OS161 DVZ Progress report 1

I. SYSTEM CALL PROGRESS

__getcwd : 4 | __chdir : 4 | execv : 2 | _exit : 5 | Read : 5 | Write : 5 | Open : 5 | Close : 5 | Fork : 3 | Getpid: 5 | Waitpid: 2 | Lseek: 4 | Dup2: 4

II. SYSCALLS NOT SCORED 4

- execv (Eric): Do I use the run program function inside runprogram.c? Will I need to update the PCB? Do I need to switch the context involving curthread->t_context? When I change out the current process, do I need to destroy the old per proc file table that is associated with it?
  This involves copying in the program string and loading the executable with the loadelf function similar to runprogram.c.
- Fork (John): Why is it necessary to assign the address space outside of the fork handler but the trapframe inside the fork handler? What synchronization method should be used inside the fork handler? I see that splhigh() might be an option but I'm not sure. Should I release the fork lock before or after the thread fork? The bulk of the work is already done as I copy all of the data needed to start the new process as a clone of the current one, combine it into a PCB, and then start the new process with thread_fork, inside which I edit the trapframe and activate the address space before transitioning to user mode.
- Waitpid (Emma): How does a parent determine whether or not to wait for the child? How to check and handle invalid status? The idea would be to scan through the exitcode and signal the cv wait, Post exitcode, signal cv and parent wakeup and check through exitcode, if found clear from table and respond to exitcode and continue running, move on the next wait, when parent wake up did it find the exitcode, similarly to how cv work,if not found then continue to sleep, probably do in while loop, check if exited, sleep only neecessary

III. INDIVIDUAL GROUP MEMBER QUESTIONS

- Emma-
  - I learned how the flow of calling the syscalls from the dispatcher and how to store arguments in corresponding registers. I also learned more about how the VFS connects with the file structure as I implement file system calls lseek and dup2.
  - One thing I wished I knew more about is how the parent process can wait for child process to exit because that's the key of waitpid. I also watched YouTube videos explaining waitpid in Unix system and how there's different kind of waiting happening in waitpid, non-blocking (polling) - the WNOHANG status and blocking from parents which I will need to figure out in waitpid for os161. I also need to figure out how the PCB will interact when calling waitpid because as I look at the test cases for bad_waitpid there's a lot of error checking edge cases I need to worry about
  - Difficulties I encountered was from not having concrete backend structured because as we move along implementing system calls, a lot of our Backend had to change and it's hard to keep track since we don't have that concrete and laid out in the design doc, we should have done it more systematically. Another difficulty arised is for me to understand how dup2 works because lseek is relatively straightforward for me but dup2 involves having another pointer

pointing to the same file and adding the reference count which is harder to visualize in terms of how it interacts with the file tables than I imagined
    - Check in global table, and dup to proc table. Difference between copy and duplicate, copy has own entries and seek pos, duplicate has same seek pos in all file, seek pos to a pointer instead of value, seek pos all the same, copy from pos, new pointer from new struct, copy.
    - Is dup2 dup from local table or to local table?
- Eric-
    - I learned more about the global variables that I can use to get important information such as curproc and curthread. I also learned about the UIO block which is key in moving stuff from userspace to kernel space. I also learned about the vfs functions which is helpful since they accomplish many of the things needed there.
    - One thing I wish I knew more about OS161 that I haven't figured out yet is the way registers work for context switching since I need that for execv. I feel that I would need those in order to have a program execute properly especially for an environment.
    - I encountered many difficulties while implementing the project. Most of it involved trying to figure out how the web of files interacted with each other and how I needed that in order to correctly implement many of the functions. I had to ask a lot of questions in both office hours and discuss with my groupmates in order to steer myself in the right direction and will continue to do so as I continue to work on implementing the project.
- John -
    - I learned much more about the VFS and accompanying UIO systems in writing my syscalls, and so far I've learned a lot about the proc and thread subsystems as well working with fork. If it counts, I also learned a lot about the copyin/copyout subsystems and the syscall handler.
    - One thing I wish I knew better about OS/161 is the address space subsystem, it would help a lot with showing me the best way to copy/use/destroy them appropriately. I know most of this but want to know more.
    - I had a lot of trouble understanding exactly how to use the copyin/copyout and UIO things at first and it took me a while to figure those out. I also had trouble with making the file and proc table searching work at first and testing the tables, because bugs with these would usually mess everything up. Starting from nothing was the hardest part.

IV. TEAMWORK ON GIT

Yes John works on main branch, Emma and Eric work on feature branch and we follow the branch-name convention for easy merge and debug. When we merge we sit together to resolve conflicts and compile and boot the kernel after merging. This collaboration style worked very well for the most part and merges worked without much issue since we agreed what and what not to do beforehand.

V. DESIGN DOC REFLECTION

- **What was easier to accomplish than you thought it was going to be, and what was harder?**
    - Emma: I thought handling the backend was harder than I thought because initially in our design doc we didn't specify in detailed how the tables and file structure should be made so we changed a lot when we started implementing the system calls. Handling the off_t type to cast to 32 bits for syscall dispatcher was easier than I thought it would be after talking to Matt in Office Hours and ask about how copyin, join32to64 and split64to32 work
    - John: Getting open and close to work and handling errors in the syscall handler were much easier than I expected them to be, but one surprisingly hard thing for me was getting printing to and reading from the console to work correctly, there were so many small error conditions.
    - The backend stuff was pretty easy. Since it was us who was writing it, we knew the ins and outs. Working with the syscalls for me were much more difficult because of how I had to start digging through to see what other functions existed in os161 and how to interact with it.
- **Did your design change over the course of the implementation? If so, briefly, how?**
    - Emma: The design of dup2 change since the file structures and file tables changed significantly because now we have duplicate function besides copy file function and the seek position is a pointer and we have local_fd for each file along with the global_fd
    - John: The design for read and write changed significantly when I had to deal with everything surrounding read and write interacting with the console, and the initial file struct I created also changed a lot. Read and write initially used kprintf and kgets, neither of which worked very well, so I switched to using getch and putch loops. The File struct now contains much more extra info to make handling files easier.
    - Eric: The file structures needed to be updated to accommodate for newer needed functionality that we would need in order to make the syscalls work.

VI. WORKFLOW GOING FORWARD

Going forward I think we'll largely keep the same workflow in terms of branching, meeting together, and working with git, but going forward John will start to use feature branches too because main should always be reliable and at times this hasn't been the case with our current setup. Otherwise we work very well together and this smooth workflow should be maintained.

VII. TIMELINE

In terms of a timeline, we hope to have waitpid and exec at least partially implemented by Sunday and the rest of our functions done by that point so we can spend the last week tightening up our implementation and implementing our equitable shortest job first scheduler.