Recitation 4: UNIX Part 2

We cover UNIX in two parts.This part focuses on processes and the shell; the first part focused on naming and the filesystem.

Relation to lecture

- UNIX paper has lots of connections with lecture: kernel, hierarchical organization (of processes in this case) to achieve modularity, naming, virtual memory to enforce modularity, process communication (in UNIX this happens via pipes, but it is conceptually similar to bounded buffers)

Process

- Organized hierarchically ("parent" process, "child" process(es))
- Communicate via pipes
- Fork spawns a child process. Child has exact copy of the parent, but in its own virtual address space
- Execute replaces the current running code with the content of the file

Shell

- Shell = commandline interface for accessing system calls
- A basic shell: read user input, fork a process, execute it, wait for it to finish, and go again.
    - We fork before executing; if we skipped fork, execute would simply replace the shell (and then the shell would be gone once execute finished)
    - Bypass the "wait" step to run a process in the background
    - You can implement this yourself in Python!
- Shell always creates a new process.

Discussion

- How does "everything is a file" differ from previous designs?
- UNIX was designed for programmers, by programmers. Who was a programmer in this context? How does this affect the way we use computers today?
- Why isn't inode information in the directory? Why is file position maintained in the kernel?
- UNIX is a "monolithic" kernel — what does that mean?
- How did the UNIX developers envision their system being used in 2023, if at all? What would the world be like if Multics had become the dominating operating system instead?