Commander process

What you need to do for the first part

Before describing the project

The job of the CPU is to fetch, decode, execute an instruction and continue these steps. Imagine we have one processor and many processes such as email, word, C compiler and so on.

The Operation System (OS) divides time in small portions (quantum of time) and assigns CPU to each process (email, word, or the compiler). To make this work, a queue is needed to make all the processes wait for their turn. A timer and interrupt system needed to interrupt the CPU after the quantum time for a process is over. Structures are needed to save all the register values of the CPU and let the registers to be used by the other process. Then after the first process gets the CPU for another time, the saved CPU values will be placed in CPU registers one more time. It looks like as if each process has its own CPU.

We want to simulate what is explained. For example, a structure or class is defined which will play the role of CPU registers. Part of the code you will write will do the function of the CPU. In other words, CPU will be a simulation of the CPU. An integer field is designated for PC (program counter) and another field for the accumulator. Also, to simplify this simulation, the project has defined a simple set of instruction set. For example, your code reads the next instruction from the array or a file (i.e. this is the simulation of CPU fetching the next instruction). Then using a switch structure, your code decides if the instruction is a S 20 or A 20 and so on (i.e. this is the simulation of decoding the instruction by the CPU). If the instruction is S 20 then the value 20 is stored in a variable (i.e. this is the simulation of the execution cycle of the CPU where the value 20 is placed in the accumulator).

To make this simulation realistic, two processes will be created that will communicate to each other using a pipe.

Rather than having a timer and an interrupt system to interrupt CPU after a quantum amount of time, we will use the keyboard and type characters such as Q, U, P and T. When the user types character Q, it means one time unit has passed. So, CPU can fetch and execute the next instruction. Note that the CPU is also a simulation. In other words, your code will read the next instruction from an array or a file and execute that instruction. The set of instructions defined in the project are simple. For example, the instruction S 20 means to set the value of an integer variable to 20. While this program (process) is performing these tasks another process reads the keyboard for characters such as Q, U, P and T. These two processes communicate with each other using a pipe. Pipe is like a queue that is shared by the two processes. The first process writes the character to the pipe (to the queue) and the other process gets it from the other end of the pipe.

To start the project, in the first step, you do not need to worry about CPU simulation and process control block. All we need to do is to create two processes that communicate with each other using a pipe.

Your program should output a prompt such as $ and wait for the user input. The user can input one of these four characters (Q, U, P or T) as described below:

1. **Q**: End of one unit of time.
2. **U**: Unblock the first simulated process in blocked queue.
3. **P**: Print the current state of the system.
4. **T**: Print the average turnaround time, and terminate the system.

For simplicity, in the first step of the project, do not implement the function for each character. All you must do is to send this character to a pipe. The process at the other end of the pipe will receive this character. Then it should display what character it is received. The first process should continue displaying the prompt one more time. The program must end when the user enters T.

The code is called the commander process.