Processes

Tom Kelliher, CS 411

1 Announcements

2 Discussion

1. What resources are required by a process in order for it to run?
2. Figure 4.2 shows four process transitions. In theory, there could be six transitions, two out of each state. Are there conditions under which either of the missing transitions might occur?

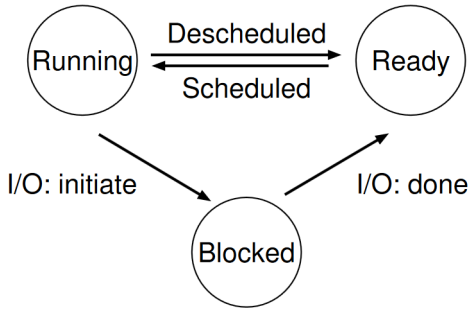

Figure 4.2: **Process: State Transitions**

   Give specific examples of process events that cause a process to transition from one state to another. How many processes can be in each state simultaneously?

3. Which of the following instructions should be privileged?
   - Set value of timer.
   - Read the clock.
   - Clear memory.
   - Issue a trap instruction.
   - Turn off interrupts.
   - Modify entries in device-status table.
   - Switch from user to kernel mode.
   - Access I/O device.
4. Describe what occurs during a context switch. Go into detail.

5. Once the operating system "hands over" the CPU to a user program, how does it get control back? Why would it be a problem if it couldn't get control back?

3 Assignment

You'll work on this assignment using the `Processes` directory of your repository. Start by fetching it from `upstream` and pushing it to your `origin`

```
git pull upstream main
git push
```

The programs you write should be saved in the `Processes` directory. Note that the `Processes` directory contains a directory, `Cpu-api`, that contains the example programs from Chapter 5.

Create a `README.md` file, saved in your `Processes` directory, to answer the questions below.

1. Write a program that opens a file (with the `open()` system call) and then calls `fork()` to create a new process. Can both the child and parent access the file descriptor returned by `open()`? What happens when they are writing to the file concurrently, i.e., at the same time?

   Name this program `fork1.c`.

2. Write a program that calls `fork()` and then calls `execvp()` to run the program `ls`. Then, modify the program to use `wait()` so that the parent waits for the child to finish. What happens if you use `wait` in the child? Finally, use `waitpid()` instead of `wait` in the parent. When would `waitpid` be useful?

   Name this program `fork2.c`.

3. Write a program that creates two children, and connects the standard output of one to the standard input of the other, using the `pipe()` system call. Your program should take file names as two command-line parameters. The child writing into the pipe will read the first file's contents, writing it into the pipe. Similarly, the child reading from the pipe will write the data it reads from the pipe to the second file. Try to handle error conditions appropriately so that none of your program's processes "hang."

   Name this program `fork3.c`.