**Objectives**

To implement a simple job scheduler that executes non-interactive jobs (e.g., jobs that do not have direct user interaction, jobs that can run in the background).

## Description

In this project we will implement a simple job scheduler that will execute non- interactive jobs (for example, jobs that do not have direct user interaction, jobs that can run in the background).

At any given time, only P jobs should be executing, and P is provided as an argument to your program.

If you have more than P jobs submitted, then these additional jobs must wait until one of the P executing jobs are completed.

You can assume that P is typically the number of cores that are available on a system and you are executing one process per core and if there are more jobs than the available number of cores these jobs must wait for one of the processes to complete.

When you launch the program with P as the command-line argument, the program should print a prompt and wait for the user to enter commands.

As the jobs are non-interactive, the output and error streams from the job must be written to separate files - *<jobid>*.out and *<jobid>*.err, where <*jobid*> is the appropriate job id that is associated with a job.

The following commands must be supported by your program:

|  |  |
| --- | --- |
| **Commands** | **Description** |
| **submit** program *arguments* | Create a new process to execute the program specified with any arguments and print a jobid to standard output. |
| **showjobs** | List all process that are either currently waiting or running (only those process that were started using the **submit** command). The output should include the jobid assigned to each job and the status of the jobs (whether it is running or waiting). If the job has completed, it will not be listed. |
| **submithistory** | List all the processes that were executed by your job scheduler, including the name of the job, the jobid that was assigned to it, the start time and end time of the job, and the status of the job (whether the job completed successfully or not). |

## Example

The following is a sample session:

**$ ./mysched 2**

Enter command> **submit /home/UAB/unan/332/shared/w1 1000**

job 1 added to the queue

Enter command> **submit /home/UAB/unan/332/shared/w2 1000**

job 2 added to the queue

Enter command> **submit /home/UAB/unan/332/shared/w3 1000**

job 3 added to the queue Enter command> **showjobs**

jobid command status

1. /home/UAB/unan/332/shared/w1 1000 Running
2. /home/UAB/unan/332/shared/w2 1000 Running
3. /home/UAB/unan/332/shared/w3 1000 Waiting Enter command>

*#after some time*

Enter command> **showjobs**

jobid command status

1 /home/UAB/unan/332/shared/w3 1000 Running Enter command> **submithistory**

jobid command starttime endtime status

1 /home/UAB/unan/332/shared/w1 1000 Thu Oct 10 17:43:44 2019 Thu Oct 1 0 17:43:44 2019 Success

3 /home/UAB/unan/332/shared/w2 1000 Thu Oct 10 17:43:44 2019 Thu Oct 1 0 17:43:44 2019 Success

Enter command>

**Guidelines and Hints**

1. Do not hardcore the value of P, make sure that your program will work for any value of P that can be between 1 to maximum number of cores available on a system (say, 8). You must test your program with at least P = 1, 2, 4 on the CS Linux servers.
2. You will need a queue to keep track of the jobs that are submitted. You can use the sample circular queue implementation provided here: queue.tar

Use the following command to create PDFs of your source code.(replace

the <Source\_code\_File\_name> and <Output\_Source\_code\_File\_name> with your C source code file name):

$enscript <Source\_code\_File\_Name>.c -o - | ps2pdf - <Output\_Source\_code\_File\_ Name>.pdf