EE 107S: Assignment 7

Due: Wednesday, November 1, 2017 at 11:59 PM

Introduction

In this assignment you will learn some basic process management and Bash scripting.

Assignment

The assignment is to create a Bash script that will launch multiple instances of a given program, `runme`, in parallel. The program takes 2 seconds to run, and our goal is to make multiple instances of this program run in approximately 2 seconds as well. Furthermore, we want to redirect the outputs of each of the instances to individual files. Here are the steps you may follow to build up your solution.

Running `runme`

To build the executable, simply type `make`. This will generate an executable called `runme`. To use the executable, you must supply an ID number. This can be any number. For example, `./runme 1` is a valid execution. Note that the output contains the ID number that was given as well as how long the program took to execute. The program doesn't do much else other than sleep for 2 seconds and then output the execution time. Execution time should be very close to 2 seconds.

Sequential `batch-seq.sh`

Now you will write a script that runs instances of `runme` sequentially. The `batch-seq.sh` script should take a single argument that specifies the number of instances of `runme` to run. If an incorrect number of arguments is supplied, you should output the following message exactly and exit with error code 1:

```
Usage: ./batch-seq.sh num
```

If exactly 1 argument is supplied, then you should continue the script like normal. For each instance of `runme` that the script executes, you should create a single file that contains the output of that instance. Recall that to redirect the output of an executable, you can use the `>` character. For example, to redirect the output of a single instance of `runme`, you can run the command:
`./runme 2 > log2.txt`.

The name of the output files should be given as `log<ID>.txt`, where `<ID>` is the ID argument of the `runme` instance. Thus, if you run `./batch-seq.sh 5`, you should expect the script to create 5 files: `log1.txt`, `log2.txt`, `log3.txt`, `log4.txt`, and `log5.txt`. The contents of `log4.txt` should look similar to:

```
Starting process 4
Execution time: 2002.88 ms
```

The other logs will look similar, except the ID will be different.

Note: Remember that your Bash script must begin with the line `#!/bin/bash`, and you must run `chmod +x batch-seq.sh` once before you can execute it.

Parallelizing and timing `batch-par.sh`

Our goal is to run multiple instances of `runme` concurrently. It is fairly simple to change a sequential version to be parallel in Bash, but first we'll look at how to time the execution of a command.

Timing commands

To see how long the execution of the entire `batch-seq.sh` script takes, you can use the `time` command. The `time` command allows you to supply a command and then it outputs the amount of time it took to run the command. If you run the command `time ./batch-seq.sh 10`, the output of the will look similar to:

```
./batch-seq.sh 10  0.02s user 0.02s system 2% cpu 20.109 total
```

The relevant number is the last one, `20.109`, which refers to the seconds of execution time. Since we are running 10 `runme` instances sequentially, the execution will be roughly $10 * 2 = 20$ seconds. This is because we must wait for each `runme` instance to fully complete after 2 seconds before moving to the next one.

Creating `batch-par.sh`

The second goal of this assignment is to create a parallel version of `batch-seq.sh` called `batch-par.sh`. The semantics of this script will be the same, but it should run all the `runme` instances in parallel. As a result, running `./batch-par.sh 10` and `./batch-par.sh 5` should yield the same runtime of 2 seconds. In an ideal world, `batch-par.sh` should run in 2 seconds regardless of the number supplied, but each parallel instance of `runme` incurs some overhead.

There are two important questions to ask when trying to run programs in parallel in Bash. Is there something we've covered in Bash that allows you to run jobs concurrently? If we manage to run all the jobs concurrently, how do we wait until they've all completed before moving on? It turns out the parallelized script will only be slightly different than the sequential script. This may be one of the few cases in which it's significantly easier to do something in a Bash script than a Python script!

Once you have parallelized `batch-par.sh` and you run the command `time ./batch-par.sh 10`, you should expect the output of time to look similar to:

```
./batch-par.sh 10  0.02s user 0.02s system 2% cpu 2.030 total
```

Since all instances of `runme` are executing in parallel, even though each one takes 2 seconds to execute, the overall time only marginally increases past 2 seconds.

Submission

You should submit a zip to Canvas file called `as7.sh` with the `batch-seq.sh` and `batch-par.sh` files in it.