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

جامعة الاسكندرية
كلية الهندسة
برنامج هندسة الحاسب والاتصالات
مادة نظم التشغيل

Lab6
CPU Scheduling Policies

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 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 all queues have $q=2^i$)
8. Aging

Getting Input from User:

The input to your program will be sent through stdin. Your output should be sent to stdout. **You are provided with a set of test cases showing the input format to test your program.** You do not have to type the input yourself. You can simply use input redirection:

```
./lab6 < 01a-input.txt
```

or pipe the input to your binary:

```
cat 01a-input.txt | ./lab6
```

Input Format:

- **Line 1: "trace" or "stats"**

  "trace" is used to ask your program to visualize the processes switching over the CPU – just like Figure 9.5 in your textbook. 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. You can see 01b-output.txt as an example.

- **Line 2:**

  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 test cases.

  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.

- **Line 3:**

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

- **Line 4:**

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

- **Line 5:**

  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

Check the files representing the input to further understand the input format.

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 lab6 in the same directory as the source code. Your program will be automatically graded. So, adhering to the output format is a must, or else your program will fail the used test cases during grading. Check the files representing the output in the provided test cases 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 on the ready list at this 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 test cases to verify that your code produces the right output format. You can use the following commands:

```
cat 01a-input.txt | ./lab6 | diff 01a-output.txt -

cat 01b-input.txt | ./lab6 | 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 test case.** The test case only passes if the command doesn't produce any differences.

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 executed. 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 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 few times before it has the highest priority.

How is aging implemented by Xinu?

Each process has an initial priority 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 chooses 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.

Bonus:

Create a Dockerfile that builds a lightweight Docker image of your code.

Deliverables:

1. Step outside the directory containing your:
    - complete source code commented thoroughly and clearly
    - Makefile
    - Bonus: Dockerfile
2. Name your work directory as id1-id2--lab# (for id: 6000-7000 and lab#6, the directory should be called 6000-7000-lab6)
3. Create a .tar.gz file using:

```
tar cvfz ids...-lab5.tar.gz <path_of_directory>
```

4. Append .pdf to the filename to be able to upload your project

Submission Form: Form

Deadline: 26th December, 2024

Notes

- Languages used: C/C++.
- Students will work in a group of 2
- Groups may talk together about the algorithms or functions being used but are NOT allowed to look at anybody's code.
- Revise the academic integrity note found on the class web page.
- No Late Submissions Allowed!