CS4375-13948 Fall 2023 Homework Report

Franz Reyes 11/10/2023

Fareyes4@miners.utep.edu

**HW 4: Lazy Allocation for xv6**

Task 1. freepmem() system call

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

* In this task, we added a new feature called freepmem() to xv6, and then we tested it using two programs: free and memory-user. The free program helps us see how much free memory is available, and memory-user keeps asking for memory and giving it back. By running them at the same time, we can figure out the maximum amount of memory we can use before the system realizes there is no more memory available.
* Doing this helps us make sure that freepmem() works correctly, showing the right amount of free memory. If there are any problems, like not getting the memory we need, we can figure out why and fix it. This task is like a hands-on lesson in understanding how xv6 manages its memory, teaching us more about how the computer keeps track of and shares memory.

Task 2. Change sbrk() so that it does not allocate physical memory.

A screenshot of a computer

Description automatically generated

* In Task 2, we changed how the sbrk() system call works in xv6. Instead of making it allocate real memory, we made it only change the virtual space (like a placeholder) without grabbing any actual memory from the computer. To do this, we removed a part in the code that was responsible for grabbing real memory.
* The sbrk() now only adjusts a number that tells the system how much virtual space the program can use for its heap (a place to store dynamic data). This change means that, even though the program thinks it has more space, it won't really have more memory until it starts using it. This adjustment gives us a different way to look at how the computer manages memory, showing that we can separate the idea of space and the actual memory used. It's like saying, "Here's more room for you," without actually giving more stuff to put in that room right away.

Task 3. Handle the load and store faults that result from Task 2

* First with task 2 I changed how the computer's memory works in xv6 by making a system call called sbrk(). Instead of asking for more real memory, this call only changed a pretend memory value, like making more room on a list without getting more information to fill it. To see if it worked, Icreated a program that uses this memory and checked if the computer acted as expected.
* Now there is a need fix the problem that task 2 created. When trying to use the pretend memory, the computer gets lost at some point because there is no more pretend memory or even real memory. One way to work on this was by adjusting the code so that, if the program asks for more room in its pretend memory, the computer can adjust to find whatever space is left. The computer knows how to get more real memory and arrange things so the program can use it without causing errors.
* To overcome difficulties, much debugging and error-checking was needed. Print statements were used as well as debugging tools were used to trace the execution flow and identify any upcoming errors. Properly handling failures in memory allocation and page table mapping helps us identify specific case of errors. Task 3 enhances understanding of the relationship between virtual and physical memory management in xv6, providing practical insights into fault handling mechanisms and dynamic memory allocation strategies within an operating system.

Task 4. Fix kernel panic and any other errors.

* In Task 4, I addressed errors that came up from Task 3 in the xv6 operating system. I made changes to the kernel/vm.c file. Specifically, I modified the uvmunmap() function to include checks ensuring that the pages being unmapped were indeed mapped before proceeding. This adjustment aimed to prevent errors related to virtual memory management and improved the system’s performance.
* Task 4 imrpoved my understanding of virtual memory behaviors in xv6 and emphasized the importance of systematic error identification and problem solving. The experience highlighted the roles of different components in the operating system. While overcoming difficulties associated with debugging, the task provided practical insights into debugging strategies and reinforced the learning of careful analysis in resolving operating system issues.

Task 5. Test your lazy memory allocation.

A screenshot of a computer

Description automatically generated

* A test case was used in Task 5 to explore xv6's lazy memory allocation. The case allocated and accessed memory to assess how xv6 dynamically allocated physical memory pages on-demand. The tests aimed to reveal insights into the OS's memory management strategy and its responsiveness to varied access patterns, involving challenges in debugging potential memory allocation issues.