CIS 3110: Operating Systems

Assignment 3: CPU Scheduling

Due Saturday, March 16, 2019 @ 11:59pm

1. Objective

The objectives of this assignment is to implement a simplified Dispatcher for an OS. It will react to events and change states of processes admitted to the system respectfully, as described in lecture notes and the textbook. Events will be simulated. The sequence of events will be given in the standard input (one event per line). Empty line will signify the end of the input. Format of one line is as follows:

<time> <event> {<process id>}

where

- <time> is an integer number denoting local time in milliseconds measured from 0 (time will be strictly increasing from line to line);
- <event> is one of the following:
  - C - create
  - E - exit
  - R N - request resource number N
  - I N - interrupt from resource number N (request accomplished)
  - T - timer interrupt
- <process id> is a nonnegative integer, unique for each process (system idle process has <process id> = 0);

Here is an example of possible several first lines of the input (comments are here for explanation only):

```
100 C 1       \\ process 1 is created
200 C 2       \\ process 2 is created
250 R 3 2     \\ process 1 requests resource 3
300 C 3       \\ process 3 is created
400 T         \\ timer interrupt
500 R 2 3     \\ process 3 requests resource 2
1000 C 4      \\ process 4 is created
1050 T        \\ timer interrupt
```

1100 I 2 3 \\ interrupt from resource 2 (request of process 3 is accomplished)
1150 I 3 1 \\ interrupt from resource 3 (request of process 1 is accomplished)
2000 E 4 \\ process 4 exits
2100 R 2 2 \\ process 2 requests resource 2
2200 E 3 \\ process 3 exits
2300 R 2 1 \\ process 1 requests resource 2
3000 I 2 1 \\ interrupt from resource 2 (request of process 1 is accomplished)
3050 T \\ timer interrupt
3450 T \\ timer interrupt
3500 E 1 \\ process 1 exits
4000 I 2 2 \\ interrupt from resource 2 (request of process 2 is accomplished)
5000 E 2 \\ process 2 exits
...

Your Dispatcher will have to keep track of events and changes in the state of the processes, taking into account the following additional conditions:

- there are 5 different kinds of resources in the system and requests can be serviced out of order of arrival;
- there is time sharing, so the process which is in running state is to be preempted as the result of the timer interrupt if there are other ready processes in the system;
- running process can also exit or get blocked because of request for a resource;
- if there are no ready user processes, then process number 0 (system idle process) is running;
- if process 0 is running and new process is created, or as the result of an event one of the blocked processes becomes ready (unblocked), this process will get CPU immediately.

When all lines of the input are processed, Dispatcher will print the following cumulative information about all processes admitted to the system during simulation:

<process id> <total time Running> <total time Ready> <total time Blocked>

For the system idle process print only <total time Running> and assume that process 0 was created at time 0. The output is one line per process in the increasing order of process IDs.

For example for the input
100 C 1
200 C 2
250 R 3 1
300 C 3
400 T
500 R 2 3
1000 C 4
1050 T
1100 I 2 3
1150 I 3 1
2000 E 4
2100 R 2 2
2200 E 3
2300 R 2 1
3000 I 2 1
3050 T

```
3450 T
3500 E 1
4000 I 2 2
5000 E 2
```

expected output is

```
0 1300
1 750 1050 1600
2 1800 1100 1900
3 200 1100 600
4 950 50 0
```

You can assume that the sequence of events given in the input is consistent and no input errors are present.

2. How to Test Your Program

We provide you with several basic test cases, in file testX.in (as the input) and its expected output testX.out, where X is a test case number.

Assume that the resulted executable file after compiling your dispatcher C program is called idispatcher. For example, you can use the test case 0, which is the example mentioned above

```
$ ./idispatcher<./test_inputs/test0.in
0 1300
1 750 1050 1600
2 1800 1100 1900
3 200 1100 600
4 950 50 0
```

Afterwards, you can check the expected result, test0.out, against the actual result from your program.

```
$ cat ./test_outputs/test0.out
0 1300
1 750 1050 1600
2 1800 1100 1900
3 200 1100 600
4 950 50 0
```

3. Submission

- If you have any problems in the development of your programs, contact the teaching assistant (TA) assigned to this course.
- You are encouraged to discuss this project with fellow students. However, you are **not** allowed to share code with any student.
- If your TA is unable to run/test your program, you should present a demo arranged by your TA's request.
- Please submit your implementation in single file **idispatcher.c**.

- How to name your programming assignment projects: For any assignment, zip all the files required to a zip file with name: CIS3110_<assignment_number>_XXX.zip, where <assignment_number> is the assisgnement number you are solving (e.g., a3 for Programming Assignment 3) and XXX is your University of Guelph's email ID (Central Login ID). This naming convention facilitates the tasks of marking for the instructor and course TAs. It also helps you in organizing your course work. Failure to follow the requirements will result in mark reduction.

  Note: to zip and unzip files in Unix:

  ```
  $zip -r filename.zip files
  $unzip filename.zip
  ```