CS 4375 Fall 2025
Homework 4: Lazy Allocation for xv6
75 points

MD Armanuzzaman

Dude Date: November 10, 2025 (11.59 PM))

xv6 currently allocates and maps physical memory frames for all allocated virtual memory pages. Most operating systems wait to allocate a physical memory frame until a virtual memory page that is part of the heap space is accessed. This lazy allocation saves both time and space is not all of the allocated virtual memory is actually used. For this assignment, we will implement lazy allocation of the heap for xv6 and also allow overcommitment of the physical memory.

Task 1. `freepmem()` system call (15 pts)

You are given the code for an xv6 free command in the file `free.c`. You should place this code in the user directory and add the command to `UPROG` in Makefile. The code calls the `freepmem()` system call which does not currently exit. The user prototype for `freepmem()` is

```
uint64 freepmem(void);
```

You should place this declaration in `user/user.h` and add an entry for freepmem in `usys.pl`. You will also need to modify `kernel/syscall.h`, `kernel/syscall.c`, `kernel/sysproc.c`, and `kernel/kalloc.c` to implement the `freepmem()` system call. After you implement the `freepmem()` system call, test it using the provided free command.

You are provided with another test program in `memory-user.c`. This program calls `sleep()` so that you can run it in the background and run the free command between allocations. Show and explain the results when you do this. How much memory can you request before `malloc()` fails?

Task 2. Change `sbrk()` so that it does not allocate physical memory (15 pts)

The `sbrk()` system call grows the heap space if needed to satisfy a `malloc()` request. Note that the heap space is in the process's virtual address space. The end of the heap space is marked by the `sz` field in `struct proc`. Currently `sbrk()` calls `growproc()` which calls `uvmalloc()` to allocate physical memory for the newly allocated virtual pages. Modify `sbrk()` so that it changes to `sz` field in `struct proc` – i.e., so that it allocates virtual memory space – but remove the call to `growproc()`. You will not get a `usertrap()` when you attempt to run any xv6 commands that use heap memory.

Task 3. Handle the load and store faults that result from Task 2 (15 pts)

Note the value of scause from Task 2. Accessing unallocated memory will result in a load or a store fault. See the RISC-V Privileged ISA document for the relevant exception codes. To handle the load and store faults that result from Task 2, you will need to check for these exception codes in `usertrap()` in `kernel/trap.c`. If the exception is a load or store fault, then you should also check if the faulting address (which will be in the stval register) is within the process's allocated virtual memory. If it is, then it is a valid virtual address and the fault occurred because physical memory has not been allocated and the page table mapping has not been installed. In this case, you should handle the fault by allocating a physical memory frame (hint: use `kalloc()`) and installing the page table mapping for the virtual page that contains the faulting address (hint: `use mappages()`).

Task 4. Fix kernel panic and other errors. (15 pts)

After you complete Task 3 and attempt to run xv6 commands, you will find that you get some errors, such as a kernel panic from `uvmunmap()` since that function expects all virtual memory pages to have been mapped. You may see other errors as well. Find and fix the errors. Try to figure out what is causing each error, rather than trying random things to try to fix it.

Task 5.Test your lazy memory allocation. (15 pts)

Test your lazy memory allocation thoroughly and show the results. Test cases should include: 1) allocating and freeing memory without touching it, 2) allocating and touching memory, 3) allocating memory and randomly touching just some pages.

Extra Credit Task 6: Enable use of the entire virtual address space. (15 pts) The current xv6 user library code seems to assume that heap size and the size of malloc requests will not be larger than a 32-bit value. Modify the code as needed to allow the heap to grow until the virtual memory space is exhausted and to allow the size of a malloc request to be as large as the remaining contiguous available virtual memory space.

Extra Credit Task 7: Allow a process to turn memory overcommitment on and off. (15 pts) Instead of running everything with lazy allocation and possible memory overcommitment, make the default be not to overcommit and add a system call that will let a process turn memory overcommitment on and off. For example, a program could turn overcommit on to allocate a large array that is expected to be used sparsely, and then turn overcommit back off.

Turn-in procedure and Grading

Please use the accompanying template for your report. Convert your report to PDF format and push your hw3 branch with your code and report to your xv6 GitHub repository by the due date. Also, turn in the assignment on Teams with the URL of your GitHub repo by the due date. Give access to your repo to the TA.

This assignment is worth 100 points. The breakdown of points is given in the task descriptions above. The points for each task will be evaluated based on the correctness of your code, proper coding style and adequate comments, and the section of your report for that task.

You may discuss the assignment with other students, but do not share your code. Your code and lab report must be your own original work. Any resources you use should be credited in your report. If we suspect that code has been copied from an online website or GitHub repo, from a book, or from another student, or generated using AI, we will turn the matter over to the Office of Student Conduct for adjudication.