Assignment 3

Using the base code in Github and your solution from the second assignment, you will need to build upon what you submitted for the Simple OS File System assignment to complete this assignment. For this assignment, you are only to make changes to frame-allocator.cpp and vm.cpp. I have outlined the functions in vm.cpp and frame-allocator.cpp that you will need to implement below.

When acquiring or releasing locks in these two files, be sure to use KERNEL_OWNED as the argument for currentPid of acquireLock() and releaseLock().

frame-allocator.cpp:

1. Implement createPageFrameMap() function. This function creates the page map that will show which page frames are available and, if used, which PID owns that page frame. There will be one byte allocated for each page of the 4 MB of RAM.
    a. Frame zero and KEYBOARD_BUFFER should be KERNEL_OWNED
    b. USER_PID_INFO should be KERNEL_OWNED
    c. GDT_LOC should be KERNEL_OWNED
    d. VIDEO_AND_BIOS_RESERVED_START to VIDEO_AND_BIOS_RESERVED_END should be KERNEL_OWNED
    e. KERNEL_BASE to KERNEL_LIMIT should be KERNEL_OWNED
    f. All else for the size of RAM should be PAGEFRAME_AVAILABLE
2. Implement allocateFrame() function. This function allocates a frame of memory to a PID.
    a. First acquire a lock on PAGEFRAME_MAP_BASE
    b. Find first available page frame and allocate to the PID
    c. Release lock
    d. Return frameNumber
3. Implement freeAllFrames() function. This function releases all page frames for a given PID.
    a. First acquire a lock on PAGEFRAME_MAP_BASE
    b. Loop through and free all frames (marking them as PAGEFRAME_AVAILABLE) for a given PID
    c. Release lock

vm.cpp:

1. Implement initializePageTables() function. This function creates the page directory and single page table for a process based on the passed in PID value.
    a. Compute the position of tables based on PID. PID 1 = PAGE_DIR_BASE. PID 2 = PAGE_DIR_BASE + MAX_PG_TABLES_SIZE.
    b. The first entry (of the page directory and page table) should be PG_USER_PRESENT_RW

    c. KEYBOARD_BUFFER buffer should be PG_USER_PRESENT_RW
    d. USER_PID_INFO should be PG_USER_PRESENT_RO
    e. GDT_LOC should be PG_USER_PRESENT_RO
    f. VIDEO_RAM should be PG_USER_PRESENT_RW
    g. All of kernel space should be PG_KERNEL_PRESENT_RW
    h. The first (and only entry) in the page directory should be PG_USER_PRESENT_RW
2. <u>Implement requestSpecificPage function.</u> This function attempts to allocate a page of memory at a specific virtual address for a process.
    a. Acquire lock on PROCESS_TABLE_LOC
    b. Allocate a page frame
    c. Insert the page frame address into the process's page table at the provided virtual address using the provided permissions
    d. Release the lock
    e. Return 1 if successful, 0 if unable to do so.
3. <u>Implement findBuffer() function using a first-fit algorithm.</u> This function attempts to find a contiguous chunk (buffer) of virtual memory within a process's virtual memory based on the number of pages needed. A pointer to the first buffer large enough should be returned. This function does not actually allocate the memory--it just finds a spot large enough. The range must start from TEMP_FILE_START_LOC.
4. <u>Implement requestAvailablePage() function</u>. This function allocates the first available page of virtual memory in the process's address space that is not used and returns a pointer to it. The range must start from TEMP_FILE_START_LOC.
    a. Acquire lock on PROCESS_TABLE_LOC
    b. Allocate an available frame
    c. Find first available (unallocated page) in the page table for that PID that is >= TEMP_FILE_START_LOC. (Meaning, a virtual address beginning at 0x210000 or larger in that address space.)
    d. Insert page frame address into the page table at that location. (You should calculate the page frame address based on the frame number.)
    e. Release lock
    f. Return pointer to the page allocated

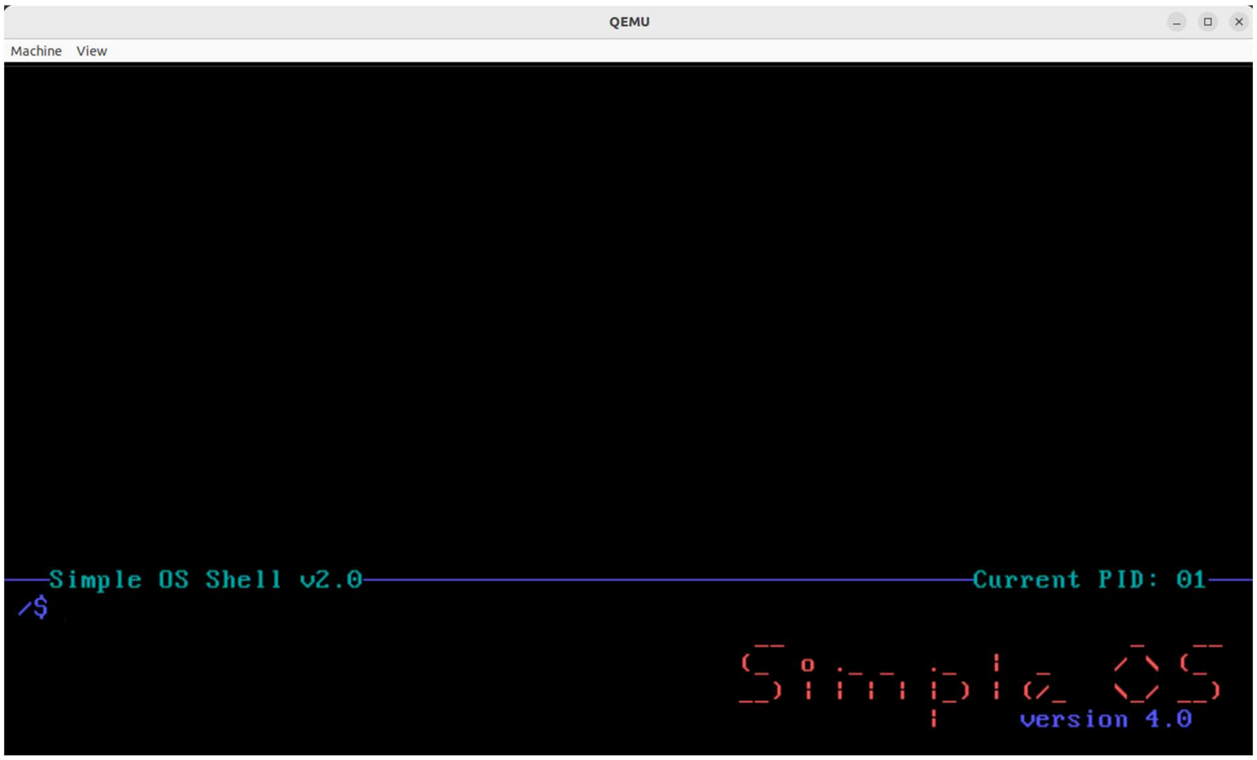
Additionally, be sure to use the constants (from constants.h) whenever possible. You should not be using a hard-coded value if there is a corresponding constant in constants.h, (but do not modify, add to, or alter constants.h in any way). You cannot create new helper functions and you should not create any new global variables. **Be sure to follow a professional coding standard, including the naming of your variables, structures, etc. You should use descriptive names that are appropriate. Nonsensical or whimsical names will lose points. You must also use the custom types where I have supplied them (e.g., uint8_t, uint16_t, uint32_t) unless you need a signed typed or one I haven't supplied. You will lose points if you don't follow this convention.**

Once you have implemented all of these functions and run “make”, it should look like the following screenshot. When you are happy with everything, upload the SUBMIT-ME.zip file to Canvas.

Additionally, you must attest and add the following honor statement to your Canvas submission (in the Canvas submission comments):

I, <state your name>, attest that this submission is my own work. I have not worked with anyone else on my implementation and have not used AI or other unpermitted sources to help me with my solution, in accordance with our definition of class cheating shown in Lecture 1.

Any submission that does not include this attestation will not receive any credit.



```
──Simple OS Shell v2.0────────────────────────────────────Current PID: 01──
/$

                                                            version 4.0
```