

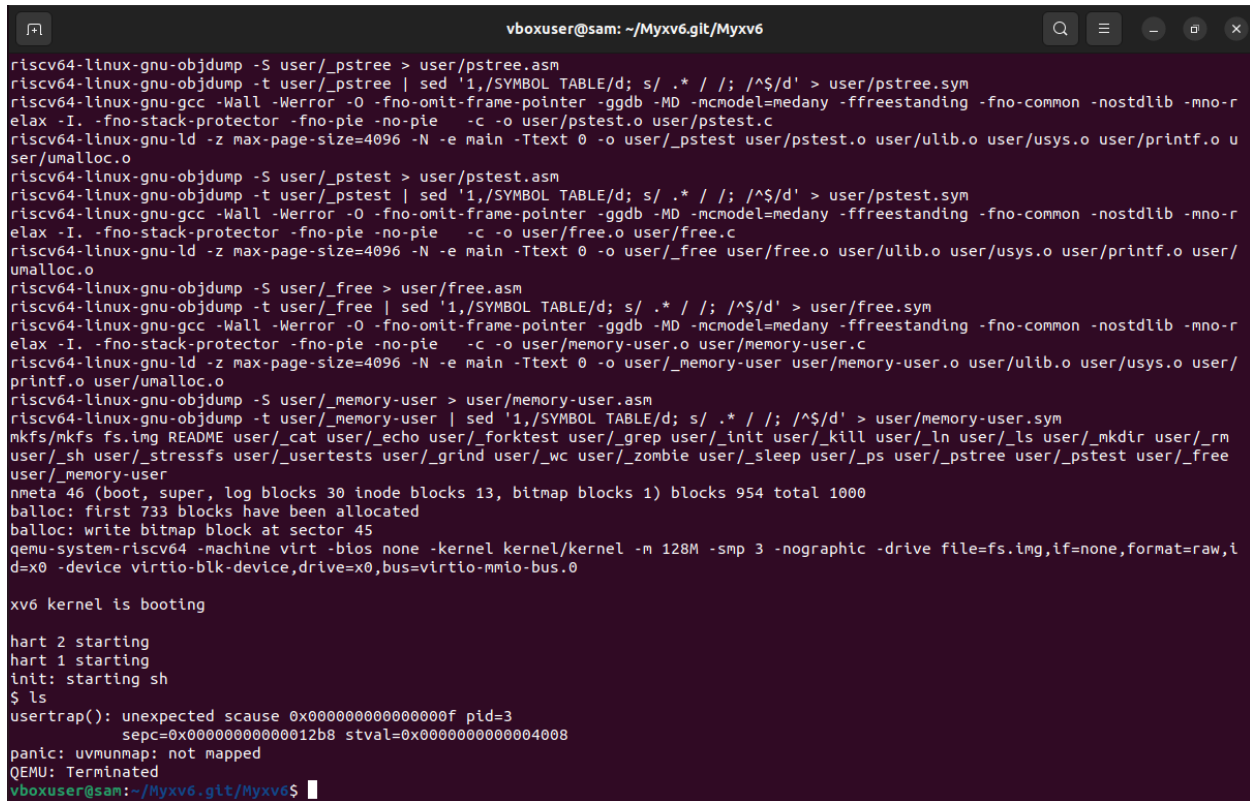
Task 1. freepmem() system call

```
vboxuser@sam: ~/Myxv6.git/Myxv6
forktest      2 5 13360
grep          2 6 26936
init          2 7 23432
kill          2 8 22552
ln            2 9 22408
ls            2 10 25960
mkdir         2 11 22672
rm            2 12 22664
sh            2 13 40672
stressfs      2 14 23640
usertests     2 15 150368
grind         2 16 37144
wc            2 17 24744
zombie        2 18 21920
sleep         2 19 22328
ps            2 20 23448
pstree        2 21 24440
pstest        2 22 23520
free          2 23 22416
memory-user   2 24 23664
console       3 25 0
$ free
4294967295
$ memory-user 1 5 1
allocating 0x0000000000000001 mebibytes
malloc returned 0x00000000000003010
freeing 0x0000000000000001 mebibytes
allocating 0x0000000000000002 mebibytes
malloc returned 0x00000000000103020
freeing 0x0000000000000002 mebibytes
allocating 0x0000000000000003 mebibytes
malloc returned 0x00000000000003020
freeing 0x0000000000000003 mebibytes
allocating 0x0000000000000004 mebibytes
malloc returned 0x00000000000303030
freeing 0x0000000000000004 mebibytes
allocating 0x0000000000000005 mebibytes
malloc returned 0x00000000000203030
freeing 0x0000000000000005 mebibytes
$
```

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

After implementing this task in which I change `sbrk ()` function to not allocate physical memory a `usertrap ()` error occurs. The `usertrap ()` indicates that something wrong happened in the user-level process. The `scause 0x...00f` is associated with the load and/or store fault exception. It might mean that the user level process was attempting to access memory that is not allocated yet, triggering the `usertrap ()`.

What I learned by doing this task is what the usertrap function does, why it appeared for this task. How sbrk() function works, how it's important for heap memory management and how to modify it in order to make it not allocate physical memory.



```
vboxuser@sam: ~/Myxv6.git/Myxv6
riscv64-linux-gnu-objdump -S user/_pctest > user/pctest.asm
riscv64-linux-gnu-objdump -t user/_pctest | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > user/pctest.sym
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -MD -mcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -c -o user/pctest.o user/pctest.c
riscv64-linux-gnu-ld -z max-page-size=4096 -N -e main -Ttext 0 -o user/_pctest user/pctest.o user/ulib.o user/usys.o user/printf.o user/_umalloc.o
riscv64-linux-gnu-objdump -S user/_free > user/free.asm
riscv64-linux-gnu-objdump -t user/_free | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > user/free.sym
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -MD -mcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -c -o user/_free.o user/_free.c
riscv64-linux-gnu-ld -z max-page-size=4096 -N -e main -Ttext 0 -o user/_memory-user user/_free.o user/ulib.o user/usys.o user/printf.o user/_umalloc.o
riscv64-linux-gnu-objdump -S user/_memory-user > user/memory-user.asm
riscv64-linux-gnu-objdump -t user/_memory-user | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > user/memory-user.sym
mkfs/mkfs fs.img README user/_cat user/_echo user/_forktest user/_grep user/_init user/_kill user/_ln user/_ls user/_mkdir user/_rm user/_sh user/_stressfs user/_ustests user/_grind user/_wc user/_zombie user/_sleep user/_ps user/_pctest user/_pctest user/_free user/_memory-user
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 954 total 1000
ballocc: first 733 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 -nographic -drive file=fs.img,if=none,format=raw,ld=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ ls
usertrap(): unexpected scause 0x000000000000000f pid=3
sepc=0x00000000000012b8 stval=0x0000000000004008
panic: uvmunmap: not mapped
QEMU: Terminated
vboxuser@sam: ~/Myxv6.git/Myxv6$
```

Figure 2: Results of running qemu after updating the sbrk() function. (Displays usertrap and panic: uvmunmap)

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

The panic uvmunmap still appears because there is still an error that is somewhere in the virtual memory system and it's related to the unmapping of the memory. There was probably an attempt to unmap some memory that hasn't been mapped.

In this task I learned how to handle errors (when panic triggers). How the usertrap function works and how it's responsible for handling exceptions and other system calls. Learned about load and store fault handling, how the process of handling exception occurs when a program in user mode tries to access memory.

One ocaused difficulties that I ran into while working on this task was trying to understand how to what cause and how to fix the usertrap that appeared. I had to go back and read how it worked.

```
vboxuser@sam: ~/Myxv6.git/Myxv6
riscv64-linux-gnu-objdump -S user/_pstree > user/pstree.asm
riscv64-linux-gnu-objdump -t user/_pstree | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -MD -mcmode
elax -I. -fno-stack-protector -fno-pie -no-pie -c -o user/pstest.o user/pstest.c
riscv64-linux-gnu-ld -z max-page-size=4096 -N -e main -Ttext 0 -o user/_pstest user
ser/unalloc.o
riscv64-linux-gnu-objdump -S user/_pstest > user/pstest.asm
riscv64-linux-gnu-objdump -t user/_pstest | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -MD -mcmode
elax -I. -fno-stack-protector -fno-pie -no-pie -c -o user/free.o user/free.c
riscv64-linux-gnu-ld -z max-page-size=4096 -N -e main -Ttext 0 -o user/_free user/f
unalloc.o
riscv64-linux-gnu-objdump -S user/_free > user/free.asm
riscv64-linux-gnu-objdump -t user/_free | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d'
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -MD -mcmode
elax -I. -fno-stack-protector -fno-pie -no-pie -c -o user/memory-user.o user/memo
riscv64-linux-gnu-ld -z max-page-size=4096 -N -e main -Ttext 0 -o user/_memory-user
printf.o user/unalloc.o
riscv64-linux-gnu-objdump -S user/_memory-user > user/memory-user.asm
riscv64-linux-gnu-objdump -t user/_memory-user | sed '1,/SYMBOL TABLE/d; s/ .* / /;
mkfs/mkfs fs.img README user/_cat user/_echo user/_forktest user/_grep user/_init u
user/_sh user/_stressfs user/_usertests user/_grind user/_wc user/_zombie user/_sle
user/_memory-user
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 954 t
ballocc: first 733 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 -
d=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting
hart 2 starting
hart 1 starting
init: starting sh
$ ls
panic: uvmunmap: not mapped
QEMU: Terminated
vboxuser@sam:~/Myxv6.git/Myxv6$
```

Figure 3: Results of running qemu after task 3 changes. (Displays panic: uvnumap)

Task 4. Fix kernel panic and any other errors.

The main file I changed was vm.c.

In this file I modified the uvmunmap and uvmcopy. In uvmunmap I modify it to check the mapping status before the function tries to unmap pages that are not mapped and also removed the panic for uvmunmap: not mapped.

In uvmcopy one of the changes was replacing some panics with continue statements in order when a page table entry wasn't found or if the page didn't exist it would continue instead of stopping.

I keep learning more on how virtual memory, page tables, memory mapping work and how I can modify them. Learned how uvmunmap unmaps virtual memory pages for given process for unmapped pages, non-leaf pages and other stuff. Learned that uvmcopy is for copying virtual memory pages from a page table to a new one.

For difficulties I really didn't know where to start in order to fix this panic, so it took me some time to complete this task because I was changing different functions and files. But I understood the kernel panic would have to be fixed in the vm.c file since I read and understood the most functions that are related to virtual memory management would be in this file.

```
vboxuser@sam: ~/Myxv6.git/Myxv6

hart 1 starting
hart 2 starting
init: starting sh
$ free
133390336
$ free -k
130264
$ free -m
127
$ memory-user 1 4 1 8
$ allocating 0x0000000000000001 mebibytes
malloc returned 0x0000000000003010
free
133349376
$ freeing 0x0000000000000001 mebibytes
free
133349376
$ allocating 0x0000000000000002 mebibytes
malloc returned 0x0000000000003020
free
133345280
$ freeing 0x0000000000000002 mebibytes
free
133345280
$ allocating 0x0000000000000003 mebibytes
malloc returned 0x0000000000003020
free
133345280
$ free
133345280
$ freeing 0x0000000000000003 mebibytes
free
133345280
$ allocating 0x0000000000000004 mebibytes
malloc returned 0x0000000000003030
free
133337088
$ freeing 0x0000000000000004 mebibytes
```

Figure 4: After fixing panic and other errors in task 4

Task 5. Test your lazy memory allocation.

Test case 1: allocating and freeing memory without touching it.

This test case verifies that memory allocation is deferred until it's used. We are checking if physical memory gets allocated only when it's necessary and making sure that the system is avoiding allocation when it's not needed. This testcase file contains the free.c code.

```
vboxuser@sam: ~/Myxv6.git/Myxv6$ 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-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ testcase1
133390336
$ testcase1 -k
130264
$ testcase1 -m
127
$
```

Figure 5: testing results for allocating and freeing memory without touching it.

Test case 2: allocating and touching memory.

The second test case is making sure there are no problems when memory is being allocated and being touched. We are making sure that the modified memory allocation functions correctly by handling the allocation and the usage of memory. The code is based on the memory-user.c.

```
vboxuser@sam: ~/Myxv6.git/Myxv6
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -MD -mcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -c -o user/testcase2.o user/testcase2.c
riscv64-linux-gnu-ld -z max-page-size=4096 -N -e main -Ttext 0 -o user/_testcase2 user/testcase2.o user/ulib.o user/usys.o user/printf.o user/umalloc.o
riscv64-linux-gnu-objdump -S user/_testcase2 > user/testcase2.asm
riscv64-linux-gnu-objdump -t user/_testcase2 | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > user/testcase2.sym
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -MD -mcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -c -o user/testcase3.o user/testcase3.c
riscv64-linux-gnu-ld -z max-page-size=4096 -N -e main -Ttext 0 -o user/_testcase3 user/testcase3.o user/ulib.o user/usys.o user/printf.o user/umalloc.o
riscv64-linux-gnu-objdump -S user/_testcase3 > user/testcase3.asm
riscv64-linux-gnu-objdump -t user/_testcase3 | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > user/testcase3.sym
mkfs/mkfs fs.img README user/_cat user/_echo user/_forktest user/_grep user/_init 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/_testcase1 user/_testcase2 user/_testcase3
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 954 total 1000
ballocc: first 806 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 -nographic -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ testcase2 1 4 1
allocating 0x0000000000000001 mebibytes
malloc returned 0x000000000000003010
freeing 0x0000000000000001 mebibytes
allocating 0x0000000000000002 mebibytes
malloc returned 0x00000000000000103020
freeing 0x0000000000000002 mebibytes
allocating 0x0000000000000003 mebibytes
malloc returned 0x000000000000003020
freeing 0x0000000000000003 mebibytes
allocating 0x0000000000000004 mebibytes
malloc returned 0x000000000000003030
freeing 0x0000000000000004 mebibytes
$
```

Figure 6: testing results for allocating and touching memory.

Test case 3: allocating memory and randomly touching just some pages.

The purpose of this last test case is to evaluate how the system handles allocating memory and the random touching of some of the pages. The test case is simulating scenarios where only some of the allocated memory is currently being used. We are verifying that our system is allocating physical memory dynamically and also allows selective access to allocated memory pages.

```
vboxuser@sam: ~/Myxv6.git/Myxv6
vboxuser@sam:~/Myxv6.git/Myxv6$ 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-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ testcase3
Usage: memory-user <start> <limit> <increment>, where <start> is initial mebibytes to allocate which is then incremented up to limit mebibytes
$ testcase3 1 3 1
allocating 1 mebibytes
malloc returned 0x0000000000003010
freeing 1 mebibytes
allocating 2 mebibytes
malloc returned 0x00000000000103020
freeing 2 mebibytes
allocating 3 mebibytes
malloc returned 0x0000000000003020
freeing 3 mebibytes
$
```

Figure 7: testing results for allocating memory and randomly touching just some pages.