

## HW 4: Lazy Allocation for xv6

### Task 1. freemem() system call

To implement the freemem() system call I modified the corresponding files needed to execute this call. I modified user.h by adding the user prototype of the function call, along with other of the files needed like, usys.p, syscall.h, syscall.c and of course sysproc.c. In sysproc.c I created the function call sys\_freemem which calls another function that I created called freeCount. This function is in kalloc.c and it just counts the amount of free memory by traversing the kmem.freelist. This was done in kalloc because in this file I had access to the linked list of memory. This was all that was needed to implement the freemem() system call.

In addition, I added the free.c and memory-user.c files to user folder provided and added them to the makefile in order to use these as commands. Then I tested the function system call by calling free various times. Then I also made sure that the memory-user command was also processed correctly. Below are the results from those testing process.

This task allowed me to understand not once again all the pieces needed to fully implement a function call but also how the commands free and memory-user require to know the amount of free memory. I now know that there is a linked list in charge of memory data and how we can easily count the amount that we currently have.

```
ashley@ashley-virtual-machine: ~/Documents
qemu-system-riscv64 -machine virt -bios none -kernel xv6.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=xv6.img,id=x1 -device virtio-net-device,net=none,id=x2 -device virtio-net-device,net=none,id=x3

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ free
133390336
$ memory-user 1 4 1 &
$ allocating 0x0000000000000001 mebibytes
malloc returned 0x0000000000003010
free
132300800
$ freeing 0x0000000000000001 mebibytes
free
132300800
$ allocating 0x0000000000000002 mebibytes
malloc returned 0x0000000000003020
free
130199552
$ freeing 0x0000000000000002 mebibytes
freeing 0x0000000000000003 mebibytes
malloc returned 0x0000000000003020
free
130199552
$ freeing 0x0000000000000003 mebibytes
free
130199552
$ allocating 0x0000000000000004 mebibytes
malloc returned 0x0000000000003030
free
125997056
$ freeing 0x0000000000000004 mebibytes
free
125997056
```

```
ashley@ashley-virtual-machine: ~/Documents
qemu-system-riscv64 -machine virt -bios none -kernel xv6.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=xv6.img,id=x1 -device virtio-net-device,net=none,id=x2 -device virtio-net-device,net=none,id=x3

xv6 kernel is booting

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

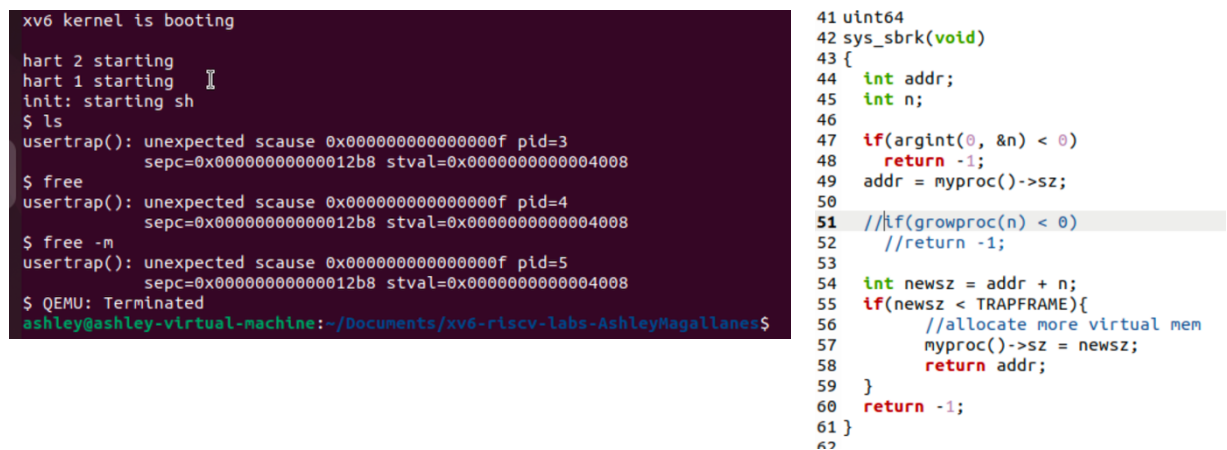
```
110
111 //calls the freeCount method implemented in kalloc.c
112 uint64
113 sys_freemem(void){
114     int freePages = freeCount() * 4096;
115     return freePages;
116 }
```

```
84 int
85 freeCount(void)
86 {
87     struct run *r;
88     int count = 0;
89     acquire(&kmem.lock);
90     for(r = kmem.freelist; r; r=r->next){
91         count = count + 1;
92     }
93     release(&kmem.lock);
94     return count;
95 }
```

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

For this task I had to modify the already provided method sbrk() which is located in sysproc.c. I first removed the if statement that checked the growproc(n) function since the goal was to not allocate the physical memory for the allocated virtual pages but instead allocate virtual memory space. I did this by as mentioned, removing this if statement and also adding a new size integer variable to hold the new size of the memory which was the original “old” size plus n. Then I checked that this new size was below the trapframe and if so then return the original size but also allocate more virtual memory. This was done by setting the sz to the new one.

When all of these modifications on the sbrk method were done I got a usertrap scause error when trying to run any xv6 command. This happened because the memory was now virtual space and not physical memory as before. Therefore, store and load faults were thrown. However, this error was fixed on task 3. These are the results of when I tried running commands with these modifications. In addition, there is a screenshot of the modified function.



```
xv6 kernel is booting
hart 2 starting
hart 1 starting
init: starting sh
$ ls
usertrap(): unexpected scause 0x000000000000000f pid=3
sepc=0x000000000000012b8 stval=0x0000000000004008
$ free
usertrap(): unexpected scause 0x000000000000000f pid=4
sepc=0x000000000000012b8 stval=0x0000000000004008
$ free -m
usertrap(): unexpected scause 0x000000000000000f pid=5
sepc=0x000000000000012b8 stval=0x0000000000004008
$ QEMU: Terminated
ashley@ashley-virtual-machine:~/Documents/xv6-riscv-labs-AshleyMagallanes$
```

```
41 uint64
42 sys_sbrk(void)
43 {
44     int addr;
45     int n;
46
47     if(argint(0, &n) < 0)
48         return -1;
49     addr = myproc()->sz;
50
51     //if(growproc(n) < 0)
52         //return -1;
53
54     int newsz = addr + n;
55     if(newsz < TRAPFRAME){
56         //allocate more virtual mem
57         myproc()->sz = newsz;
58         return addr;
59     }
60     return -1;
61 }
```

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

As mentioned above the previous task resulted in a scause error and this was due to load and store faults. Therefore, I looked at the RISC-V privileged ISA document and figured out that these exception codes were 13 and 15. All of these modifications were done in trap.c on the usertrap(). I also had to fix this by checking that the faulting address which is the stval register was within the process allocated virtual memory. This was done by using an if that checks that stval is less than the memory size.

Then, I allocated the physical memory by setting a void pointer to kalloc(), this function handles that allocation. If this allocation was done correctly then it installs the page table mapping for the virtual space page that contains the faulting address. I utilized mappages to achieve this. Mappages() maps virtual page to physical memory and inserts to the pagetable. After once again making sure that this mapping was done correctly meaning that a positive value is returned then everything was completed.

However, I handled some error cases, if the mapping wasn't done properly, then the allocated memory was freed and then the process was killed. Then if the allocation of the memory was not done successfully then the mapping wouldn't happen and once again the process would be killed. Lastly, if the faulting address wasn't valid the process would also get killed. I followed these error cases with a print statement that would let the user know what went wrong. Then exit the program at that time.

In this task I understood trap errors way more, on the previous homework assignments I had previously gotten these trap errors but didn't understand their purpose or meaning. Completing this section of the homework assignment allowed me to know how there trapframes work and in this case how they can be fixed through the kernel or supervisor mode.

This section was the more complex and difficult one in my opinion. I had a hard time because my memory wasn't being allocated properly however the issue was that I was checking the faulting address to be greater than or equal to the memory size, which was incorrect, but later this error was caught and the task was completed.

After this task was done when running any xv6 command I got a uvmunmap() panic error saying that memory wasn't mapped, which was fixed in task number four.

Below is the screenshot of my modified trap.c file and the result panic error.

```

70 //Homework 4 (task 3)
71 } else if(r_scause() == 13 || r_scause() == 15){
72 //checking if the faulting address (stval register) is valid
73 if(r_stval() < p->sz){
74 //printf("usertrap(): aaa\n");
75 //allocate physical frame memory
76 void *physical_mem = kalloc();
77 //if allocating memory was done correctly
78 if(physical_mem){
79 //maps virtual page to physical memory and inserts to pagetable
80 if(mappages(p->pagetable, PGROUNDOWN(r_stval()), PGSIZE,
81 (uint64)physical_mem, (PTE_R | PTE_W | PTE_X | PTE_U)) < 0){
82 kfree(physical_mem);
83 printf("mappages didn't work\n");
84 p->killed = 1;
85 exit(-1);
86 }
87 }
88 }else{
89 printf("usertrap(): no more memory\n");
90 p->killed = 1;
91 exit(-1);
92 }
93 }
94 }else{
95 printf("usertrap(): invalid memory address\n");
96 p->killed = 1;
97 exit(-1);
98 }
99 }
100 }
101 }

```

```

ashley@ashley-virtual-machine: ~/Documents/xv6-riscv-labs-AshleyMagallanes
riscv64-linux-gnu-gcc -Wall -Werror -fno-common -nostdlib -mno-relax -I. -c
riscv64-linux-gnu-ld -z max-page-size art.o kernel/console.o kernel/printf.o kernel/main.o kernel/vm.o kernel/proc.o kernel/sysproc.o kernel/bio.o kernel/errno.o kernel/sysfile.o kernel/riscv64-linux-gnu-objdump -S kernel/riscv64-linux-gnu-objdump -t kernel/ym qemu-system-riscv64 -machine virt -b=fs.img,if=none,format=raw,id=x0 -dev
xv6 kernel is booting
hart 1 starting
hart 2 starting
init: starting sh
$ free
panic: uvmunmap: not mapped

```

#### Task 4. Fix kernel panic and any other errors.

As mentioned in the section above task 3, would output a uvmunmap panic error. To fix this error and make sure that the program didn't keep expecting all virtual memory pages to be mapped I looked where this panic was happening. For this I used `grep -n "uvmunmap" *.c`

within the kernel directory to get the exact line position where the panic was thrown. This panic error was being thrown within the vm.c file exactly in line 178.

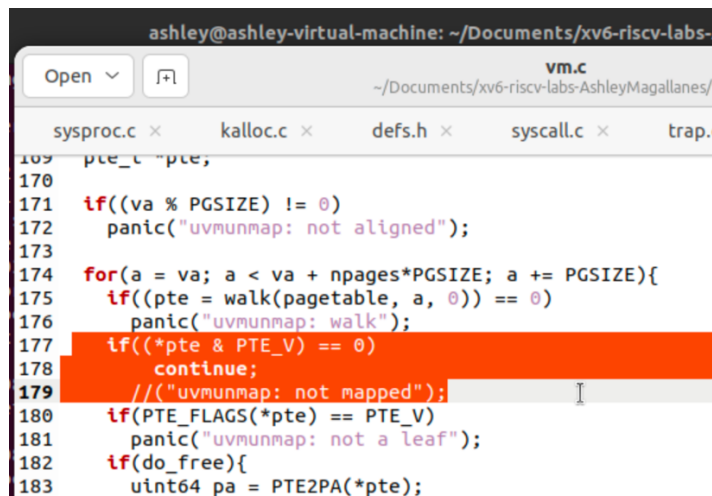
At first I tried fixing this by removing the whole if statement in charge of throwing the panic uvmunmap, however this just didn't work since it wouldn't allow me to run any xv6 command and it would return a trapframe error. Then I thought that only removing the panic print statement would work but that wouldn't fix nothing, since it would just mess with the booting of the qemu system. Therefore, I felt like it was not a bad track but if the statement was still needed then it needed to be "ignored."

The solution to fixing the uvmunmap panic issue was to add a "continue;" statement within this if statement throwing the panic. This would still allow the system to boot and all the xv6 commands to work properly, however the panic would be disregarded in a way. I didn't know that this "continue" statement was allowed in c language but it is and it is really useful.

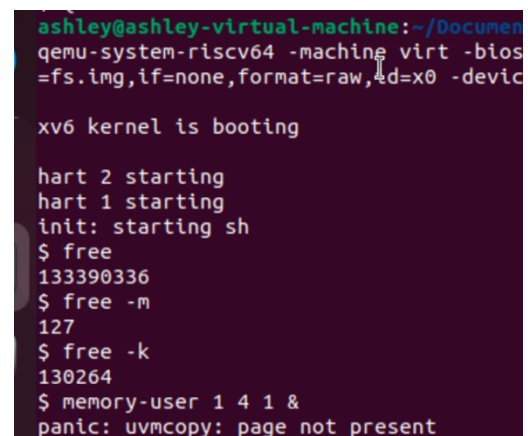
This task allowed me to gain knowledge not only on fixing panic issues but learning more about the c language syntax. I also now know how to use grep and how useful it can be! The most challenging part of this task was probably trying to fix the error by thinking about it instead of trying random things.

After, fixing this panic I got another panic only when trying to run the "memory-user" command and not all the xv6 commands as before. This panic error was now a uvmcopy, because the memory page was not present. However, this is normal for task 4 completion and was fixed in the following task.

These are some results and modifications done:



```
ashley@ashley-virtual-machine: ~/Documents/xv6-riscv-labs-
Open  [icon]  vm.c
~/Documents/xv6-riscv-labs-AshleyMagallanes/
sysproc.c x  kalloc.c x  defs.h x  syscall.c x  trap.c
109 pte_t *pte;
170
171 if((va % PGSIZE) != 0)
172     panic("uvmunmap: not aligned");
173
174 for(a = va; a < va + npages*PGSIZE; a += PGSIZE){
175     if((pte = walk(pagetable, a, 0)) == 0)
176         panic("uvmunmap: walk");
177     if((*pte & PTE_V) == 0)
178         continue;
179     //("uvmunmap: not mapped");
180     if(PTE_FLAGS(*pte) == PTE_V)
181         panic("uvmunmap: not a leaf");
182     if(do_free){
183         uint64 pa = PTE2PA(*pte);
```



```
ashley@ashley-virtual-machine: ~/Documents
qemu-system-riscv64 -machine virt -bios
=fs.img,if=none,format=raw,d=x0 -device

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ free
133390336
$ free -m
127
$ free -k
130264
$ memory-user 1 4 1 &
panic: uvmcopy: page not present
```

### Task 5. Test your lazy memory allocation.

As mentioned above the result from fixing the uvmunmap panic error was another panic error, this time it was uvmcopy. The fixing of this panic was the same as the previous. Instead, this time I used `grep -n "uvmcopy" *.c`. This led me to line 313 and the process was exactly the same since I only removed the panic line and replaced it with a `continue;` statement.

This task wasn't as complicated since I already had the experience from the previous task. The result from fixing this panic was that the memory-user was now fully working. The screenshots below showcase of the test samples that I did. I started with the original sample

given, which was the one used on task 1. The sample was “memory-user 1 4 1 &.” Then I also attempted to have a higher increment by 10. And lastly I tried doing each parameter times ten to see the result. The first two sample cases worked as they should and the last one failed. I assume this is normal. Also these cases are done by touching and allocating the memory. The last case I also tried 1 4 1 & but it was done not touching but allocating the memory. The “touching” was done in memory-user and it relied on a for loop that traverse the pages.

```
ashley@ashley-virtual-machine: ~/Documents/xv6-riscv64
riscv64-linux-gnu-objdump -S kernel/kernel > kernel/kernel.asm
riscv64-linux-gnu-objdump -t kernel/kernel | sed '1,/SYMBOL:
ym
qemu-system-riscv64 -machine virt -bios none -kernel kernel/
=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,d
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ free
133390336
$ free -m
127
$ free -k
130264
$ memory-user 1 4 1 &
$ allocating 0x0000000000000001 mebibytes
malloc returned 0x00000000000003010
ffreeing 0x0000000000000001 mebibytes
free
132300800
$ free
132300800
$ allocating 0x0000000000000002 mebibytes
malloc returned 0x00000000000003020
free
130199552
$ freeing 0x0000000000000002 mebibytes
free
130199552
$ allocating 0x0000000000000003 mebibytes
malloc returned 0x00000000000003020
free
130199552
$ freeing 0x0000000000000003 mebibytes

302 uvmcopy(pagetable_t old, pagetable_t new, uint64 sz)
303 {
304     pte_t *pte;
305     uint64 pa, i;
306     uint flags;
307     char *mem;
308
309     for(i = 0; i < sz; i += PGSIZE){
310         if((pte = walk(old, i, 0)) == 0)
311             panic("uvmcopy: pte should exist");
312         if((*pte & PTE_V) == 0)
313             continue;
314         //panic("uvmcopy: page not present");
315         pa = PTE2PA(*pte);
316         flags = PTE_FLAGS(*pte);
```

```
ashley@ashley-virtual-machine: ~/Documents/xv6-riscv64
qemu-system-riscv64 -machine virt -bios none -kernel kernel/
=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,d
xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ memory-user 1 4 10 &
$ allocating 0x0000000000000001 mebibytes
malloc returned 0x00000000000003010
free
132300800
$ freeing 0x0000000000000001 mebibytes
free
132300800
$ allocating 0x0000000000000002 mebibytes
malloc returned 0x00000000000003020
free
130199552
$ freeing 0x0000000000000002 mebibytes
free
130199552
$ allocating 0x0000000000000003 mebibytes
malloc returned 0x00000000000003020
free
130199552
$ freeing 0x0000000000000003 mebibytes
```

```
xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ memory-user 1 4 1 &
$ allocating 0x0000000000000001 mebibytes
malloc returned 0x00000000000003010
free
133349376
$ freeing 0x0000000000000001 mebibytes
free
133349376
$ allocating 0x0000000000000002 mebibytes
malloc returned 0x00000000000003020
free
133345280
$ freeing 0x0000000000000002 mebibytes
frww
exec frww failed
$ free
133345280
$ allocating 0x0000000000000003 mebibytes
malloc returned 0x00000000000003020
free
133345280
$ freeing 0x0000000000000003 mebibytes
```

Extra Credit Task 6. Enable use of the entire virtual address space.

List what files you changed and describe the changes.

Summarize what you learned by carrying out this task.

Extra Credit Task 7. Allow a process to turn memory overcommitment on and off.

Describe your approach to this task. Commit and push your code changes to your hw4 branch.  
Show the results of a test program that turns memory overcommitment on and off.  
Summarize what you learned by carrying out this task.