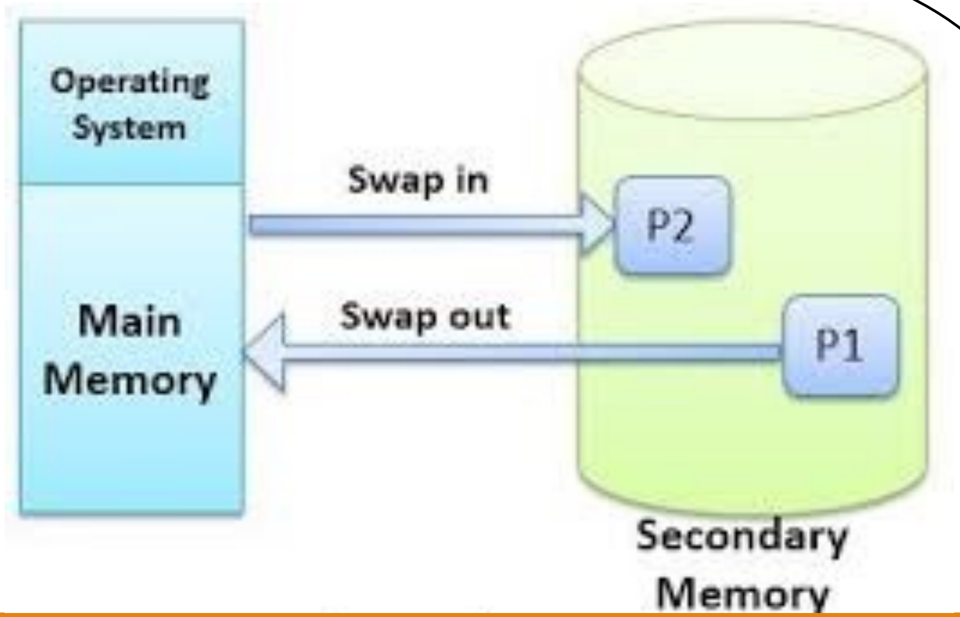
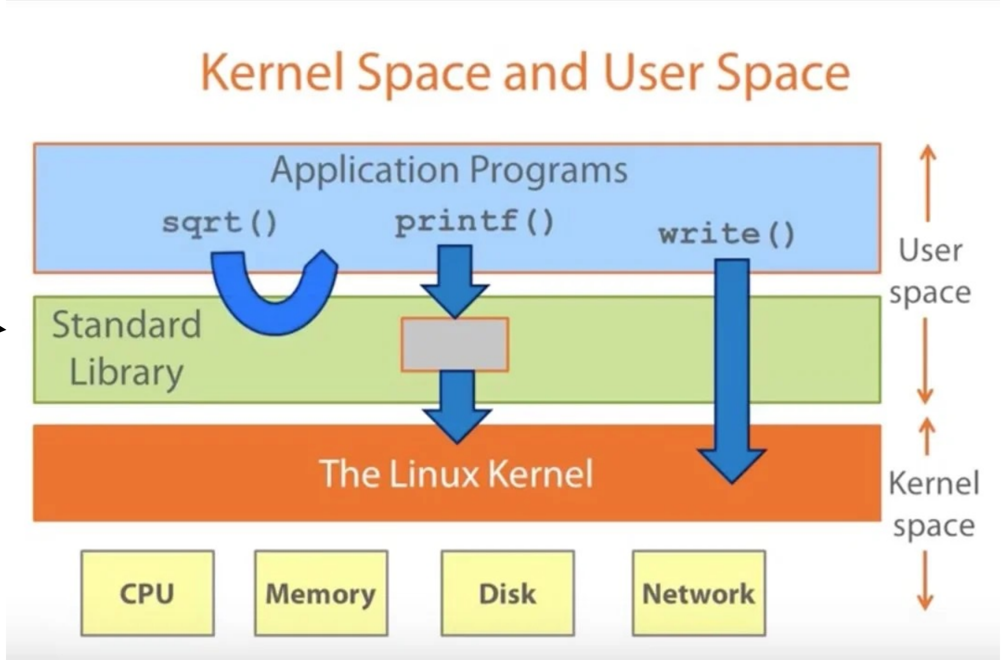
CS2106 Lab 4

LAB 9 & 13

Overview

- Swap: Extends memory by moving less- used pages to secondary storage
- Normally, kernel implements swap
- Objective: Implement user-space swap library





Segmentation Fault

- Occurs when process performs invalid memory access → process terminates [SIGSEGV]
- Can install a signal handler for SIGSEGV
  1. Return to instruction that caused invalid memory access
  2. Swap memory in user-space
  3. Retry the instruction
- Essentially, a page-fault handler

Lab 4 Implementation Guide

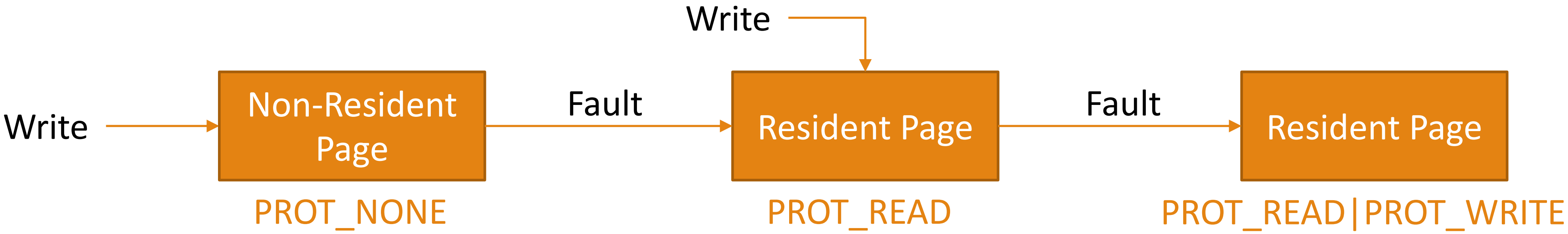
- userswap.c – only modify this file
- userswap.h
- workload_*.c – feel free to write additional test cases
- Make file – add your own workload to the Makefile

Ex0 – Allocating Memory

- Implement `userswap_alloc` to allocate requested amount of memory
  1. Size rounded up to multiple of page size (4096 bytes)
  2. Memory initially non-resident (PROT_NONE)
  3. Return pointer to start of memory
  4. Keep track of size of memory allocation using some data structure (for freeing)
  5. Use mmap
- Implement userswap_free to free entire memory allocated starting at a provided address
  1. Use munmap
- Implement SIGSEGV signal handler that calls the page fault handler
  1. Use sigaction
- Implement the page fault handler
  1. Use mprotect to make the page resident

Ex1 – Extend Handlers

- Extend the SIGSEGV handler to verify faulting memory address is in controlled memory region
  1. If in controlled memory region, continue page fault handler (make page resident etc.)
  2. Else, remove handler, reset the action taken for SIGSEGV and return immediately



- Extend page fault handler
  1. Upon write, accessed page is made PROT_READ | PROT_WRITE
  2. Track the state of each page using some data structures:
     a. 4-level page table
     b. List of lists of pages

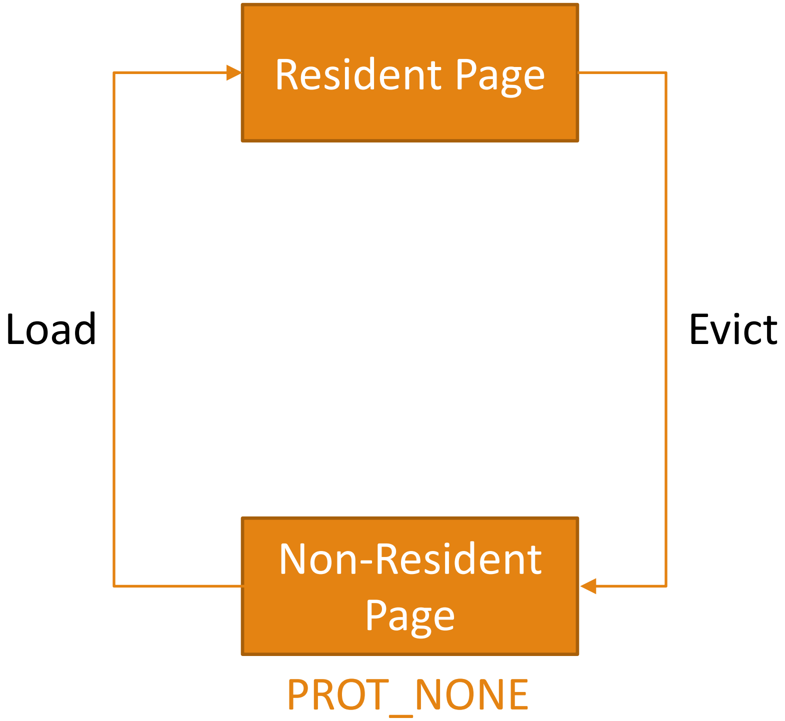
- Extend userswap_free so that it cleans up the data structures

Ex2 – Page Eviction

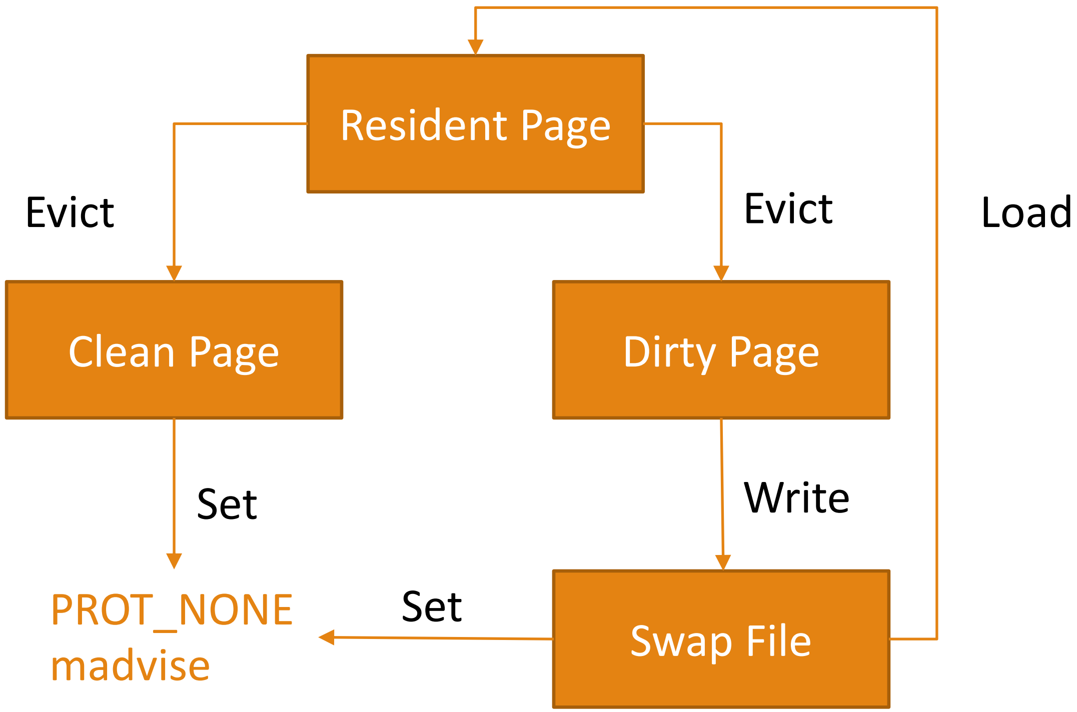
- Extend page fault handler
  1. Evict page if LORM will be exceeded when making a page resident
  2. Use a data structure to keep track of which resident to evict first (FIFO eviction algorithm)
- Extend userswap_free to update the data structure when allocation is freed
- Implement userswap_set_size to set LORM to a particular size
  1. Size must be a multiple of page size (round up if not)

Ex3 – Implement Swap

Ex2



Ex3



Ex3 – Implement Swap

- Extend page fault handler

1. Page is evicted → if dirty, new contents written to swap (do not use buffering)
2. Page is freed by calling madvise and mprotect
3. Non-resident page loaded from swap file if swap file exists (else it should be a freshly allocated page)
4. One swap file for the entire process
5. Size of swap file cannot exceed total memory in controlled region
6. Find place within the swap file to evict it to and keep track of the location for loading
7. Keep track of locations in the swap file which have been freed for reuse

Ex4 – Implement userswap_map

- Implement userswap_map
    1. Similar to uswerswap_alloc, except that the region will be backed by the file

- Extend page fault handler such that upon read to non-resident pages allocated by userswap_map, page is filled with contents of the file at the location

- When evicting page allocated by userswap_map, just discard the page without storing the contents

Ex5 – Extend handlers

- Extend page fault handler
  1. When a page allocated by userswap_map is dirty and evicted, its contents are written back to the backing file

- Extend userswap_free so that dirty pages are written back to backing file (do not close the FD)

Ex6 – Bonus

- Synchronise concurrent access of controlled memory regions
- Graded based on performance and quality
- Make sure it can work for Ex1 – Ex5
- Include a writeup of your idea

Remarks

- Page size is 4096 bytes
- Make sure to verify success/failure of every syscall
- Implementation should not require more than 128 bytes of overhead per page of memory
- Do not use mmap syscall to map file into memory (use -1 for fd argument)
- Ensure reasonable performance (< 10 seconds)

Final Remarks

- Try to do it exercise by exercise to ensure you implement every function properly
- Read the requirements carefully (and repeatedly)
- Read the FAQ if you are lost
- Deadline: 12/11, 2pm
- Wait for update on lesson next week