CS 377 – Operating Systems

Discussion Session 3 Questions

Name: _____

Write your answers individually without consulting your notes, the textbook, or the internet. Be succinct (complete sentences not necessary). **Remember to turn your paper over.**

1. **Process Lifecycle**. The lifecycle of a process consists of five execution states, which are (in no particular order): running, terminated, new, ready, and waiting. Say what each of these states means in a few words and fill in the state labels in the sequence graph shown below.

   Solution.

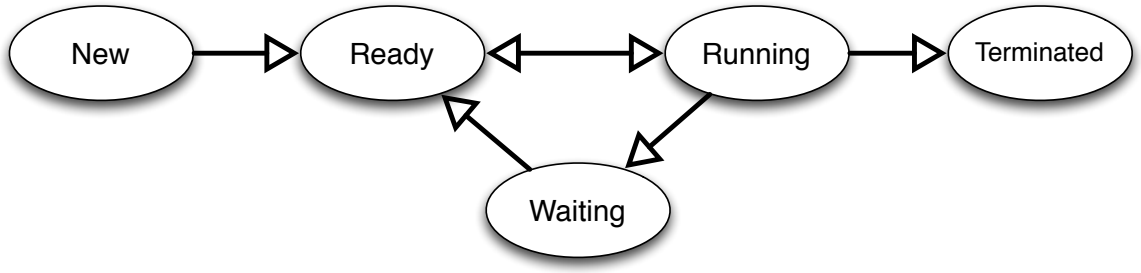

Figure 1: A state sequence graph depicting the lifecycle of a process.

`new` – *the process state is being created and initialized by the OS*
`ready` – *the process is ready to run and waiting to be executed by the CPU*
`running` – *the process is executing instructions on the CPU*
`waiting` – *the process is waiting for an event to complete (e.g., an I/O operation)*
`terminated` – *the process is finished and is being destroyed by the OS*

2. **Process Communication**. Two models of process communication are message passing and shared memory. Which model would you use for describing the scenario of (a) having a face-to-face conversation, and (b) communicating by letters? Briefly justify.

   Solution. *A face-to-face conversation would be an example of shared memory. Each person can speak (i.e., write to shared memory) and the listener will receive the message immediately (i.e., reading shared memory). However, both people cannot speak at the same time (since updates overwrite each other) or listen at the same time (since no communication will occur).*

   Communication by letters would be an example of message passing. Here, each person can simultaneously dispatch letters (messages) and they are explicitly received and read by the other party. Additionally, there may be some delay between message send and message receipt, and messages may not arrive in the order they were originally sent.

3. **Process Creation and `fork`**. Consider the following C program:

```
1   #include <stdio.h>
2
3   int main() {
4     printf("1");
5     int pid = fork();
6     if (pid > 0) {
7       waitpid(pid, 0, 0); // wait until process ID pid has terminated
8       printf("2");
9     }
10    pid = fork();
11    if (pid > 0) {
12      waitpid(pid, 0, 0);
13      printf("3");
14    } else {
15      printf("4");
16      return 0;
17    }
18    printf("5");
19    return 0;
20  }
```

(a) Including the initial process launched when the program is started, how many processes are created from running the above program?

Solution. *Four processes - the initial process, the child process created from the fork at line 5, then two additional processes from the fork at line 10 (one each from the initial process and its child process).*

(b) Assuming no output buffering (that is, output from each `printf` is written to the screen immediately), what is the output from running the above program?

Solution. 14352435

(c) Suppose we delete the call to `waitpid()` at line 12. Would this change your answer to part (b)? Why or why not?

Solution. *Deleting this call means that the parent and child processes resulting from the fork() at line 10 will run concurrently. This means that the output of the program is nondeterministic – if the child executes on the CPU first, then "4" will be printed first, while if the parent executes first, then "3" will be printed first. Thus, while the same output from part (b) is possible, different outputs are also possible.*