Operating Systems
Assignment 1

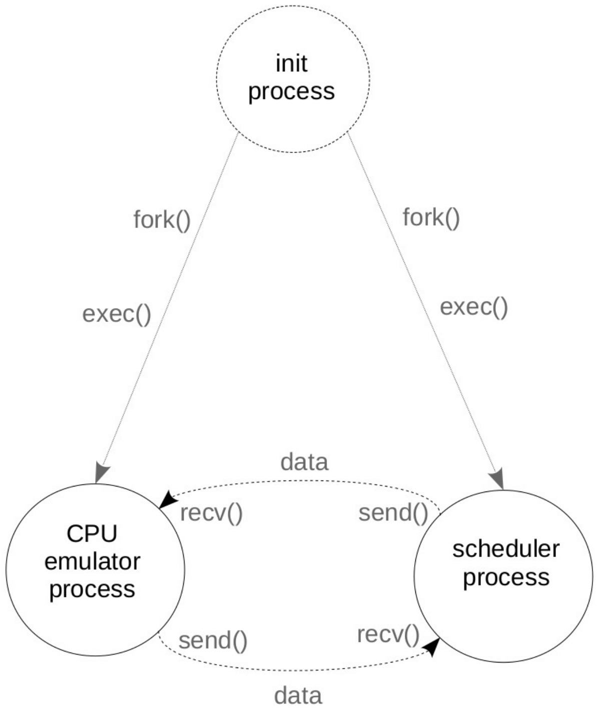
Objectives:

PID 1 (init), process creation via fork-exec, process control blocks (PCBs), inter-process communication (IPC), basic FIFO scheduling.

Task: CPU Scheduling Simulation, Part I:

For this assignment, we will simulate the execution of processes on a single CPU core.

To do this, we will use two processes: one that acts as a CPU emulator, and one that acts as a scheduler. The scheduler will maintain a queue of processes that are ready to run. It will pop a process off the front of the queue, and send it to the CPU emulator. The CPU emulator will run the process for a fixed period of time, then stop and return the updated PCB information to the scheduler (this is *round-robin* scheduling). The scheduler will then append the process to the back of the queue (if it is not finished), pop the next process from the front of the queue, and send to the CPU. This will repeat until all processes have completed.



We will start up these processes by using an *init* program - much like the init process that starts up all of the OS's services during boot, but much simpler. This init program will simply start both of the two child programs, then wait for them to finish, and close.

We will then need two-way communication between these two processes in order to send the PCB information back and forth.

Later in the semester, we will re-use the software we built for this assignment to experiment with different CPU scheduling algorithms (these can be seen in Chapter 6).

For now, we will just build the infrastructure to fork two processes, and pass messages back and forth between them.

What you need to do:

Write the init program in C, which will fork off the two child processes.

Decide how to implement the two-way IPC. You can use a TCP socket, a two-way pipe (mkfifo), shared memory*, or two one-way pipes*. Think about why you chose one way over the other, and include a comment about it in your summary write-up. (*you would probably need to use the init program as the CPU emulator for these methods to work).

Implement the CPU emulator, which simply receives PCB information, decrements the number of remaining CPU cycles of the PCB by a fixed amount, and sends it back to the scheduler.

Implement the scheduler, which reads process data from a file, adds the processes to a queue, then pops the front, sends it to the CPU, receives process data back from the CPU, and appends the process to the back of the queue (if it has not completed yet).

Notes:

You can write the CPU emulator and the scheduler in any language you want. But the *init* process must be written in C, and must use *fork*() and *exec*().

For now, we will use *blocking IPC* - which means your program will stop when waiting for an incoming message. Later, we'll look at *non-blocking IPC*, which is a little more complicated.

What to turn in:

1. A (breif) summary PDF, explaining your approach and the choices you made
   for language and IPC method and why you chose them over other alternatives.
   This should be approximately one page max - just summarize your choices.
2. The output of your CPU emulator - *included as an Appendix in your pdf*.
3. The source code of three files: init.c, your CPU emulator, and scheduler.

<u>This should be four (4) files total</u>:

summary.pdf
init.c
cpu_emulator.{x}
scheduler.{x}

(where {x} is the programming language you chose to use).

<u>Zip files and other compressed folders will receive zero credit</u>. Just attach your four files to the email.