

HW 4: Lazy Allocation for xv6

Task 1. freepmem() system call

```
xv6 kernel is booting

init: starting sh
$ free
133390336
$ free -k
130264
$ free -m
127
$ memory-user 1 4 1 &
$ allocating 0x0000000000000001 mebibytes
malloc returned 0x00000000000003010
freeing 0x0000000000000001 mebibytes
allocating 0x0000000000000002 mebibytes
malloc returned 0x0000000000000103020
freeing 0x0000000000000002 mebibytes
allocating 0x0000000000000003 mebibytes
malloc returned 0x00000000000003020
freeing 0x0000000000000003 mebibytes
allocating 0x0000000000000004 mebibytes
malloc returned 0x0000000000000303030
freeing 0x0000000000000004 mebibytes
█
```

After implementing freepmem() into myxv6 I was able to see how much free memory I have. I was able to see the results by running free.c & memory-user.c. Where I could see the free memory being allocated when giving values while using the commands “free” and “memory - user”

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

```
omar@omar-VirtualBox:~$ cd myxv6/
omar@omar-VirtualBox:~/myxv6$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -
us=virtio-mmio-bus.0

xv6 kernel is booting

init: starting sh
$ ps
usertrap(): unexpected scause 0x000000000000000f pid=3
      sepc=0x00000000000012d0 stval=0x0000000000014008
panic: uvmunmap: not mapped
QEMU: Terminated
```

After modifying `sys_sbrk()` I was given two errors. One in the `usertrap()` and another with `uvmunmap()`.

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

```
omar@omar-VirtualBox:~/myxv6$ make qemu
riscv64-linux-gnu-gcc -Wall -Werror -O -f
o-pie -c -o kernel/trap.o kernel/trap.c
riscv64-linux-gnu-ld -z max-page-size=4096
kernel/spinlock.o kernel/string.o kernel/
nel/fs.o kernel/log.o kernel/sleeplock.o
riscv64-linux-gnu-objdump -S kernel/kerne
riscv64-linux-gnu-objdump -t kernel/kerne
qemu-system-riscv64 -machine virt -bios n
us=virtio-mmio-bus.0

xv6 kernel is booting

init: starting sh
$ ls
panic: uvmunmap: not mapped
█
```

I fixed the error in `usertrap()` by handling the fault codes that were causing the error. Those being the load and store faults. After fixing the faults I was still getting another error with the `uvmunmap()` call.

Task 4. Fix kernel panic and any other errors.

```
xv6 kernel is booting

init: starting sh
$ ls
.          1 1 1024
..         1 1 1024
README    2 2 2226
cat        2 3 23920
echo       2 4 22776
forktest   2 5 13496
grep       2 6 27080
init       2 7 23576
kill       2 8 22688
ln         2 9 22568
ls         2 10 26104
mkdir      2 11 22824
rm         2 12 22808
sh         2 13 40832
stressfs   2 14 23792
usertests  2 15 150520
grind      2 16 37312
wc         2 17 24912
zombie     2 18 22064
uptime     2 19 22200
time1      2 20 23168
matmul     2 21 23520
sleep      2 22 22488
time       2 23 23128
ps         2 24 24168
pexec      2 25 23248
free       2 26 22576
memory-user 2 27 23808
console    3 28 0
$
```

I was able to fix the `uvmunmap()` by going to where that panic call would happen in `vm.c` and just let it continue rather than panic, but `myxv6` was still not running commands. In order to fix these errors, instead of just correcting `uvmunmap()` panic error I had to do the same with `uvmcopy()`. Then I was able to do commands.

Task 5. Test your lazy memory allocation.

```
xv6 kernel is booting

init: starting sh
$ memory-user 1 24 6 &
$ allocating 0x0000000000000001 mebibytes
malloc returned 0x00000000000003010
freeing 0x0000000000000001 mebibytes
allocating 0x0000000000000007 mebibytes
malloc returned 0x00000000000103020
freeing 0x0000000000000007 mebibytes
allocating 0x000000000000000D mebibytes
malloc returned 0x00000000000803030
freeing 0x000000000000000D mebibytes
allocating 0x0000000000000013 mebibytes
malloc returned 0x0000000000203030
freeing 0x0000000000000013 mebibytes
█
```

```
$ free -k
130264
$ memory-user 64 100 10 &
allocating 0x0000000000000040 mebibytes
malloc returned 0x00000000000003010
$ freeing 0x0000000000000040 mebibytes
allocating 0x000000000000004A mebibytes
malloc returned 0x0000000004003020
freeing 0x000000000000004A mebibytes
allocating 0x0000000000000054 mebibytes
malloc returned 0x0000000003603020
freeing 0x0000000000000054 mebibytes
freeallocating 0x000000000000005E mebibytes
malloc returned 0x0000000002C03020
freeing 0x000000000000005E mebibytes
█
```

```
$ free
133390336
$ free -k
130264
$ memory-user 300 500 25 &
allocating 0x000000000000012C mebibytes
malloc returned 0x00000000000003010
$ freeing 0x000000000000012C mebibytes
allocating 0x0000000000000145 mebibytes
malloc returned 0x0000000012C03020
freeing 0x0000000000000145 mebibytes
allocating 0x000000000000015E mebibytes
malloc returned 0x0000000011303020
freeing 0x000000000000015E mebibytes
allocating 0x0000000000000177 mebibytes
malloc returned 0x000000000FA03020
freeing 0x0000000000000177 mebibytes
allocating 0x0000000000000190 mebibytes
malloc returned 0x000000000E103020
freeing 0x0000000000000190 mebibytes
allocating 0x00000000000001A9 mebibytes
malloc returned 0x000000000C803020
freeing 0x00000000000001A9 mebibytes
allocating 0x00000000000001C2 mebibytes
malloc returned 0x000000000AF03020
freeing 0x00000000000001C2 mebibytes
allocating 0x00000000000001DB mebibytes
malloc returned 0x0000000009603020
freeing 0x00000000000001DB mebibytes
allocating 0x00000000000001F4 mebibytes
malloc returned 0x0000000007D03020
freeing 0x00000000000001F4 mebibytes
free
133390336
$ free -k
130264
$
```

Here is how I tested the lazy allocation with different values using memory-user command.