CSC-335: Project 3 Satus Report 1

Hope Crisafi, Tyler Nass, Zach Dubinsky

Professor Anderson

Wednesday, May 9, 2023

Major Subsystem	Owner	Progress (1-5)
Scheduler	Tyler	1
Process Table	Hope	5
File Table	Zach	5

Syscall	Owner	Progress (1-5)
getpid	Tyler	5
fork	Hope	4
execv	Zach	1
waitpid	Tyler	4?
_exit	Hope	5
open	Zach	4
read	Zach	4
write	Zach	4
lseek	Hope	1
close	Zach	4
dup2	Tyler	5
chdir	Tyler	5
__getcwd	Hope	1

- execv
    - Specific Questions
        - What does it mean to load a program image onto a process?
        - Does execv call fork?
        - Is a new process created or is the current process modified?
    - Approach
        - Did not work on this system call at all.
        - Focused entirely on the file table and related calls to test the file table.
        - Will pursue the same approach for written syscalls: reading documentation to understand the goal of the syscall and implementing to accomplish this goal.
    - Group Members
        - Zach
- Scheduler
    - Specific q's
        - What exactly does our planned implementation of priority-based round robin scheduling entail?
        - Our priority is going to be (at least partially) based on the amount of time a process is waiting - how do we keep track of time in C/in OS161?
    - Approach
        - Haven't gotten to working on this subsystem yet. I will go about it by examining the basic version of the scheduler that is in OS161 to understand how it works & then I will figure out how to implement our planned scheduling system, a priority-based round robin scheduler.
    - Group members
        - Tyler
- lseek
    - Specific q's
        - Could I potentially get a quick diagram of how this function behaves throughout its execution? I understand what it is supposed to do, but need help on how I should do it (note: have not started implementing it yet)
    - Approach
        - Use synchronization mechanisms so it can remain atomic. Use KASSERTs to check the validity of the input before performing any calculations. I will then use a switch case (?) to determine where the file pointer should be moved to, based on the input.
    - Group members
        - Hope
- __getcwd
    - Specific q's

        - How should I interface with the buffer in a robust manner? How is this getting the CWD by "computing" it?
    - Approach
        - Use synchronization mechanisms so it can remain atomic. Use KASSERTs to check the validity of the input before performing any calculations. I will access the buffer via the input pointer, and then use the buflen to calculate the current directory. The length of the CWD will be stored in the buffer.
    - Group members
        - Hope

- **Each team member should answer the following:**
    - **What subsystems of OS/161 did you learn more about while working on your project?**

Zach: I learned more about the file system, and how to protect bad access to system resources. In particular, there are many reserved system constants such as the console read and write file ids that we have to account for when writing anything. Moreover, some file names are improperly formatted and ensuring that there are no *bad* accesses to the system is a priority.

I also had to work in iterative fashion because, as is the case with a major subsystem, it is not always clear when and where certain computation should be performed. In the case of *dup2*, I had to work with Tyler to make sure that the file table was protected but the system call was able to perform its desired function.

Hope: I worked with a lot of functions that involved the process (proctable, fork, exit). By writing these and learning the other modules they interface with helped me understand how they function as a part of the OS. I feel like I really understand all aspects and behaviors of a process now, and how the process interacts with syscalls.

Tyler: Probably the process table. The syscalls I implemented relied on the process table. Especially in how PIDs are assigned to processes, as I helped in the development of the process table, specifically in how it assigns PIDs to new processes.

    - **What is one thing that you wished you knew about OS/161 that you haven't figured out yet?**

Zach: I still don't fully understand the function of execv, and how to set read/write file permissions. I suspect this has to do with the *mode* parameter of the *open* syscall which I do not have used.

Hope: I am still a little confused on how fork() is supposed to work implementation-wise (in theory it all makes sense). I also have a hard time understanding how I am supposed to use the preexisting OS161 code in the context of the functions I write. Eventually I figure it out, but it's a learning curve.

Tyler: How the provided scheduler works in OS161. I have not yet started working on the scheduler, so knowledge of how it works would probably be helpful for my work in the rest of the project.

- **What difficulties did you encounter while implementing your project?**

Zach: Starting from such a blank canvas proved to be a challenge in this project. As I mentioned earlier, there was a great deal of coming back to and changing the file table because it did not align with the function of a syscall. Especially as our group learned more about the implementation of each syscall. Debugging proved to be a major challenge as OS/161 is picky about changes: the addition of four lines of code was enough cause for an hour of debugging in some cases!

Hope: Understanding some of the errors that occur, as a lot of them are non-descriptive and unhelpful in pinpointing the problem. A lot of the time it came down to guessing and checking, and brute-force debugging, which is a slow and tedious process. Also, there were a lot of false dawns in terms of having things completed. I thought I had my proctable completed 3 times, and each time I had to go back and change something considerable.

Tyler: The fact that certain syscalls are hard to test/cannot be tested without having other syscalls already implemented. Trying to implement waitpid when exit wasn't fully implemented was difficult—and I imagine trying to implement exit while I hadn't fully implemented waitpid was also difficult. Getting through this required regular communication between team members to make sure we were always on the same page.

- **How was your group's teamwork?**

Zach: We were able to collaborate effectively with appropriately delegated roles and system calls. When we struggled, we leaned on each other for insight into our independent implementations. When a subsystem and a syscall were not in alignment, one group member would adjust so we could integrate the code. With our work portioned for each individual's success, and with clear communication about progress, we were able to connect independently-developed components almost seamlessly.

Hope: Very good. We have very good communication and leadership. I did not feel burdened to complete my other groupmates' tasks, which took some worry off my chest and allowed me to fully focus on my own tasks.

Tyler: I think we have collaborated very well so far. We each had our own assigned work to do, but we also didn't strictly keep ourselves from putting some work into each other's pieces when it was appropriate to do so. For example, Zach implemented a function that greatly simplified what needed to be in dup2 & I helped with designing some aspects of the process table when it was appropriate.

- **Did you collaborate effectively using git?**

Zach: Our git strategy had no issues. We emphasized frequent commits and merging so all change was meticulously tracked, and could be restored when new bugs appeared. Honestly, we might have even overdone it. I can count no more than five merge conflicts faced among our group during the course of the week.
Hope: Yes, we used git to the fullest extent and effectively. I learned a lot (I am still relatively new to git), and Zach is a Git master.
Tyler: Our use of git has been very effective. We worked on separate branches and also usually worked on separate files, meaning there were few merge conflicts and the ones we did have were not very major.

- **What was easier to accomplish than you thought it was going to be, and what was harder?**

So far, nothing has been easier than expected to implement. Mostly everything has been as expected. However, the data structures (proctable and filetable) were particularly challenging. Throughout the process we made several revisions and encountered new errors while developing the rest of OS/161. Additionally, we have had difficulty implementing execv, waitpid, and fork. They are challenging to implement in their own right, and that we need more than one of these big pieces to test any of them is an inherently hard task.

- **Did your design change over the course of the implementation? If so, briefly, how?**

Not much. The only thing that changed is we decided against creating a PCB data structure, in favor of maintaining process information in the process struct. However, our first draft implementations of each major subsystem were changed as we added new features. These changes streamlined the internals of the subsystem but did not change our overarching design.

- **What are you going to do differently as you complete the implementation? Answer in terms of your design, group dynamic, and workflow.**

The current development process we currently have in place seems to be working well for us, so there is nothing that we really plan on changing. We will continue to have group meetings to strategize and touch base with each other. We all complete our work in the pasta lab, often all at the same time. This enables us to communicate with each other for git workflow, and is good to ask each other questions as one of us implements code another wrote.

- **Come up with a specific timeline for completing the remainder of the implementation.**

Week 9 we should have at least 2 group meetings and pasta work sessions. This will allow us to progress on our implementations in a relatively low-stress environment, and we will not be directly pressed by due dates. Week 10 we will likely be meeting in the pasta lab nightly to all work on our implementations at the same time and collaborate. Our goal will be to have all of our individual implementations done two days before the final due date, so we can ask lingering questions and do some clean-up.

Component	Task	Deadline	Owner
Scheduler	Implement	Final deadline	Tyler
fork	Debug	Final deadline	Hope
execv	Implement	Beginning of wk. 9	Zach
waitpid	Test	Asap	Tyler
open	R/W-only permssions	Final Deadline	Zach
read	R/W-only permssions	Final Deadline	Zach
write	R/W-only permssions	Final Deadline	Zach
lseek	Implement	Final Deadline	Hope
close	R/W-only permssions	Final Deadline	Zach
__getcwd	Implement	Final Deadline	Hope