John L. Peterson
jlp5729
Prof. Chidambaram
August 15, 2022

# Project 5 - xv6 Pagefault Handler & File-backed mmap

## Part 5a - Pagefault Handler

### Approach and methods

My overall approach was to modify existing mmap code as little as possible and handle all new functionality with the pagefault_handler function inside trap.c. Before beginning PJ5, though, I had to make some modifications to my base code to pass the PJ4 hidden tests. This involved modifications to the argint and argptr in order to check whether free regions were allocated or unallocated, as well as actually deallocating unused mmap regions when calling munmap (my PJ4 solution skipped the deallocation portion failing one hidden test). At first I attempted to add another item to the mmap list struct to indicate if a region was mapped, but it became evident that the pagefault handler only allocates one page at a time, so this entry was not useful. The simple pagefault handler involved understanding the mappages call and moving the allocuvm/deallocuvm from my mmap function into the pagefault_handler function.

### Calculations

I did not utilize any additional tests of this function besides the built-in test; however, this pagefault_handler function was vital to the implementation of my mmap functionality in Part 5b, so after completing and testing part 5b, I felt confident that it was performing as expected.

### Discussion

The most difficult part of this was understanding the different bit flags for the trapframe, the write protection/user bits for page table entry, and how to get everything we needed out of just a trapframe entry. The xv6 book and the x86 documentation helped that make sense. I think it would have been interesting to write more tests that specifically tested the pagefault handler; this might have required more system calls to change values for page table entries or protection bits, but as mentioned above, these functions get exercised thoroughly in Part 5b so it was not required in the scope.

## Part 5b - mmap Part 2: File-Backed Mappings

### Approach and methods

The approach for part 5b was again to add most code in the pagefault_handler function of trap.c. This involved understanding further the page table entry flag bits, as well as the file system/read/write functionality. One of the first things I did was separate the mmap functionality into mmap.c and mmap.h, as my PJ4 implementation had various functions throughout proc.c and vm.c. The only new code required in vm.c was a function to retrieve the page table entry for a given page directory and address; this was necessary to handle PTE flags and write one page to memory on page faults. In proc.c, I had to update my fork function to actually make a deep copy of the mmap list, including file duplication and file descriptors, as this was not correct in my PJ4 implementation. Fileseek was relatively simple to implement after understanding the inode/file structure for xv6. To write async, this required various error checking as well as a loop through the memory region mmaped to the file to write out the data. My approach was page-by-page file write with dirty bit validation prior to writing, but I did not coalesce the writes as suggested as an improvement. Munmap also needed to be modified to deallocate regions when munmapping, as mentioned above in my 5a writeup. The rest of the changes involved pagefault_handler. This included checking mmap region type, getting the page table entry, checking and setting page table flags, as well as handling edge cases for large files, small files, mmap regions larger than file backing, and error validation for all (or rather most) function calls. Additionally, handling program closure on invalid memory access required some more tweaks, including calling exit() on an invalid mmap access to avoid an infinite loop where the program was not getting killed properly in the child function.

### Calculations

I adapted the test_6 function from the included tests to test large files as well as various edge cases mentioned on Piazza. I also used a big file generator found from the MIT PDOS class website, adapting it to generate large files for my hello.c test file to process. It was during this testing and revising process that the most tweaks/changes came. Passing included tests did not cover any of the edge cases mentioned, and each of them presented interesting challenges. One of the more difficult edge cases I encountered was read-only file-backed mmap regions. I was setting the flags for the PTE to read-only, but my own pagefault handler was trapping back to the pagefault handler when trying to write the read-only page on the initial file load accesses and getting stuck. The solution involved re-masking the PTE_W bit to enable write access when accessing the mmap region for the first time to load in the file, but not for any subsequent access. Another error that testing revealed was edge cases surrounding mmap regions that were larger than the included file. Instead of handling file writes/reads based on the size of the mmap region, I was utilizing the file length info from the inode to determine bounds of access, so this led to undefined behavior when accessing the mmap region outside of the mmap file. Additionally,

trying to write more bytes to the file than were in the initial file was failing, as well. I needed to set more fine-tuned access and boundaries to memory regions inside the pagefault handler. Another thing I learned while testing was the way that xv6 and unix handles files: specifically, the way that functions like strcpy handle null-termination. My files were getting updated properly but my test was not reflecting that because the null termination bits I had set inside the file weren't printing the regions I expected. To summarize, this project forced me to confront my lack of testing experience and I spent about 75% of the time alternating between testing and troubleshooting, as opposed to previous projects where testing was more of an afterthought.

## Discussion

As mentioned above, this project required much more extensive testing, as well as the understanding of the underlying operating system. Much of the groundwork was laid in PJ4, and my mmap function base worked well enough after tweaks from the hidden cases, allowing me to focus on file-backing and pagefault handling. I would have liked to optimize my pagefault_handler further, as a lot of the code was added in somewhat haphazardly as tests revealed more issues with handling edge cases. Additionally, if I had more time I would have implemented more optimizations in msync including the specification-mentioned idea to coalesce filewrite calls when adjacent pages are dirty.

I would have likely made some more system calls to aid in testing if I were trying to dive further, including the ability to manually change flags in page tables or trigger edge cases without having to carefully craft file access calls at a certain boundary.

Overall, getting full marks, I was pretty happy with my solution. I feel like I gained a great appreciation for the intricacies of kernel programming and was able to synthesize the experience from the other projects with the knowledge gained in lectures.