Alexandria University
Faculty of Engineering
Comp. & Comm. Engineering
CC373: Operating Systems

جامعة الاسكندرية
كلية الهندسة
برنامج هندسة الحاسب والاتصالات
مادة نظم التشغيل

Lab4
CPU Scheduling

Introduction:

You are asked to implement one program in C/C++ to analyze, and visualize the following CPU Scheduling algorithms. You are encouraged to use STL so as to reduce the implementation effort (you do not need to implement queues or priority queues yourself):

1. FCFS (First Come First Serve)
2. RR (Round Robin)
3. SPN (Shortest Process Next)
4. SRT (Shortest Remaining Time)
5. HRRN (Highest Response Ratio Next)
6. FB-1, (Feedback where all queues have q=1)
7. FB-2i, (Feedback where q= 2^i)
8. Aging

Getting Input from User:

The input to your program will be sent through `stdin`. Your output should be sent to `stdout`. To test your program, you are provided with a set of testcases showing the format of the input. You do not have to type the input yourself. You can simply use input redirection (e.g., `./lab4 < 01a-input.txt`), or pipe the input to your binary (e.g., `cat 01a-input.txt | ./lab4`).

Input Format:

Line1. “trace” or “stats”.
“trace” is used to ask your program to visualize the processes switching over the CPU – just like the Figure 9.5 in your textbook (or slide 30 in the class presentation). You can see `01a-output.txt` as an example.
“stats” is used to ask your program to write some statistics on the scheduled processes – just like Table 9.5 in your textbook (or slide 31 in the class presentation). You can see `01b-output.txt` as an example.
used to visualize the processes

Line2. a comma-separated list telling which CPU scheduling policies to be analyzed/visualized along with their parameters, if applicable. Each algorithm is represented by a number as listed in the introduction section and as shown in the attached testcases.
Round Robin and Aging have a parameter specifying the quantum *q* to be used. Therefore, a policy entered as 2-4 means Round Robin with *q*=4. Also, policy 8-1 means Aging with *q*=1.

Line3. An integer specifying the last instant to be used in your simulation and to be shown on the timeline. Check the attached testcases.

Line4. An integer specifying the number of processes to be simulated. None of the processes is making a blocking call.

Line5. Start of description of processes. Each process is to be described on a separate line. For algorithms 1 through 7, each process is described using a comma-separated list specifying:
- One character specifying a process name
- Arrival Time
- Service Time

For algorithm 8, each process is described using a comma-separated list specifying:
- One character specifying a process name
- Arrival Time
- Priority

Output Format:

As mentioned earlier, your output will be sent to `stdout`. You must provide a Makefile through which your program is to be compiled to produce a binary called `lab4` in the same directory as the source code. Your program will be automatically graded. So, adhering to the output format is a must, else your program will fail the used testcases during grading. Check the files representing the output in the provided testcases for the strict format.

When visualizing the processes, you will use an asterisk "*" to show that the process is running in this period. You will use a period "`.`" To specify that the process was in the ready list at this time period.

In addition to any test case you write to test your code to make sure it takes care of boundary conditions, race conditions, …, you can use the provided testcases to verify that your code produces the right output format. You can use the following commands:

```
cat 01a-input.txt | ./lab4 | diff 01a-output.txt -
cat 01b-input.txt | ./lab4 | diff 01b-output.txt -
...
```

These commands show the differences (hence the command `diff`) between your program's output and the expected output. In case of discrepancies, your program fails the testcase.

Aging Scheduling Policy:

Xinu is an operating system developed at Purdue University. The scheduling invariant in Xinu assumes that at any time, the highest priority process eligible for CPU service is executing, with round-robin scheduling for processes of equal priority. Under this scheduling policy, the processes with the highest priority will always be executing. As a result, all the processes with lower priority will never get CPU time. As a result, starvation is produced in Xinu when we have two or more processes eligible for execution that have different priorities. For ease of discussion, we call the set of processes in the ready list and the current process as the eligible processes.

To overcome starvation, an **aging scheduler** may be used. On each rescheduling operation, a timeout for instance, the scheduler increases the priority of all the ready processes by a constant number. This avoids starvation as each ready process can be passed over by the scheduler only a finite number of times before it has the highest priority.

How aging is implemented in Xinu?

Each process has an *initial priority* that is assigned to it at process creation. Every time the scheduler is called it takes the following steps.
- The priority of the *current* process is set to the *initial priority* assigned to it.
- The priorities of all the ready processes (not the current process) are incremented by 1.
- The scheduler choses the highest priority process from among all the eligible processes.

Note that during each call to the scheduler, the complete ready list has to be traversed.

Deliverables:

- Step outside the directory containing your
  - complete source code, commented thoroughly and clearly
  - `Makefile`
- Name your work directory as id1-id2-lab# (for ids: 5678, 6789 and lab4, the directory should be called 5678-6789-lab4)
- Create a .tar.gz file using tar `cvfz 5678-6789-lab4.tar.gz` <path-to-directory>
- Append .pdf to the filename to be able to upload your project
- Submit to the form (Only one student submits for the whole group). Make sure to get a receipt.

Note that your source program will be compiled, then will be tested by running the following command, which should produce nothing in case of a successful test:

```
make
cat inputfile | ./lab4 | diff outputfile -
```

Note that if your `Makefile` produces a binary with any other name than lab4, it will be automatically skipped resulting in you getting no grade at all.

Note that your `Makefile` MUST NOT exist inside any subdirectory. Directly place it inside the directory to be id1-id2-lab4/Makefile

Note that the binary file produced by the `Makefile` MUST as well reside directly at id1-id2-lab4/lab4
If your binary resides at any other path, it will be automatically skipped resulting in you getting no grade at all.

Note that if your zipped file does not STRICTLY follow the naming convention, an ID might not be extracted and the grade assigned for your code will not be granted to that ID.

<u>**Note:**</u>
This is a team project. The team size is at most 2. One submission will be made for the project.