

UG HW5: Anonymous Memory Mappings for xv6

Task 1.

```
usertests      2 15 150432
grind          2 16 37208
wc             2 17 24816
zombie        2 18 21984
private       2 19 23840
console       3 20 0
$ private
usertrap(): unexpected scause 0x0000000000000002 pid=4
             sepc=0x0000000000000028 stval=0x0000000000000000
$
```

- a. The system call to `mmap()` creates a shared memory region of size specified by the parameter `buffer_t` which is a struct in the program itself.

The system call `munmap()` is used for the unmapping of the previously used shared memory when it's no longer in use. In the case of this program the `munmap` function was commented out which means that after execution the memory was not released and therefore causing errors.

The program aborted because the `munmap` function was not being called after execution `mmap` which makes it likely for the operating system to produce errors.

- b. I implemented a function called `valid_address()` which will ensure the address is within the range of that process block of memory. I also used `kalloc()` to allocate physical memory in case of page faults. Lastly I included `mappages()` which takes care of the mapping between previously mentioned physical memory to the fault virtual address.

```
student@Ubuntu:~$ cd myXV6
student@Ubuntu:~/myXV6$ make qemu
qemu-system-riscv64 -machine virt -bios none
3 -nographic -drive file=fs.img,if=none,format=raw,cache=writeback,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ private
total = 55
$
```

- c. The issue of running the program with the function call to `munmap` commented out is that the memory allocated by `mmap` was never actually released which can cause a memory leak which can cause the system to run out of memory if this happens often, this causes a kernel panic.

Task 2.

- a. `uvmcopy()` creates private mappings which ensures that changes made to the child process do not extend to other subprocesses. `Uvmcopyshared()` is the exact opposite because it allows the duplication of the parent's page table for the child's table as well as their mapping. Meaning that a change in just one of the processes will affect all the others too.
- b. The modifications added to `fork()` function ensure that both shared and private memory are allocated correctly, by duplicating the memory mapped region table from the parent process to the subprocess created. In the case of private mappings, the memory-mapped region creates new isolated memory for the subprocess copying from the parent, whereas for shared mappings, the subprocess memory is already part of the existing memory that was created by the parent process.
- c. The difference in outcomes for both system calls has to do with the fact `prodcons1` is utilizing the `MAP_SHARED` flag with a `TRUE` value, which as previously mentioned, allows subprocess created to share memory and changes among them. So, all subsequent processes in `prodcons1` are contributing to the total.

```
student@Ubuntu:~$ cd myXV6
student@Ubuntu:~/myXV6$ make qemu
qemu-system-riscv64 -machine virt -bios none
3 -nographic -drive file=fs.img,if=none,format=raw,drive=x0,bus=virtio-mmio-bus.0

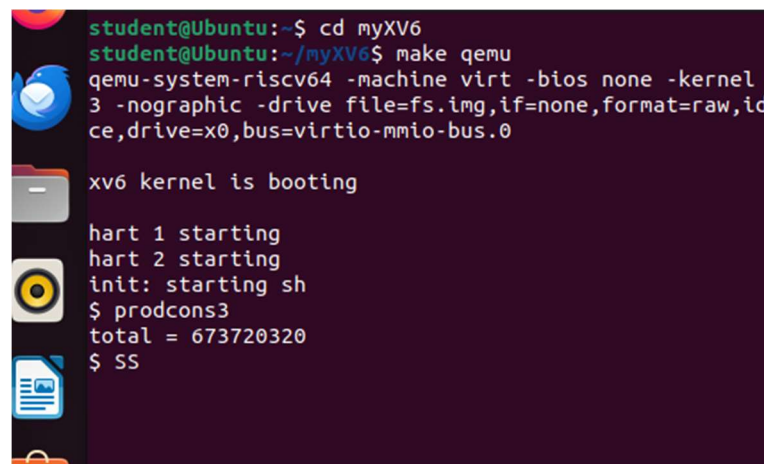
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ prodcons1
total = 55
$ prodcons2
total = 0
$
```

Prodcons2 in the other hand, sets the MAP_PRIVATE flag to a TRUE value, which isolates the memory region for the subprocesses created and prevents them from reflecting updates when executing, therefore changes to the total are never made and they remain at zero.

Task 3.

- a. The main reason the prodcons3 is producing false data is that in the producer() function itself, the index of the buffer is not being dynamically increased but rather is set to 1 in every iteration, this could be the main reason.
- b. This was fixed by adding some code in the usertrap() function in order to create a new mapping for the physical page to the page tables that are shared by the entire structure of processes and their memory.

A terminal window with a dark purple background. The prompt is 'student@Ubuntu:~\$'. The user enters 'cd myXV6' and then 'make qemu'. The output shows 'qemu-system-riscv64 -machine virt -bios none -kernel ... 3 -nographic -drive file=fs.img,if=none,format=raw,id=... ce,drive=x0,bus=virtio-mmio-bus.0'. Below this, the text 'xv6 kernel is booting' appears. Then, 'hart 1 starting', 'hart 2 starting', and 'init: starting sh' are shown. The user enters '\$ prodcons3' and the output is 'total = 673720320'. Finally, the user enters '\$ ss'.

Summary:

I got to understand more in depth the functioning of processes and memory, specifically how each process can have different memory regions that are isolated from its own “family” even if it they are the direct result of the main parent process, likewise it was useful to understand the idea behind the shared memory-mapped regions and how the updates in a specific process have effects on the rest of the “family” processes.