CIS 5480 Recitation 3 - Process Groups & Terminal Control

Welcome to recitation!!! 🦚

Name: _____ **PennID: _____**

1. It's Terminal (Control)

1.1 Sessions and Terminals

The following output is given from a run of the ps command. Use the details to answer the questions below. Note: TPGID is the PGID for the current foreground process on a given terminal.

USER	PPID	PID	PGRP	TPGID	SID	TTY	CMD
kyrdo	35943	35950	35950	68461	35950	pts/0	-bash
kyrdo	53690	53695	53695	68468	53695	pts/2	-bash
kyrdo	35950	68406	68405	68461	35950	pts/0	sleep 200
kyrdo	35950	68461	68461	68461	35950	pts/0	cat
kyrdo	35950	68462	68461	68461	35950	pts/0	grep How
kyrdo	35950	68463	68461	68461	35950	pts/0	uniq
kyrdo	53695	68464	68464	68468	53695	pts/2	sleep 300
kyrdo	53695	68465	68464	68468	53695	pts/2	sleep 200
kyrdo	53695	68468	68468	68468	53695	pts/2	ps

Foreground. For each foreground job, identify the following: 1) the commands that make up the job and their associated process ID, 2) the process group of the job, 3) the job's parent process and 4) what terminal and session the process is connected to.

Background. For each background job, identify the following: 1) the commands that make up the job and their associated process ID, 2) the process group of the job, 3) the job's parent process and 4) what terminal and session the process is connected to.

1.2 Process Groups and Penn-Shell

Your penn-shell partner claims that you do not need to isolate a foreground job into its own process group and assign it terminal control as the parent process will wait for it to finish anyway before trying to do anything with the terminal. Do you agree with your partner? Why or why not?

2. Race to the Terminal

2.1 Who Gets There First?

Your penn-shell partner has written the following code to launch a single foreground job. Assume the shell currently owns the terminal and `pid` will also be used as the new job's PGID.

```
pid_t pid = fork();

if (pid == 0) {
   execvp(argv[0], argv);
   perror("execvp");
   exit(EXIT_FAILURE);
} else {
   setpgid(pid, pid);
   tcsetpgrp(STDIN_FILENO, pid);

   waitpid(pid, NULL, 0);
}
```

Your partner tests the shell and notices that it usually works, but occasionally setpgid()/terminal setup fails or the child process gets stopped.

a. Recall that a race condition occurs when some behaviour depends on the order in which the processes are run. What race condition exists between the parent and child?

b. Consider the following two possible schedules:

   Schedule A: Parent executes setpgid() before the child executes execvp().
   Schedule B: Child executes execvp() before the parent executes setpgid().

   Explain what can happen in each case.

c. Modify the code (above) so that establishing the child's process group does not depend on whether the parent or child runs first.

2.2 Pipeline Launch

Your shell is launching a pipeline. All processes in the pipeline should belong to the same process group, with the first child acting as the process-group leader. Consider the following simplified code:

```
pid_t leader_pid = 0;

for (int i = 0; i < pipeline_length; ++i) {
   pid_t pid = fork();

   if (pid == 0) {
       if (i == 0) {
           // First child creates the process group
           setpgid(0, 0);
       } else {
           // Later children join the first child's group
           setpgid(0, leader_pid);
       }

       execvp(...);

   } else {
       if (i == 0) {
           leader_pid = pid;
       }
   }
}
```

Assume the first child receives PID 200, so the intended process group for the entire pipeline is PGID 200.

a. There is a race condition in this code snippet. What is it?

b. Modify the shell so that creation of PGID 200 does not depend solely on whether child 200 happens to run before child 201.

3: Who Gets the Signal?

Assume the following:
Shell PID = 100
Shell PGID = 100
Child PID = 200

For each scenario below:

1. Identify the foreground process group.
2. Identify which process(es) receive SIGINT if the user types Ctrl-C.
3. Identify which process(es) receive SIGTSTP if the user types Ctrl-Z.

Scenario A: No PGrp Setup

```
pid_t pid = fork();

if (pid == 0) {
   execvp(...);
} else {
   waitpid(pid, NULL, 0);
}
```

Scenario B: PGrp But No Terminal Handoff

```
if (pid == 0) {
   setpgid(0, 0);
   execvp(...);
}

setpgid(pid, pid);
waitpid(pid, NULL, 0);
```

Scenario C: Foreground Job Setup

```
setpgid(pid, pid);
tcsetpgrp(STDIN_FILENO, pid);
waitpid(-pid, NULL, 0);
```

Scenario D: Pipeline

Shell: PID 100, PGID 100

cat: PID 200, PGID 200
grep: PID 201, PGID 200
wc: PID 202, PGID 200

Foreground PGID = 200