Advanced Operating Systems
Report

Point total: 20
Point values are unique to each project assignment.

Note: One page (approximately 600 words). Discuss any non-trivial routines implemented to complete the task and high level motivations behind your design decisions. Discussions on approaches taken to debug your solution on your own are also appreciated.

Part 1 Report - 10pts
Part 2 Report - 10pts

Breakdown of the 10pts:

1. Approach and methods - 5pts total. 0pts if what you wrote about in your report does not reflect the code you submitted. You must provided analysis or reasoning for why a change or code segment was required to get full points.

2. Calculations - 3pts testing results and any extra test cases you wrote yourself

3. Discussion - 2pts observations from test results, why do you think your solution worked/ didn't work, key takeaways.

Solution:

Part A - NULL pages

Approach and methods -

I implemented the numproctect and protect in vm.c which is responsible for memory allocation. First, I check the integrity of the parameters. And then, I design the logic by referencing others' functions in vm.c. I design the for-loop to read the page range staring at addr and of len pages and get the address of the page table entries in the page by calling walkpgdir function. I also use PGROUNDDOWN to convert an address in the middle of a page to an address at the beginning of the page. And then the program would check the permission in the page table. And then I change the protection bits to be read only if it is not in use status. In numproect() function, I use for-loop to iterate through the pages in the range starting at addr to len pages, and get the address in the page table by by calling walkpgdir function in the loop. And then, I check the permission in the page table, and then I change the protection bits back to both readable and writable.

Calculations - I passed all tests 1 - 9. I tested the protection function in the test10 created by using test8 template and adding munprotect before executing fork function check.

Discussion -

In this project, I experienced hand-on a simple two-level page table as discussed in class. In the testing case, some testing is used to validate parameters such as invalid page range address or unaligned page address. In the walkpddir, I understand how to translate from a virtual address to a physical address or a physical address to a virtual address and how to check the permission of the page address. In the test8, when a protection fault occurs, the page address is marked as writable. Hence, in the test10, I put munprotect before executing fork to trigger protection fault and observe whether my numprotect work or not.

Part B - Kernel Threads

Approach and methods -

I implemented the thread-join and thread-clone function in proc.c which is used for creating a new kernel thread and waiting child thread. In the clone function, I reference the fork function which is used to create a new process and return a child's PID. Specifically, tstate is used for judging whether the thread in current running process is in running status or not, and tstack is set to the thread stack for this process. In the join function, I inspect the wait function because the process's children have to wait for a child thread to complete in the join system and returns the PID of the waited-for child. And also, the function would free the user address space if the process's status is changed to ZOMBIE which records a child's demise. And then the location of the child's user stack arguments from the tstack is copied into the argument stack.

Calculations -

I passed all tests 1 - 8. I copied test 8 to test9 and then increased the number of transactions and added an extra job to execute thread.

Discussion -

I think my solution works well. In the clone function, I get the current process and copy its information to the thread stack since all threads in a process share the heap, data, and code, there is no need to involve the kernel involved. In the testing case9, the program would ensure that only one thread can access resources as variable value at a time by dining lock and release. it is possible that three threads will lock each other, resulting in very slow execution. However, if I change the order of execution and separately run three threads and join together, it seems to work rather than running for a long time.

It is possible that they are not running at the same time. So, all threads might have to be declared before joining together.