Operating Systems Lab 2a: Scheduling

The second lab for Operating Systems will consist of two parts: 2a and 2b. The first part is on scheduling, while the second part will be on page replacement. The second part will be published next week; both parts have the same deadline!

Scheduling

In this exercise you are going to make your own virtual schedulers. You will make both a simple non-pre-emptive First-Come-First-Serve scheduler, and a pre-emptive Round-Robin scheduler with three priority classes and ageing. Priority 1 will be highest, and 3 will be the lowest. Use 10 ticks as the quantum for pre-emptive scheduling.

Remember that most processes are a mix of CPU and I/O. Assume that there is only one CPU and one I/O device, so only one process can do CPU work and one process can do I/O work at any moment. The scheduler will receive its input on `stdin`, which will look like:

```
0 1 100 25 100 20 50 20 100 10 200 -1
15 3 30 15 30 10 40 10 50 -1
...
```

Each line represents one process. The first column will be arrival time of the process, the second is its priority (this one can be ignored by the FCFS-scheduler, and is either 1, 2 or 3). After that, there are alternating CPU- and I/O-times, starting with CPU time. A -1 indicates the end of a process.

When all processes are done, the scheduler should print the calculation of the average turnaround time for all processes to `stdout` (do not forget the newline!). This number will be a double, when printing make sure to truncate the number to only print an integer, for example using `printf("%.0lf\n", ...);`. Remember that turnaround time is the time between creation of the process (i.e. entering the queue) and finishing the process.

For the pre-emptive Round-Robin scheduler, you are also asked to implement ageing. If ageing (or any other priority changing method) is not implemented, lower-priority processes might run into 'starvation', where a process does not get the opportunity to run. In the book (Tanenbaum, section 2.4.3, page 160) this is explained in greater detail. Furthermore, while the Round-Robin has three priority queues for the CPU, for the I/O it will act non-preemptively with a single queue (just like FCFS).

The main idea behind ageing is to make the priority of a process dynamic, such that after some time a lower-priority process can get a higher priority. To keep matters simple, the age of a process is simply the amount of time spent idling. After a process **surpasses** the age of 100 (which is equivalent to 10 quanta), its priority will increment and its age will reset (that is, the age will tick 98, 99, 100, 0, 1, ... - so age 100 is a valid age that does not need priority changes yet). When a process gets time to execute, its age will reset as well, but the priority will not reset. If a process gets time to execute at the same moment that its priority would increase, both should take place. Processes in the highest priority class (1) cannot age.

In the case that at a given time, both a new process starts and an existing process finishes an I/O block, first the CPU time of the new process should be queued, and then the CPU time of the existing process. The arrival times of processes are unique and increasing line-by-line. To clarify, this is the order of operations you will have to do in each iteration of Round-Robin:

1. Promote waiting processes to higher priority.

2. Handle new, incoming processes.
3. Handle processes finishing an IO task.
4. Handle processes finishing a CPU task.
5. Pre-emptively stop a process from running a CPU task.

On Themis, submit two separate implementations, one for the Round-Robin implementation and one for the FCFS implementation. Both have the exact same test cases. For Round-Robin, Themis will accept answers that are within 1% of the expected answer.

Report

For this lab, you will have to write a short report (pdf or README). In there, briefly describe your design decisions and the implementation of your code. There should be one report for both parts of the lab. Some remarks regarding its contents:

- Absolutely no more than 1500 words; but in general no more than necessary. The report should be short and concise.
- Discuss the implementation decisions, such as data structures used.
- Give a high-level overview of the code such that it helps us understand the code more quickly. This can and should also be done through code comments; they should complement each other. This overview should include general program flow and behaviour of important/critical functions.
- There is no need to discuss difficulties encountered throughout implementation.
- While the report should be concise, it should still be written in proper English.