Vaibhav Chattree
vc6246

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

The first part of the project had us implement lazy page allocation. This is where the pages only get allocated when there is a pagefault. First I removed the mappages call in mmap. Mmap still will claim the virtual addresses and add the information to the memory region data structure. I then created a new method in trap.c which is the pagefault handler. I passed this method the entire trapframe. I use a struct to keep of the mmap regions which also contains a pointer to the next mmap region. In pagefault handler, I create a new instance of this struct. I then get the fault address by rounding down the rcr2. While the region is not page aligned, I check that I am between the fault address and I can write. I then call memset and mappages. If not, I just break. I check if I am in the trap handeler, and if so, I panic. Also, I added the debugging print statement as instructed for when the process does not run correctly.

The second part of the project had us implement file backed mmap regions. These regions are created with the contents of the file descriptor, and they can write the changes made to the region to the file. This has the advantage of reducing writes to persistent storage which is a heavy operation. Also, multiple changes can be written at once. The first change I made was to add the function fileseek in file.c. I update the file struct offset field to the passed in offset value. I also used ilock and iunlock to acquire the inode lock to prevent potential races. This method allows us to seek around the file before reading or writing. I then created the new header file mman.h for which I added flags for if pages can be written, and if a mmap region is anonymous or file backed.

In the mmap function, changes were needed to handle file descriptors. I check the flag for anonymous vs file descriptor. If the flag is set to file descriptor, and the fd is equal to the fdalloc curproc ofile, I use filedup to duplicate the file descriptor. In munmap, I added a fileclose on the curproc ofile mmap region match fd. I then use deallocuvm with parameters of curproc pgdir, region match addr and length, and region match addr. In msync, I first find the memory region using the start addr and length. I loop through the regions to do this. I then write the pages to the file. I check if a page is mapped with walkpagedir. Then I use filewrite for the curproc ofile region match fd to write to the file descriptor. I also use pte and check the dirty bit, PTE_D, so I am only writing pages which have been changed on the disk.

I passed all of the provided test cases, and also wrote some of my own test cases. The first test checks if the mmap return address is page aligned. I passed the case by checking to make sure the free space address modulus the pgsize is 0. The third test checks for anon mmap without prot write. I pass this test by keeping a prot flag in my mmap region struct which I check. The fifth test is a check on file base mappings persisting after file close. I pass this by ensuring my mappings are written to the file and are not lost on file close. I also wrote my own tests to test the dirty bit, msync for multi page file backed regions in both main and child processes, and that passing standard fds into mmap throws an error.

Vaibhav Chattree
vc6246

The testing results helped me to realize errors in my code and fix them. I failed the fourth and sixth test initially. The fourth test made me go back and update my mmap to handle the spanning of multiple pages. And the sixth test made sure I caught a bug in my mysnc implementation where I had the wrong logic in an if statement. Overall, I believe my solution worked since I followed the spec closely to ensure I was meeting the requirements. I used a struct for the mmap regions which also pointed to the next region which worked well. I also caught bugs after initially failing some of the test cases. This project helped deepen my understanding of page fault handling and file backed memory mapping. Implementing lazy page allocation was useful to improve the efficiency of the system. It also helps to create the illusion of unlimited resources. File backed memory maps also improve the efficiency of the system. This is because writes to persistent storage are avoided, and then multiple changes are written at the same time. This project developed my understanding of some efficiencies which exist in full-fledged operating systems.