Kennesaw State University

CSE 3502 Operating Systems - Spring 2020

Project 3 - Memory Management

Instructor: Kun Suo
Points Possible: 100

Assignments

Assignment 1: (50 points)

This project reviews the key memory management concepts we studied in class: virtual memory, page tables and the page cache (e.g., TLB). Virtual memory often refers to the process address space assigned to user-space processes. Such virtual addresses need to be translated into physical addresses with the help of page tables. The page cache is used to temporarily store recently read or written pages in memory.

Write a system call *sys_getmemInfo (*e.g., follow the steps in Project 1) to report statistics of a process's virtual address space. You can use *printk* in your system call and collect the information in *dmesg* command. The system call should take a process ID as input and outputs the following information about the process:

1. Each virtual memory area's access permissions.
2. The names of files mapped to these virtual memory areas.
3. The total size of the process's virtual address space.

Using two user-level programs (assignment1-user1.c, assignment1-user2.c) to test your system call. One test program just calls the new system call and report the calling process's statistics. The other test program should create multiple threads (e.g., using Pthread in project 2) and report information about individual threads. The purpose of the second test program is to study if threads share the same address space.

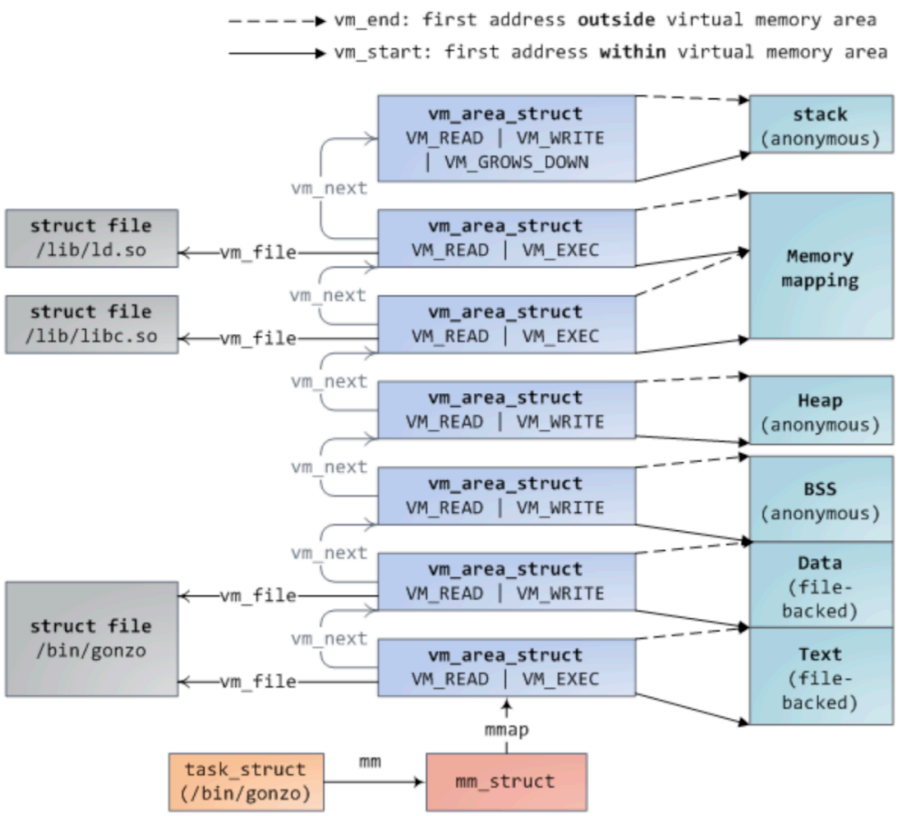
Hints:

The Linux kernel uses the memory descriptor data structure to represent a process's address space. The memory descriptor struct *mm_struct* is defined in *<linux/mm_types.h>* and is included in a process's data structure *task_struct*.

In *mm_struct*, the *mmap* field points to a linked list of struct *vm_area_struct*. The *vm_area_struct* describes a single memory area over a contiguous interval in an address

space. The *vm_start* and *vm_end* point are the start and end address of individual *vm_area_struct*. All the virtual memory areas together form a process's virtual address space. To calculate the size of a virtual address space, one only needs to sum the sizes of individual virtual memory areas (VMA). You can verify your result using command *pmap*. Variable *vm_page_prot* in struct *vm_area_struct* represents the access permission of this VMA. Variable *vm_file* in struct *vm_area_struct* represents the file this VMA is mapped to and the file name can be accessed through *vm_file->f_path.dentry->d_name.name*.

Data structure of virtual memory of one process:



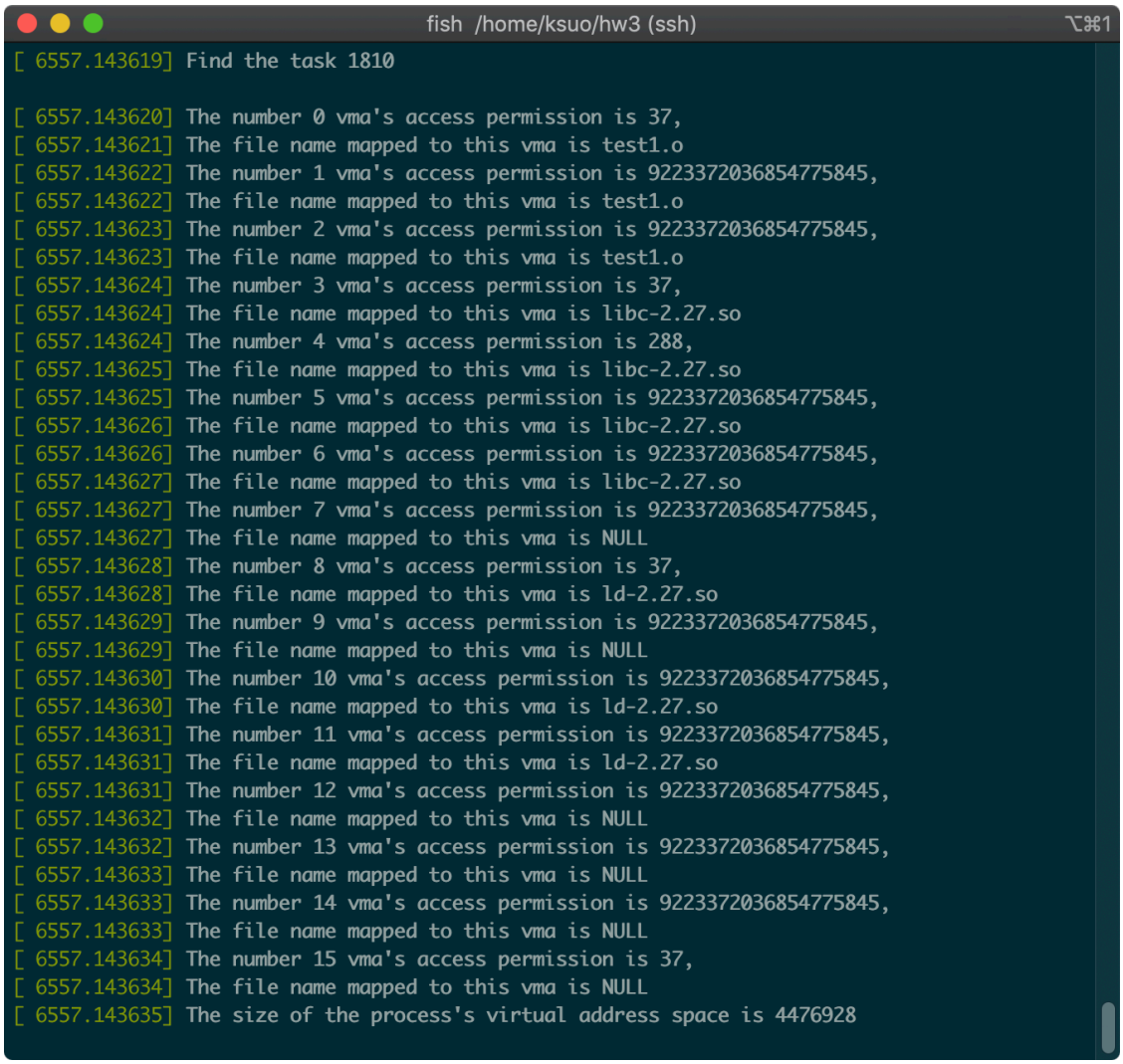
Submission requirements:

(1) Create one folder named **kernel** and put your modification (file diff1.txt, diff2.txt, …) of the Linux kernel into this folder. Please use diff command to highlight your modification:
$ diff -u original_file.c modified_file.c. > diff.txt

(2) Create another folder named **user** and put your two user-level test programs inside.

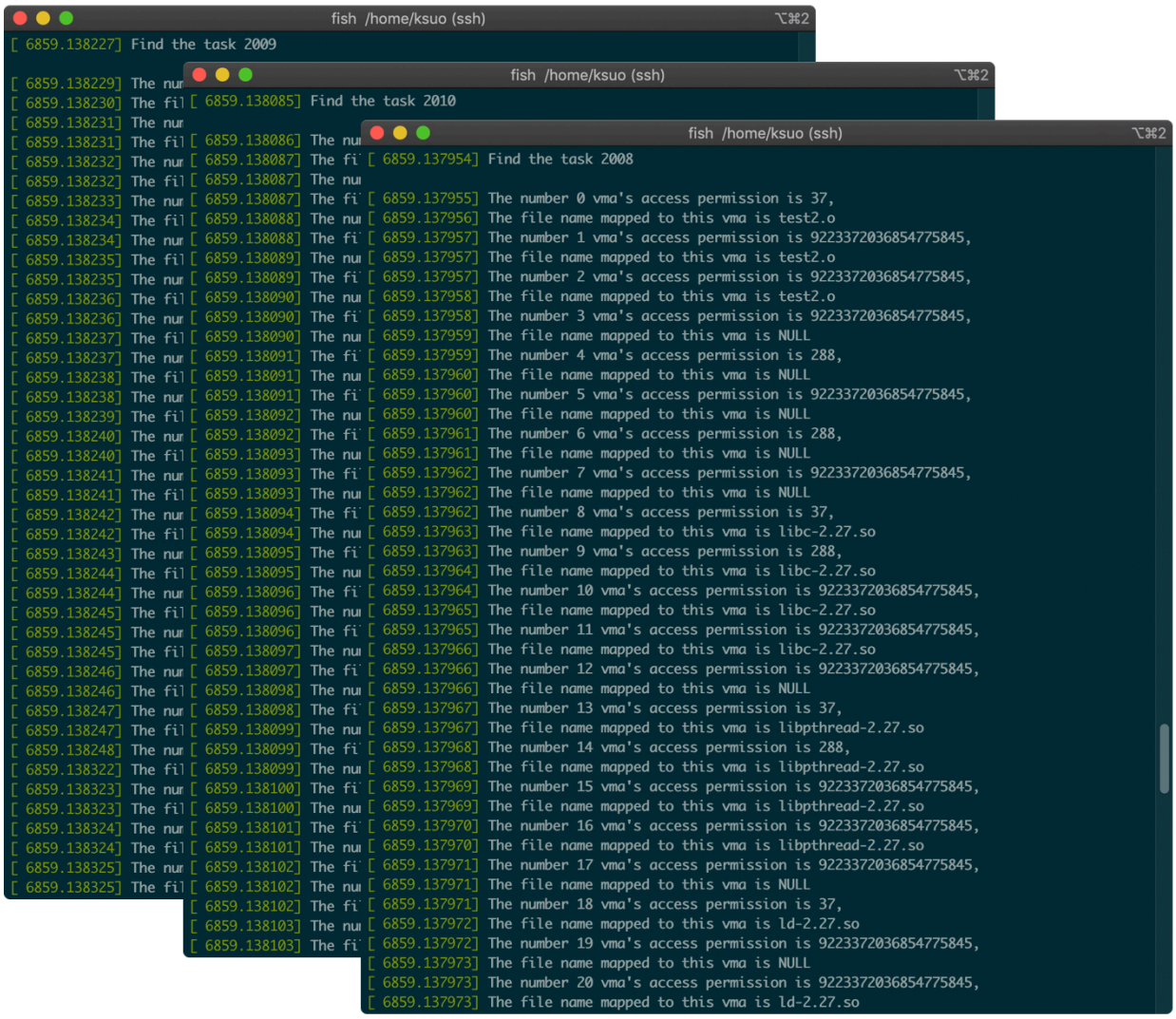
(3) Put kernel folder and user folder into folder **Assignment-1**.

Examples of output:
(1) System call output of a single process:



Examples of output:
(2) System call output of multiple threads:

User level test program creates 4 threads and each thread executes *getmeminfo* system call. Here is the dmesg output from the operating system: print out the virtual memory information of the created pthreads.



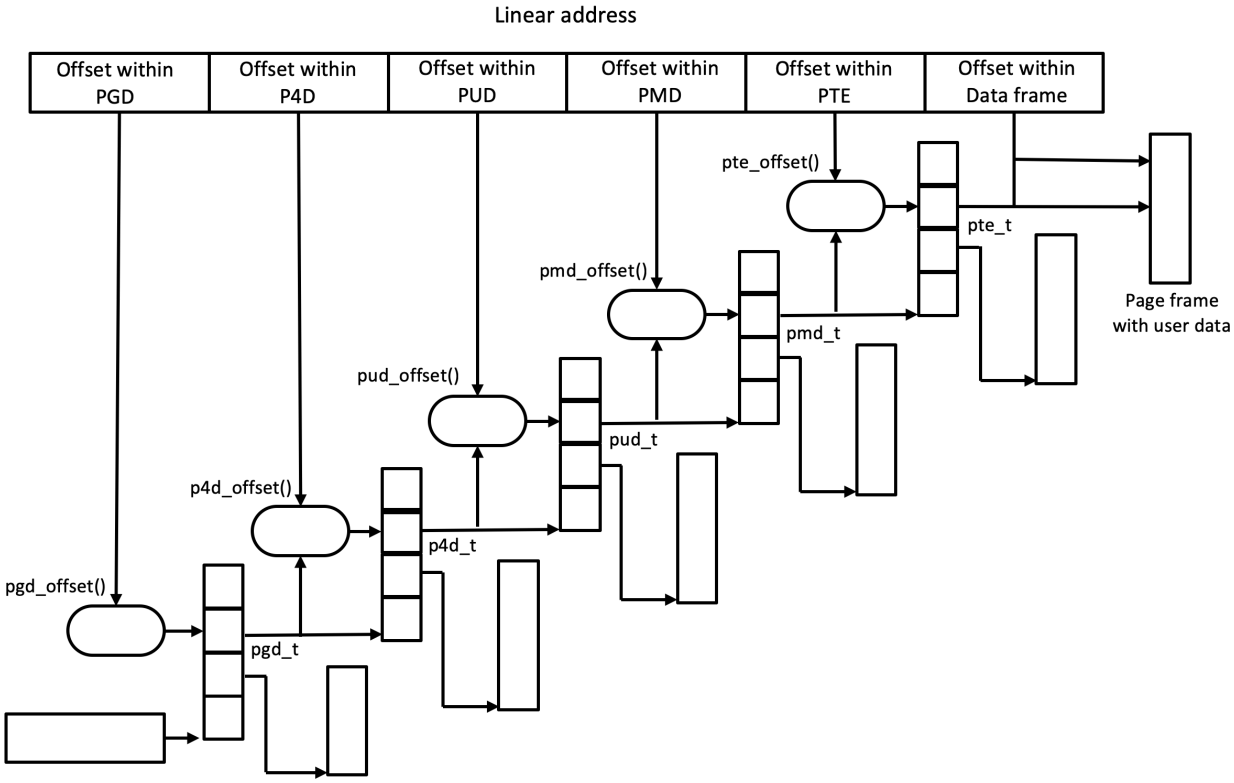
Assignment 2: (50 pts)

Given the above virtual memory areas used by a process, write a system call *sys_getPageInfo* to report the current status of specific addresses in these virtual memory areas. The system call takes the pid of a process as input and outputs the following information of start address *vm_start* in each *vm_area_struct* that the process has:

1. If the data in this address is in memory or on disk.
2. If the page which this address belongs to has been referenced or not.
3. If the page which this address belongs to is dirty or not.

Using user-level programs (assignment2-user1.c) to test your system call.

Data structure:



Hints:

The page descriptor (data struct *pgd_t*) contains information about the page. You need to figure out how to obtain a reference to the page descriptor given a virtual address and read to information from the page descriptor. To get current status of specific addresses, you need to test the corresponding flag bit of the address's page table entry (data struct *pte_t*). Note that Linux uses multi-level page tables, you might need multiple steps to reach the page table entry of a given virtual address.

Table: Page Table Entry Status Bits

Bit	Function
_PAGE_PRESENT	Page is resident in memory and not swapped out.
_PAGE_ACCESSED	Set if the page is referenced.
_PAGE_DIRTY	Set if the page is written to.

The function *pte_present* is used for judging a page is in memory or disk. If the return value is 1, the page is in the memory, otherwise it is in the disk. The function *pte_young* is used for judging a page is referenced or not. If the return value is 1, the page has been referenced, otherwise it has not been referenced yet. The function *pte_dirty* is used for judging a page is dirty or not. If the return value is 1, the page is dirty, otherwise it is not dirty.

Submission requirements:

(1) Create one folder named **kernel** and put your modification (file diff1.txt, diff2.txt, …) of the Linux kernel into this folder. Please use diff command to highlight your modification:
$ diff -u original_file.c modified_file.c. > diff.txt

(2) Create another folder named **user** and put your user-level test programs inside.

(3) Create a folder to put the screenshot of your program output.

(4) Put ***kernel folder, user folder and screenshot folder*** into folder **Assignment-2**.

Zip all the files and folders together into one zip file and name it as CS3502_[your D2L user name], e.g., **CS3502_mahmed29.zip**, and upload the file onto D2L.

Examples of output in assignment 2: