CPSC 3220 Assignment 2 (50 points)

This is a team assignment. Done in groups of 2 (no more than 2) or individually.

Problem Statement

You will implement a simulation of two simple scheduling policies on a single CPU: Priority Based (non-preemptive) and Custom (Preemptive). The policy to simulate should be selected by a command line argument of -prio, or -custom, respectively. Input data file will be passed as input redirection. Output filename will be passed as output redirection. (Examples below)

Grading

- (4 points) Select policy using CLAs.
- (6 points) Read from a specified file and write to the specified output file via input/output redirection.
- (18 points) Priority Based (non-preemptive) implementation and correct output
- (18 points) Custom (preemptive) implementation and correct output
- (4 points) Submission in tar.gz format that contains source code files without subdirectories.

Note: any corrupt tarball submission or a submission that does not compile on SoC Linux machines will receive 0 points. You should have NO compile or runtime warnings. Please recompile after making the last moment changes to make sure your program still works.

Discussion

You are a programmer at a company that aims to design better scheduling algorithms for the new Linux-like Operating system. You were tasked with implementing TWO new algorithms. The first one will implement Priority Based non-preemptive algorithm that works the following way. Every clock tick the process with the highest priority number (lowest number) will be selected and executed non-preemptively. If several processes have the same priority number, then the one with the shortest service time will be given execution priority. If both priority and service time are the same, then you can arbitrarily pick the process to execute.

The second algorithm will implement a Custom Scheduling algorithm, which is a variation of the Priority Based algorithm. This is a preemptive algorithm. Every clock tick a process with the highest priority (lowest number) will be selected to run, and will run for one time quantum (one clock tick). Its priority will then be lowered by one (which means your integer is incremented) and it will be placed back on the ready queue. Higher number = lower priority. If several processes have the same priority, the process with the shortest remaining service time will be selected to run next. If both priority and service time are the same, then you can arbitrarily pick the process to execute.

You will read groups of three numbers from an input file via input redirection. The first number on each line is the arrival time of a task, the second number is the service time of that task, and the third number will be the priority of the process. The input groups will be ordered according to ascending arrival times, and multiple tasks can arrive at the same time.

For each scheduling algorithm your program should assign alphabetic task identifiers to each task in ascending order, starting with tid Y. You may assume that there will be no more than 26 tasks. When task ids reach the end of the alphabet, the names will roll over to the beginning of the alphabet. Note that there may be idle times prior to the first task arriving, idle times between groups of tasks, as well as tasks arriving at the same time. Below is the example of a valid workload:

```
3 4  7
10 4  7
10 2  8
```

Your simulation should first print (to the specified output file) a trace of each time unit, labeled as the discrete time value at the beginning of the time unit and showing what task is running on the CPU and what tasks are waiting in the ready queue during that time unit. Any arrivals, preemption, and scheduling occur at the discrete time value that starts a time unit and incur zero overhead. Completions occur at the discrete time value that ends a time unit.

During the trace, the tasks should be identified as the tid concatenated with rst, where tid is the task identifier assigned upon arrival and rst is the current *remaining service time* evaluated at the start of a time unit.

You should stop the simulation after all tasks have completed. (Note that this means that the CPU is empty, the ready queue is empty, and there is no more input.)

Note that for Custom, a running task is preempted only when an arriving task has less service time.

After the simulation ends, your program should print (to the output file via redirection) a task summary table, ordered by ascending task identifier and containing the tid, arrival time, service time, priority, completion time, response time, and wait time for each task. Finally, your program should print (to the same file) a table of service time and wait time pairs from the task summary table, ordered by ascending service time.

Language and filename Requirements

This program is written in C programming language. You will use linked lists, with each task being represented by a *struct task* shown below. You can only access this struct via a pointer to this struct. You can name that pointer as you wish. Name your source file *sched.c.* Ensure it has your name on top of the file in comments.

```
struct task{
  int
  task_id,    /* alphabetic tid can be obtained as 'A'+(task_counter++) */
    arrival_time,
    service_time,
    remaining_time,
    completion_time,
    response_time,
    wait_time;
    priority;
  struct task *next;
};
```

Turn in your program via Canvas. Your submitted file is a *.tar.gz* archive with no subdirectories. Your code must run on the School of Computing machines to be graded.

Guidelines

The entire code should be written by you and your partner, if you have one. You are not allowed to consult anyone else except your teacher and TA. If you have a partner, only ONE SUBMISSION PER TEAM is to be made. Teammates share the grade, so team up wisely.

You should not send code to anyone or receive code from anyone, whether by email, Discord, social media, printed listings, photos, visual display on a computer/laptop/cell-phone/etc. screen, or *any* other method of communication.

Do not post the assignment, or a request for help, or your code on *any* web sites.

The key idea is that you shouldn't short-circuit the learning process for others once you know the answer. (And you shouldn't burden anyone else with inappropriate requests for code or "answers" and thus short-circuit your own learning process.)

Example Expected Output

Let *in1* be an input file containing:

```
3 4  7
10 4  6
10 2  8
```

For the command line "./a.out -prio < in1 > out1", the output written to *out1* file would be:

```
Priority scheduling results

time  cpu  priority ready queue (tid/rst)
----------------------------
  0                    --
  1                    --
  2                    --
  3    Y4     7        --
  4    Y3     7        --
  5    Y2     7         --
  6    Y1     7         --
  7                     --
  8                     --
  9                     --
 10    Z4     6         A2 8
 11    Z3     6         A2 8
 12    Z2     6         A2 8
 13    Z1     6         A2 8
 14    A2     8         --
 15    A1     8         --

              arrival  service completion response wait
tid   prio    time     time    time       time     time
--------------------------------------------------------
 A      8      10       2        16         6        4
 Y      7       3       4         7         4        0
 Z      6      10       4        14         4        0

service wait
 time   time
------- ----
  2      4
  4      0
  4      0
```

For the command line "./a.out -custom < in1 > out1", the output would be:

```
Custom(preemptive) scheduling results

time   cpu   priority  ready queue (tid/rst)
---------------------------------
  0                     --
  1                     --
  2                     --
  3    Y4     7         --
  4    Y3     6         --
  5    Y2     5         --
  6    Y1     4         --
  7                     --
  8                     --
  9                     --
 10    Z4     6         A2 8
 11    Z3     7         A2 8
 12    Z2     8         --
 13    Z1     9         --
 14    A2     8         --
 15    A1     7         --

              arrival service completion response wait
tid   prio     time    time    time       time     time
--------------------------------------------------------
 A     8         10       2        16          6       4
 Y     7          3       4         7          4       0
 Z     6         10       4        14          4       0
```

```
service wait
 time   time
------------
  2       4
  4       0
  4       0
```

It is recommended that you use several input files with different number of processes and a variety of priorities to test your program. Hand trace data from each simulation to verify the numbers in the table. It is your responsibility to determine the correctness of your output. Please refer to the chapter discussing the five times (arrival, service, wait, response and finish times).

If you believe there is an error or a typo in the file, please report it to your teacher.