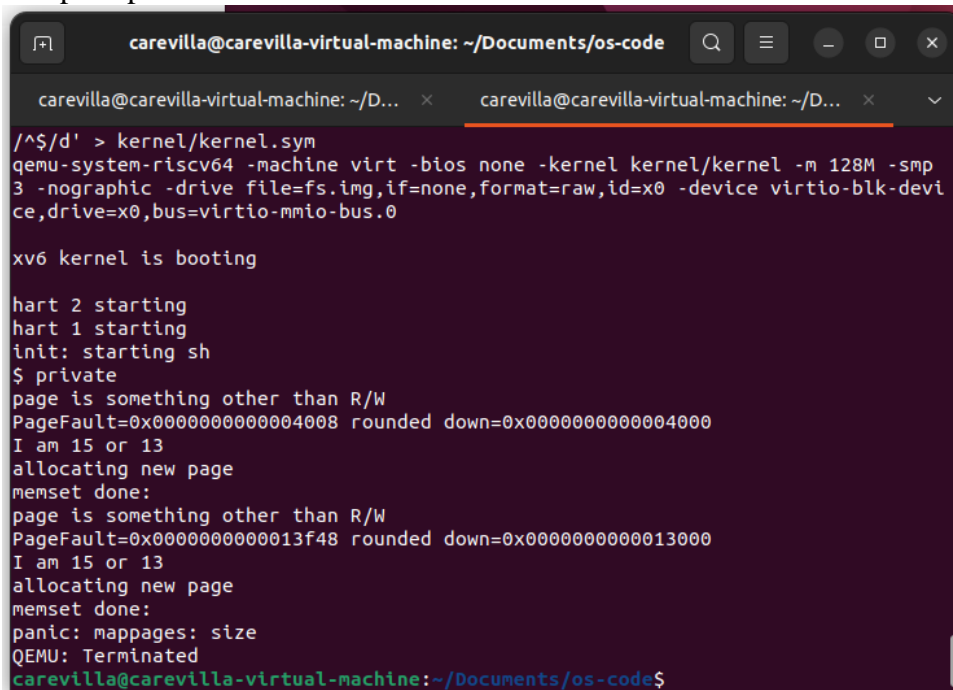


UG HW5: Memory Mappings for xv6

Task 1: Implement mmap(), munmap() with lazy allocation:

- 1) For part a, we were asked to implement mmap and munmap system calls in our xv6. This step was straightforward thanks to the word document which provided us with the code that must be implemented. We were given the private.c file to test our implementation. The private file uses mmap system call by the following
 - a) Program calls mmap() to create memory region of the size determined by the buffer_t structure
 - b) The Program also passes PROT_READ | PROT_WRITE which are flags to give both reading and writing permissions in the shared memory region
 - c) The Program also passes MAP_ANONYMOUS | MAP_PRIVATE which are flags to create anonymous mapping and show the mapping is private to the process
- 2) Before implementing the changes in trap.c and fixing my homework 4 lazy allocation I was prompted with this error.



```
carevilla@carevilla-virtual-machine: ~/Documents/os-code
carevilla@carevilla-virtual-machine: ~/D... x carevilla@carevilla-virtual-machine: ~/D... x
/^$/d' > kernel/kernel.sym
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp
3 -nographic -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-devi
ce,drive=x0,bus=virtio-mmio-bus.0

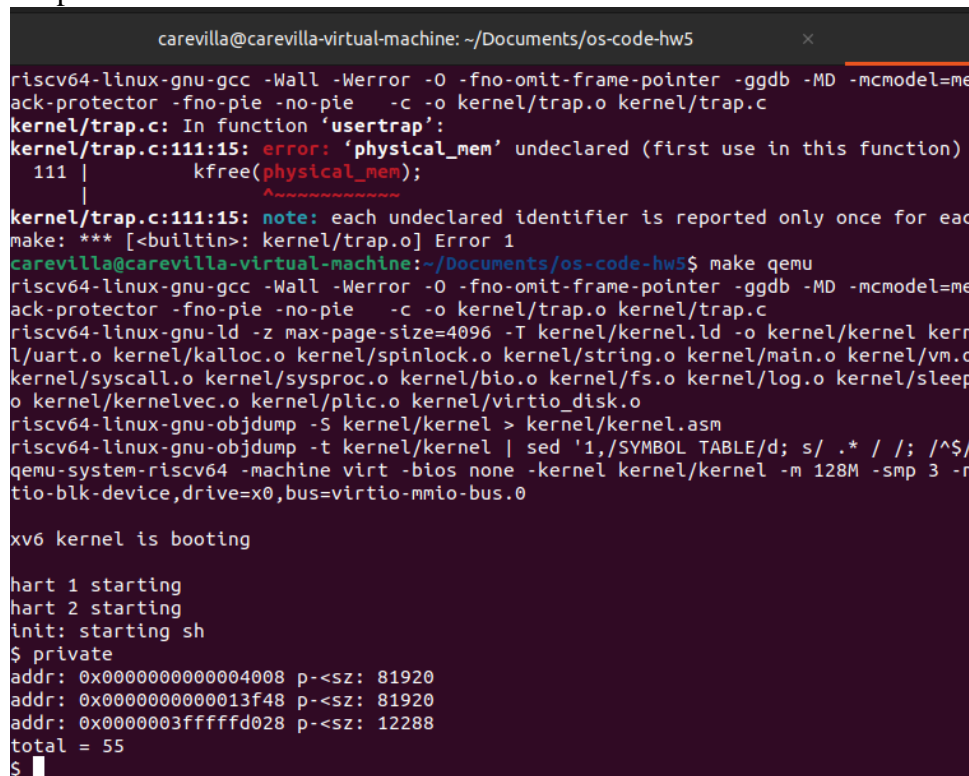
xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ private
page is something other than R/W
PageFault=0x0000000000004008 rounded down=0x0000000000004000
I am 15 or 13
allocating new page
memset done:
page is something other than R/W
PageFault=0x0000000000013f48 rounded down=0x0000000000013000
I am 15 or 13
allocating new page
memset done:
panic: mappages: size
QEMU: Terminated
carevilla@carevilla-virtual-machine:~/Documents/os-code$
```

after various debugging and trying to figure out what was wrong, I found out that for some unknown reason when the private command was being executed on the third store fault was never being sent to usertrap and somewhere my mappages size was being given

a value of 0. I was never able to fix the problem, so I ended up having to redo homework 4 to ensure my lazy allocation was properly implemented. The error the private command should cause is that the allocated memory ends up being greater than the page size so it must access the mmr to allocate the memory.

- 3) The steps I took to modify usertrap were the following
 - a) I added a helper function called checkAddress which is responsible for ensuring the faulting address is within an allocated memory range or within a valid memory-mapped-region
 - b) Once the above is confirmed kalloc() was used to allocate physical memory when the valid page fault occurs
 - c) Called mappages() to map the allocated memory from kalloc and to ensure proper permissions are set to the faulting virtual address.
- 4) The difficulties I encountered during this task were understanding how page faults work and how to handle them in usertrap. Understanding memory management and understanding how virtual memory and physical memory differ. Understanding how the memory-mapped-region structure can be utilized to help us allocate a faulty address. The steps I took to overcome these challenges were extensive debugging, testing, understanding xv6 operating system, and memory management. After fixing these issues the private command ran as followed



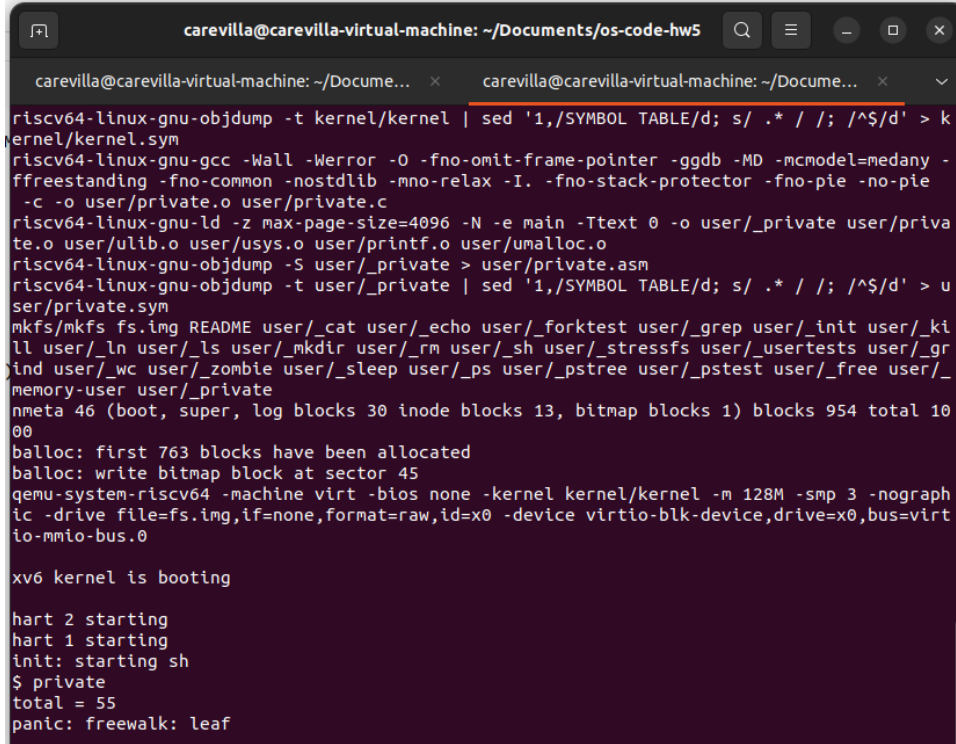
```
carevilla@carevilla-virtual-machine: ~/Documents/os-code-hw5
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -MD -mcmodel=me
ack-protector -fno-pie -no-pie -c -o kernel/trap.o kernel/trap.c
kernel/trap.c: In function 'usertrap':
kernel/trap.c:111:15: error: 'physical_mem' undeclared (first use in this function)
  111 |         kfree(physical_mem);
      |               ^
kernel/trap.c:111:15: note: each undeclared identifier is reported only once for each
make: *** [<built-in>: kernel/trap.o] Error 1
carevilla@carevilla-virtual-machine:~/Documents/os-code-hw5$ make qemu
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -MD -mcmodel=me
ack-protector -fno-pie -no-pie -c -o kernel/trap.o kernel/trap.c
riscv64-linux-gnu-ld -z max-page-size=4096 -T kernel/kernel.ld -o kernel/kernel.elf
kernel/kernel.elf: warning: kernel/uart.o: kernel/kalloc.o: kernel/spinlock.o: kernel/string.o: kernel/main.o: kernel/vm.o:
kernel/syscall.o: kernel/sysproc.o: kernel/bio.o: kernel/fs.o: kernel/log.o: kernel/sleep.o: kernel/kernelvec.o: kernel/plc.o: kernel/virtio_disk.o:
riscv64-linux-gnu-objdump -S kernel/kernel > kernel/kernel.asm
riscv64-linux-gnu-objdump -t kernel/kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -r
tio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ private
addr: 0x0000000000004008 p-<sz: 81920
addr: 0x00000000000013f48 p-<sz: 81920
addr: 0x00000003ffffffd028 p-<sz: 12288
total = 55
$
```

- 5) The final task for task 1 asked us to comment out the call munmap(). When the line is commented out it causes a kernel panic, which will be shown in the next image below. The reason we get a kernel panic is because the munmap releases the allocated memory in the buffer caused by the private command. By commenting out this function means

structures like page tables still have active entries for memory regions even though they are no longer being used. This leads to errors in memory access and allocation causing kernel panic. After implementing the changes in freeproc the private command works the same as before because now we have released the allocated memory not being used in the page table entries.



```
carevilla@carevilla-virtual-machine: ~/Documents/os-code-hw5
carevilla@carevilla-virtual-machine: ~/Docume... x carevilla@carevilla-virtual-machine: ~/Docume... x
riscv64-linux-gnu-objdump -t kernel/kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > k
kernel/kernel.sym
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -MD -mmodel=medany -
ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie
-c -o user/private.o user/private.c
riscv64-linux-gnu-ld -z max-page-size=4096 -N -e main -Ttext 0 -o user/_private user/priva
te.o user/ulib.o user/usys.o user/printf.o user/umalloc.o
riscv64-linux-gnu-objdump -S user/_private > user/private.asm
riscv64-linux-gnu-objdump -t user/_private | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > u
ser/private.sym
mkfs/mkfs fs.img README user/_cat user/_echo user/_forktest user/_grep user/_init user/_ki
ll user/_ln user/_ls user/_mkdir user/_rm user/_sh user/_stressfs user/_usertests user/_gr
ind user/_wc user/_zombie user/_sleep user/_ps user/_pstree user/_ptest user/_free user/_
memory-user user/_private
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 954 total 10
00
ballocc: first 763 blocks have been allocated
ballocc: write bitmap block at sector 45
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nograph
ic -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virt
io-mmio-bus.0

xv6 kernel is booting

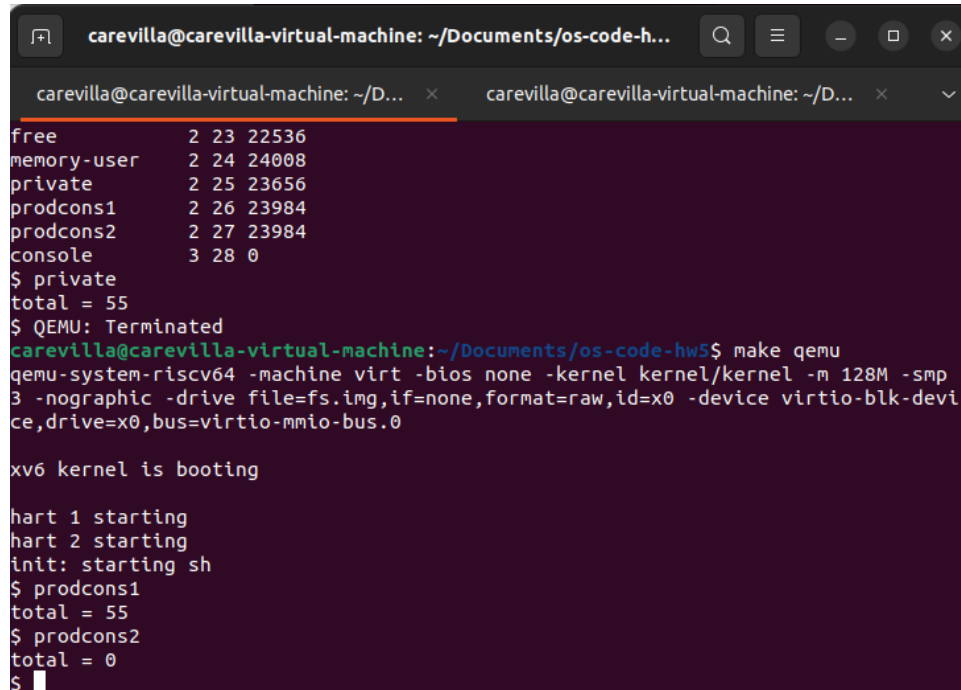
hart 2 starting
hart 1 starting
init: starting sh
$ private
total = 55
panic: freewalk: leaf
```

Task 2:

- 1) To briefly describe the difference between `uvmcopy` and `uvmcopyshared` function.
 - a) `uvmcopy` allocates a new physical memory page, copies the content, and maps the new physical memory to the child's page table. This allows independent copies of each physical memory for the parent and child. This function is suitable when we want the child process to have an independent copy without "sharing" with the parents.
 - b) `uvmcopyshared` doesn't allocate a new physical memory page instead it maps the physical memory in the parent's page table to the child's page table allowing both processes to share the same physical memory and any modifications are seen to both processes. '
- 2) After modifying the fork function, we can now effectively handle both private and shared memory mappings. For private mappings each valid mmr leads to the allocation of new private memory for the child, basically copying from the parent process. On the other hand, for shared mappings, the child is integrated into the existing memory family. The way we determine the

type of mapping to be utilized is by the MAP_PRIVATE and MAP_SHARED flags and those flags are what gives the distinction to call uvmcopy() or uvmcopyshared().

- 3) Looking at prodcons1 and prodcons2 test files we can see that prodcons1 utilizes a MAP_SHARED flag allowing changes to be shared from both parent and child process leading to a successful output of 55. On the contrast, with prodcons2 it utilizes a MAP_PRIVATE flag meaning the child process has independent map sharing and it is not visible to the parent, which yields an output of 0. Upon running both prodcons1 and prodcons2 the outputs are shown in the image below.



```
carevilla@carevilla-virtual-machine: ~/Documents/os-code-h...
carevilla@carevilla-virtual-machine: ~/D... x carevilla@carevilla-virtual-machine: ~/D... x
free          2 23 22536
memory-user   2 24 24008
private       2 25 23656
prodcons1     2 26 23984
prodcons2     2 27 23984
console       3 28 0
$ private
total = 55
$ QEMU: Terminated
carevilla@carevilla-virtual-machine:~/Documents/os-code-hw5$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp
3 -nographic -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-devi
ce,drive=x0,bus=virtio-mmio-bus.0

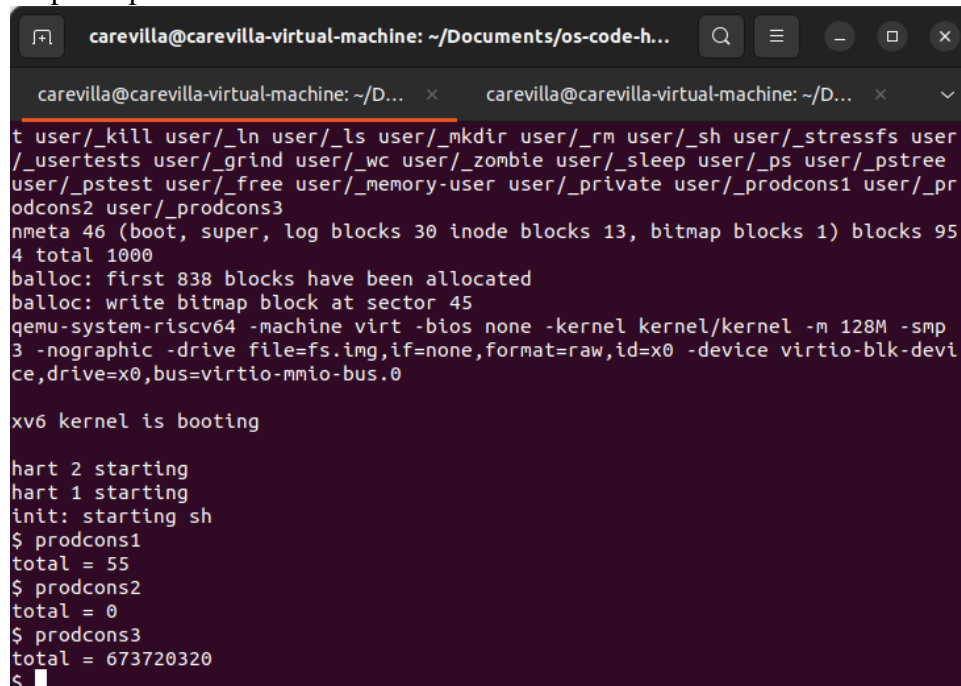
xv6 kernel is booting

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

Task 3:

1. For task 3, I was able to successfully add the new prdocons3 test file and noticed upon running the command the output yielded an extremely large number. Although I am not sure exactly what causes this large input, I did notice upon looking at the source code that the producer statically always assigns the buffers next index to be 1. This leads me to conclude that since the producer is failing to correctly increment the next index it is leading to problems with managing the shared memory region correctly. The image of

output is provided below

A terminal window with a dark background and light-colored text. The window title is 'carevilla@carevilla-virtual-machine: ~/Documents/os-code-h...'. The terminal shows the output of an 'xv6' kernel boot. It lists various user programs like _kill, _ln, _ls, _mkdir, _rm, _sh, _stressfs, _usertests, _grind, _wc, _zombie, _sleep, _ps, _pstree, _ptest, _free, _memory-user, _private, _prodcons1, _prodcons2, _prodcons3. It then shows 'nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 954 total 1000', 'balloc: first 838 blocks have been allocated', 'balloc: write bitmap block at sector 45', and 'qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0'. The output continues with 'xv6 kernel is booting', 'hart 2 starting', 'hart 1 starting', 'init: starting sh', and then three lines of '\$ prodcons' commands showing 'total = 55', 'total = 0', and 'total = 673720320'. The prompt '\$' is visible at the bottom.

```
carevilla@carevilla-virtual-machine: ~/Documents/os-code-h...
carevilla@carevilla-virtual-machine: ~/D... x carevilla@carevilla-virtual-machine: ~/D... x
t user/_kill user/_ln user/_ls user/_mkdir user/_rm user/_sh user/_stressfs user
/_usertests user/_grind user/_wc user/_zombie user/_sleep user/_ps user/_pstree
user/_ptest user/_free user/_memory-user user/_private user/_prodcons1 user/_pr
odcons2 user/_prodcons3
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 95
4 total 1000
balloc: first 838 blocks have been allocated
balloc: write bitmap block at sector 45
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp
3 -nographic -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-devi
ce,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ prodcons1
total = 55
$ prodcons2
total = 0
$ prodcons3
total = 673720320
$
```

Summary:

After completion of this homework assignment, I feel I have gained a greater understanding of how operating systems manage memory. Understanding how memory is allocated, mapped, and protected. I was able to further enhance my skills with Error Handling and debugging specifically thanks to task 1 part b. I also was able to improve on how to properly handle page fault by first validating the fault address, allocating physical memory, and mapping the addresses into the page table.