CSS-430 : Operating Systems : P3

**Note:** the OSC book says your solution for this project can be written in C or Java. For our class however, it must be in C.

### Scheduling Algorithms

This project involves implementing several different process scheduling algorithms. The scheduler will be assigned a predefined set of tasks and will schedule the tasks based on the selected scheduling algorithm. Each task is assigned a priority and CPU burst. The following scheduling algorithms will be implemented:

* First-come, first-served (FCFS), which schedules tasks in the order in which they request the CPU.
* Shortest-job-first (SJF), which schedules tasks in order of the length of the tasks' next CPU burst.
* Priority scheduling, which schedules tasks based on priority.
* Round-robin (RR) scheduling, where each task is run for a time quantum (or for the remainder of its CPU burst).
* Priority with round-robin, which schedules tasks in order of priority and uses round-robin scheduling for tasks with equal priority.

Priorities range from 1 to 10, where a higher numeric value indicates a higher relative priority. For round-robin scheduling, the length of a time quantum is 10 milliseconds.

#### I. Implementation

The implementation of this project should be completed in C. Program files supporting C are provided in the source code download for the text. (See <http://www.os-book.com>) These supporting files read in the schedule of tasks, insert the tasks into a list, and invoke the scheduler.

The schedule of tasks has the form [***task name***] [***priority***] [***CPU burst***], with the following example format:

T1, 4, 20

T2, 2, 25

T3, 3, 25

T4, 3, 15

T5, 10, 10

Thus, task T1 has priority 4 and a CPU burst of 20 milliseconds, and so forth. It is assumed that all tasks arrive at the same time, so your scheduler algorithms do not have to support higher-priority processes preempting processes with lower priorities. In addition, tasks do not have to be placed into a queue or list in any particular order.

There are a few different strategies for organizing the list of tasks, as first presented in Section [5.1.2](https://jigsaw.vitalsource.com/books/9781119320913/epub/OPS/c05.xhtml#c05-sec-0014). One approach is to place all tasks in a single unordered list, where the strategy for task selection depends on the scheduling algorithm. For example, SJF scheduling would search the list to find the task with the shortest next CPU burst. Alternatively, a list could be ordered according to scheduling criteria (that is, by priority). One other strategy involves having a separate queue for each unique priority, as shown in Figure [5.7](https://jigsaw.vitalsource.com/books/9781119320913/epub/OPS/c05.xhtml?favre=brett#c05-fig-0007). These approaches are briefly discussed in Section [5.3.6](https://jigsaw.vitalsource.com/books/9781119320913/epub/OPS/c05.xhtml#c05-sec-0022). It is also worth highlighting that we are using the terms ***list*** and ***queue*** somewhat interchangeably. However, a queue has very specific FIFO functionality, whereas a list does not have such strict insertion and deletion requirements. You are likely to find the functionality of a general list to be more suitable when completing this project.

#### II. C Implementation Details

The file driver.c reads in the schedule of tasks, inserts each task into a linked list, and invokes the process scheduler by calling the schedule() function. The schedule() function executes each task according to the specified scheduling algorithm. Tasks selected for execution on the CPU are determined by the pickNextTask() function and are executed by invoking the run() function defined in the CPU.c file. A Makefile is used to determine the specific scheduling algorithm that will be invoked by driver. For example, to build the FCFS scheduler, we would enter

make fcfs

and would execute the scheduler (using the schedule of tasks schedule.txt) as follows:

./fcfs schedule.txt

Refer to the README file in the source code download for further details. Before proceeding, be sure to familiarize yourself with the source code provided as well as the Makefile.

#### What to do when there are equally good choices

#### All tasks arrive at the same time. So there are many, many alternative, correct answers for the FCFS schedule. (In fact 5! = 120 choices). Similarly, T3 and T4 have equal priority; T2 and T3 have equal CPU burst time. All these factors mean there are many correct answers. This makes grading really hard. To avoid this problem, we apply one extra rule: when there is choice between two or more equally good tasks, choose the one that comes first, in the sequence T1, T2, T3, T4, T5. So, for example, the unique, correct solution for the FCFS schedule is: 20\*T1, 25\*T2, 25\*T3, 15\*T4, 10\*T5.

#### There are several ways to code this choice. Here is some starter code that may help:

#### bool comesBefore(char \*a, char \*b) { return strcmp(a, b) < 0; }

#### // based on traverse from list.c

#### // finds the task whose name comes first in dictionary

#### Task \*pickNextTask() {

#### // if list is empty, nothing to do

#### if (!g\_head)

#### return NULL;

#### struct node \*temp;

#### temp = g\_head;

#### Task \*best\_sofar = temp->task;

#### while (temp != NULL) {

#### if (comesBefore(temp->task->name, best\_sofar->name))

#### best\_sofar = temp->task;

#### temp = temp->next;

#### }

#### // delete the node from list, Task will get deleted later

#### delete (&g\_head, best\_sofar);

#### return best\_sofar;

#### }

#### What to Submit

#### Please upload your solutions in the following C files:

#### schedule\_fcfs.c

#### schedule\_sjf.c

#### schedule\_priority.c

#### schedule\_rr.c

#### schedule\_priority\_rr.c

#### In addition, if you had to change any of the downloaded files to get your solution working, please upload those altered files too. (Please aim to keep those changes small. You might get by with a tiny change like inserting #include-guards into one or two .h files)