CS240 - Operating Systems
Lab Assignment #3
Spring 2024

The purpose of this project is to use UNIX system calls to create and manipulate processes.

You are to write a simple "parallel batch processing" facility. This program should execute the UNIX "commands" that are contained in a data file. The name of this file should be read from `stdin`. The program should determine the maximum number of processes to have active at any one time from a value input from the command line. The program should then execute each command listed in the data file. Several commands, up to the maximum number of processes, should execute at the same time. If the number of commands in the data file exceeds the maximum number of processes, then the batch system should wait until one (or more) of the commands completes before starting to execute another command; there should be no more than the maximum number of processes executing at any one time.

Since it can be difficult to see what is happening with several processes executing at once, you will probably want to use the program you wrote for assignment #1 to test your batch processor. Since it remains in the system for a while (because of the sleeping it does) and it announces itself by name and process ID, an otherwise cluttered output should be somewhat more easily decipherable. However, note that the batch processor should be able to run *ANY* executable file, not *just* your sleep program.

As an example, suppose a file named `batchcmds` is created with the following lines. Assume that `testprog1` through `testprog6` are identical files, each containing the executable produced by compiling your assignment #1. This copying and renaming isn't necessary for proper operation, but it makes the output reporting somewhat easier to follow:

```
testprog1 5 5
testprog2 4 10
gcc testprog.c
ls -l
testprog3 13 3
testprog4 1 15
testprog5 3 5
testprog6 10 8
```

The following command would execute the commands in the batch file, up to 4 commands at a time (assuming the name of the batch processor program is `batchpro`). Note that the name of the batch file is entered on the command line here, by redirecting standard input.

```
batchpro 4 <batchcmds
```

The exact sequence in which the programs in the example shown above will be executed will be determined by your program, the O/S scheduler and the other load on the system; however,

a plausible sequence might be for `testprog1` and `testprog2` to be launched, followed by `gcc` and then `ls`. These latter two will likely get finished before the `testprogs`, since they don't sleep. When they complete, then `testprog3` and `testprog4` will execute. Since `testprog4` only sleeps for 15 seconds total, it will probably finish next, at which time `testprog5` should execute. As soon as another process terminates, `testprog6` will execute. It will be the last to terminate, since it waits a total of 80 seconds (which will seem like an eternity!).

Some hints:

1. Be very careful when you test the batch processor - it is surprisingly easy to get into an infinite loop while creating processes!! This has the effect of filling up the system's process table, meaning nobody else can execute any more programs (including `root`, since even logging in requires that a new process be created!!). Be sure to verify that your looping logic is correct before doing the actual `fork-exec` commands. A good way to do this is to comment out the dangerous commands and add some messages to "simulate" the correct operation, so you can trace the logic of your program before you try actually creating real processes.
2. This program should use the `makearg` function that you wrote for assignment #2.
3. Your batch program should produce **NO** output itself - it should simply execute programs that themselves may produce output.