John L. Peterson
jlp5729
Prof. Chidambaram
July 26, 2022

Project 4 - xv6 Kmalloc & Anonymous mmap

Part 4a - Kmalloc

Approach and methods

My overall approach was to reuse as much of umalloc.c as possible. This involved digging into the C book and the xv6 manual to determine what needed to change. For the purpose of this portion, I ended up being able to change the call to sbrk to a call to kalloc inside of morecore. This fulfilled the requirement of requesting more memory; however, it also required adding a check inside kmalloc to check that requested size wasn't larger than size possible through kalloc (one page size). Adding a panic to check size less than PGSIZE-sizeof(Header struct) did the trick. kmfree function did not need any modifications. Additionally, I implemented the null page checks from Project 3 to help avoid any accidental null pointer accesses or buggy behavior.

Calculations

I did not utilize any additional tests of this function besides the built-in test; however, this kmalloc function as well as kfree/kmalloc were vital to the implementation of my mmap functionality in Part 4b, so I felt confident that it was performing as expected.

Discussion

The most difficult part of this was understanding the malloc function as implemented by Kernighan and Ritchie. Once I grokked that, making the small changes necessary made sense. I think it would have been interesting/worthwhile to write more tests that specifically tested kmalloc and kalloc; this might have required more system calls so that the test program could access the kernel memory allocated and deallocated, but as mentioned above, these functions get exercised thoroughly in Part 4b so I did not determine that worth my time.

Part 4b - mmap Part 1: Anonymous Mappings

Approach and methods

My overall approach included adding a linked list for both mmap regions and munmap regions, inside the proc struct for each process. This involved initialization functions, helper functions to edit and access these data structures, a function to copy mappings when a process is forked, and mmap/munmap functions added inside vm.c. Additionally, I added functions to print the mmap and munmap lists, and system calls to access those functions (this will be detailed more in the Calculations section).

Initially, I misunderstood the project specification, and did not realize that all mmap regions had to start at a page-aligned address. I struggled to build helper functions and a linked list structure that would specify start and end of allocations, where inside or over a page a mapping was. It was all working decently well but I realized my mistake running the included test_6 and had to revamp my code. This led to more issues, since my functions and data structures were now needlessly complex, and led to debugging issues and code that could certainly be more efficient and easily understandable than it is currently.

My decision was to keep the free list sorted low to high, as this would allow for easier merging of adjacent munmap regions; this was not as necessary once reducing the complexity to meet the project spec, but still came in handy for this purpose.

Additionally, I modified the argptr function within syscall.c; this would allow me to use argptr to parse an address that was above process size, but below KERNBASE (i.e. able to be mmap'ed). This made more sense to me than, for example, writing a new function or doing my own validation checks on a call to argint function.

To handle programs that finished, code was required inside freevm to deallocate the mmap and munmap linked lists, as well in the wait() function. I did not put any code in exit() because after exit is called, it appears that xv6 then executes wait, so I did not want to duplicate efforts or have issues calling kmfree on data structures that were no longer extant. Also that if a process is killed by the kernel, I don't believe that exit() gets called.

Calculations

I utilized the included tests first to ensure basic functionality, but quickly agreed with the TA's suggestion that more tests were needed. I ended up adapting some of the existing tests into hello.c file, in order to test an exhaustive region of mmap and munmap consecutively.

It became clear that I had misunderstood the way to utilize deallocuvm and allocuvm, and was allocating memory that was not necessary to be allocated; this helped prevent my kernel from running out of memory when mmap'ing a large number of sections. This test file also included a random number generator adapted from an academic paper (link in the code), which could help me ensure that I was handling edge cases.

System calls printmmap and printmunmap became helpful when troubleshooting the helper functions that accessed/merged my linked list. I had an issue where free mappings were not joining or splitting properly; this was due to the complexity of my initial code not being simplified properly.

I am not certain that I fully handled forked processes properly; I made sure that the mmap and munmap linked list structures were copied, but I did not end up utilizing my full copy_mmap function as written because none of the tests I wrote made it seem like this was necessary. Perhaps with file-backed mmap this functionality will become more relevant/required.

Additionally, I did not implement or test concurrency; this could have been tested more using fork and wait calls and seeing if the program ran out of memory or had trouble accessing the memory it should have had access to.

Discussion

This assignment was challenging, and is certainly intended to be a "warmup" for the following project. I believe that if I had taken more time to thoroughly understand the project specification and ask questions on Piazza before diving in, that I would be in a better place for the next assignment.

I did not implement any locks or concurrency beyond copying the linked lists on a fork call, and as mentioned above, I am not certain that forked processes are handled correctly. However, I do feel confident that I do not have any major memory leaks or issues with data structures not being tracked appropriately.

I did not deallocate memory or shrink the process size when items were munmap'ed from the end of the process space. Ideally I would like to implement this functionality and I think it would make my kernel more efficient. As of now, mmap'ing a large space and then munmapp'ing leaves the memory allocated when it would be more efficient to remove this space and reduce the process size to allow for more concurrent memory usage and fewer page tables required for each process.

If I had more time, I would implement the above, clean up my code further, make a new c file for my mmap functions (which I may do for project 5), as well as more thoroughly test the hint address capability and concurrency via fork(). I am not certain what tests I am failing on the grader, but once those are released I will plan to utilize them in improving my base code before tackling the file-backed and other mmap functionality in the next project.