Lab 2

User Addresses!!!

- When dealing with user addresses, make sure you are offsetting from main_memory
- This is a quite common bug and will result in segfaults
- Use `main_memory[pcb->base + user_address]`
    - This gets the byte (char) value at that user address
    - This does not get a pointer
    - This does not get anything else but the char
- If you want to interpret the address as something else, cast it like this

```
char** pointerarray = (char**) &main_memory[pcb->base + user_address]
```

UC SANTA BARBARA

Please read the man pages! I'll go over each of them because people are still confused

- I'm gonna go over these calls
- Ioctl
- Fstat
- Exec
- Fork
- Wait
- Exit
- The other ones are pretty trivial
- **Just remember that for any user address you are accessing, you want to make sure its in bounds**

ioctl(fd, request, user_mem_address)

Check the fd
Check the request
User_mem_address is an user address so convert it to a system address and use ioctl_console_fill

fstat(fd, user_mem_address)

Check the fd
User_mem_address is an user address so convert it to a system address and use stat_buf_fill

Exec - replaces entire program with new one

Remember argv is a char**
- An user array filled up with user_addresses that point to strings

Save the args
PerformExecve:
    Loads user program
    Sets the registers (StackReg, brk)
    Calls moveargs, initcruntime
Free the args

Fork - makes a new program that's a mirror of current program

- Needs to have a new memory block
- Needs a new pcb
- Needs to have a new pid
- Needs to have a new base
- Sets parent pcb
- All the other registers are the same
- Modifies parents children
- Copies entire memory block to new block
- Adds both pcbs to readyq

Wait

Only argument is a useraddress to wstatus variable

- Check if there are even any children
- Blocks and waits for a child to call it
- Delete a child from my waiters list
- Set wstatus address to be the exitval of the child
- Clean up the child

Exit

Multiple things to do here

- Clean up the memory block
- Save the exitvalue in the pcb
- Send all children to INIT
- Clean up all the zombies (Two ways here)
- Modify parents children to remove current pcb
- Clean up current pcb if parent is INIT
- Otherwise add current pcb to parents waiters

Quick note on input if you want it to be like linux

https://sites.cs.ucsb.edu/~rich/class/cs170/labs/linux-testing.html

- KOS doesn't flush input on enter so running cat80 won't have the same behavior as it does in linux
- KOS as a flag -t, that will set a global variable isTTY to be true
- This will send a [\n,-2] on an enter, so you can treat it as a end of input signal
- But also make sure you handle multiple -2's because there are multiple -2's sent

Questions/office hours

Exec

Exec.c:

```
    char *path = "argtest";
    char *argv[10] = { "argtest", "Once", "I", "Had", "a", "Girl",
    "on", "Rocky", "Top", NULL};

    printf("printf 0x%x, 0x%x %s\n", argv, argv[0], argv[0]);
    execve(path, argv);
```

Exec inputs:

```
int execve(const char *pathname, char *const argv[]);

What is the type of pcb->registers[5]?
What is the type of pcb->registers[6]?
If main_memory started at 0x0
   pcb->registers[5] = 80,
   pcb->base = 40?

     What is the value of pathname?
     At what address is the actual pathname?

Let pcb->registers[6] = 72
     What is the value of argv?
     What is the type of argv[0]?
     If argv[1] = 96, where is “Once” located?
     What is the value of argv[9]?
```

Fork

How many processes return from fork?
Fix the following code for do_fork()?

```
void SysCallReturn(struct PCB_struct *pcb, int return_val) {
       pcb->registers[PCReg] = pcb->registers[NextPCReg];
       pcb->registers[2] = return_val;
       dll_append(readyq, new_jval_v(pcb));
       kt_exit();
}

void *do_fork(void *arg){
       struct PCB_struct* pcb = (struct PCB_struct*) arg;
       //whats missing here
       struct PCB_struct* newpcb = malloc(sizeof(struct PCB_struct));
       //pcb only has base, limit, brk, pid, pPCB so far
       newpcb->base = pcb->base;
       newpcb->limit = pcb->limit;
       newpcb->brk = pcb->brk;
       newpcb->pid = pcb->pid;
       newpcb->pPCB = pcb;
       for (int i = 0; i < NumTotalRegs; i++) {
              newpcb->registers[i] = pcb->registers[i];
       }
       memcpy(newpcb->base, pcb->base, pcb->limit);
       SysCallReturn(newpcb, 0);
       SysCallReturn(pcb, newpcb->pid);
}
```

Wait

Whats a zombie? Who cleans up after a zombie?
What is an orphan? Who cleans up after an orphan?

Consider the following:
Init is initialized
    Who is Init's parent?
proc1 is initialized
    Who is proc1's parent?
proc1 calls fork(), so proc2 is initialized
    Who is proc2's parent?
proc2 calls fork() twice, so proc3 and proc 4 is initialized
    Who is proc3's parent?
    Who is proc4's parent?
proc4 calls exit()
    Who and when is it going to get cleaned up?
proc2 calls exit()
    Does anyone clean up proc2 yet?
    Who is proc3's parent?
proc1 calls wait()
    Who does proc1 clean up in this call?
proc1 calls exit()
    Who cleans up proc1?
proc3 calls exit()
    Who cleans up proc3?