Document Version: 1.1 (a newer version number means an update on the project)
Assignment Date: 9/11/2018
Due Date - No Penalty: 9/25/2018, 11:59pm
Due Date - 10 Points Penalty: 10/2/2018, 11:59pm
Group Info: Individual assignment - no groups

Objectives:

- Start writing non-trivial C programs
- Practice with data structures
- Simulate some CPU scheduling algorithms
- Get ready for the Test 1

1 Project Description

In this project, you will write a program called **sched**, which simulates First Come First Served (FCFS) and Preemptive Priority (PP) CPU scheduling algorithms. The program will be invoked as follows:

```
./sched <inputfile> <outputfile> <algorithm> [limit]
```

- **<inputfile>** is the name of a text file including the information about the processes to be scheduled and it will have the following format:

  ```
  <pid> <arrival-time> <burst-time> <priority>
  ```

  - <pid> is a positive integer number representing the process id of the process
  - <arrival-time> is a non-negative integer number given in **milliseconds** representing the arrival time of the process
  - <burst-time> is a positive integer number indicating the amount of CPU-time that the process requires
  - <priority> is a non-negative integer number indicating the priority of the process. Lower number indicates a higher priority.

  For example, the following can be a sample input file:

  ```
  1 0 4 70
  2 4 19 120
  3 9 5 100
  4 14 3 0
  5 16 2 3
  6 23 2 20
  ```

  Each line of the input file represents a different process, starts without any space, ends with a newline, and there is a single space (' ') in between each field in a line. Note that the example input file provided above is just for illustration purposes. Your program will be tested using input files containing much larger number of processes. Here is more information about the content of the input file:

  - All three fields are guaranteed to appear in the input file.
  - Processes are sorted by their arrival times in increasing order.
  - Processes have unique arrival times.
  - Arrival time of the first process is guaranteed to be 0.

- **<outputfile>** is the name of the text file including the result of the simulation. You will receive the name as a command line argument, create the file, and fill its content. Each line of the output file will contain the simulation result of a different process previously provided in the input file and the results will have the following format:

  ```
  <pid> <arrival-time> <finish-time> <waiting-time>
  ```

  The results in the output file should appear in sorted (increasing) order with respect to their `<finish-time>`. Each line of the output file should start without any space and should end with a newline, where there is a single space (' ') in between each field in a line. Here is more information about the content of the output file:

  - `<finish-time>` is a positive integer number in **milliseconds** representing the completion time of the process with respect to its `<arrival-time>`
  - `<waiting-time>` is a non-negative integer number in **milliseconds** representing the total time that the process waited in the ready queue

- **<algorithm>** can be one of the followings:

  - FCFS
  - PP

  FCFS will simulate the First Come First Served CPU scheduling algorithm and PP will simulate the Preemptive Priority CPU scheduling algorithm. **All units are in milliseconds and all values will be integers. If there is a tie for PP, then FCFS's rule will be used to break the tie.**

- **[limit]** is an **optional** command-line argument. If this argument is not provided, then you will simulate the whole input file until its end; otherwise, you will only simulate the first `[limit]` processes. For instance, if limit is given as 10, then you will only simulate the first 10 processes and you program will terminate without simulating the rest of the input file. `[limit]` is a positive integer less than or equal to the number of lines in the input file.

  For example, if your program is invoked as:

  ```
  ./sched in.txt out.txt FCFS
  ```

  then it will read the input file "`in.txt`", compute the finish time and waiting time of each process provided in this input file by simulating the FCFS algorithm, and write the result for each process to "`out.txt`". Note that input and output file names can be different. Assume that the content of "`in.txt`" is as the sample input file provided above. Then after this invocation, the content of the output file should be as follows:

  ```
  1 0 4 0
  2 4 23 0
  3 9 28 14
  4 14 31 14
  5 16 33 15
  6 23 35 10
  ```

  For the same input file content, if your program is invoked as:

  ```
  ./sched in.txt out.txt PP
  ```

  then the content of the output file should be as follows:

```
1 0 4 0
3 9 14 0
4 14 17 0
5 16 19 1
6 23 25 0
2 4 35 12
```

For the same input file content, if your program is invoked as:

```
./sched in.txt out.txt PP 3
```

then the content of the output file should be as follows:

```
1 0 4 0
3 9 14 0
2 4 28 5
```

2 Development

You will develop your program in a Unix environment using the C programming language. *gcc* must be used as the compiler. You will be provided a Makefile and your program should compile without any errors/warnings using this Makefile. Black-box testing will be applied and your program's output will be compared to the correct output. A black-box testing script will be provided in BlackBoard, make sure that your program produces the success messages in that test. A more complicated test (possibly more then one test) will be applied to grade your program. Submissions not following the specified rules here will be penalized.

3 Checking the Memory Leaks

You will need to make dynamic memory allocation. If you do not deallocate the memory that you allocated previously using **`free()`**, it means that your program has memory leaks. To receive full credit, your program should be memory-leak free. You can use `valgrind` to check the memory-leaks in your program. `valgrind` will output:

"All heap blocks were freed - no leaks are possible"

if your program is memory-leak free. Check `man valgrind` for more details on `valgrind`.

4 Submission

Submission will be done through Blackboard strictly following the instructions below. Your work will be penalized 5 points out of 100 if the submission instructions are not followed. Memory leaks and compilation warnings will also be penalized with 5 points each, if any. You can check the compilation warnings using the **-Wall** flag of gcc.

4.1 What to Submit

1. `sched.c`: including the source code of your program.

4.2 How to Submit

1. Create a directory and name it as your UofL ID number. For example, if a student's ID is 1234567, then the name of the directory will be 1234567.
2. Put all the files to be submitted (only the ones asked in the *What to Submit* section above) into this directory.
3. Zip the directory. As a result, you will have the file 1234567.zip.
4. Upload the file 1234567.zip to Blackboard using the "Attach File" option. You do not need to write anything to the "Submission" and "Comments" sections. NO LATE SUBMISSIONS WILL BE GRADED!
5. You are allowed to make multiple attempts of submission but only the latest attempt submitted before the deadline will be graded.

5 Grading

Grading of your program will be done by an automated process. Your programs will also be passed through a copy-checker program which will examine the source codes if they are original or copied. We will also examine your source file(s) manually.

6 Changes

- Version 1.1: Typo fixed in algorithm abbreviation (NPP changed to PP).