Team 08
s192865 KATOUZIAN Pouria
s201354 Gerard Manon

Overview of your approach

We have utilized the structure explained in Appendix B of the assignment and followed the TA's instructions to complete the project. Step 1 corresponds to our function `handleSimulationEvents`, step 2 to `assignProcessesToResources` and step 4 corresponds to `advanceProcessTime` and `advanceSchedulingTime`.

Instead of following the third recommended step, which involved determining the next event time, we adopted a strategy of progressing through the simulation one time step at a time. This decision was prompted by the complexity introduced by implementing multi-core and multilevel queue systems, which necessitated extensive condition checking to identify the next event time. By advancing incrementally, we mitigated the risk of overlooking critical conditions. We addressed events comprehensively at each time step. Furthermore, this approach aligns more closely with the behavior of real-world schedulers, which must evaluate potential actions at each discrete time interval without skipping ahead.

In addition, we implemented a code that could generalize to any interrupt and switch in/out duration.

Joker

We did not use any joker. We implemented both multicore CPU and multi-level queue.

Project Comprehension

Q) Assuming that there is no context switch time (set to 0) and all processes are equally important, what do you think is the best scheduling algorithm among the four proposed? Why?

In the absence of any form of priority considerations and negligible context switch costs, we can eliminate priorities and also SJF as it has a significant disadvantage of leading to starvation. Between FCFS and Round Robin, the Round Robin scheduling algorithm stands out as the optimal choice because it avoids the convoy effect, where short jobs are stuck behind long ones, and ensures fairness while preventing starvation.

Q) Do you think this algorithm is implemented in most operating systems? Why?

Due to our research and understanding, various scheduling algorithms are used in different operating systems based on their necessity. In most operating systems, Round Robin (RR) is utilized, albeit in specific cases and at a lower level within their own scheduling algorithms, which are typically more complex. These systems often combine multiple scheduling algorithms or utilize different ones altogether. For instance, the Multilevel Feedback Queue algorithm, Completely Fair Queuing (CFQ), and hybrid scheduler (combining Multilevel Feedback Queue scheduling and Round Robin (RR)) are employed in Windows, Linux, and macOS.

In all three operating systems, RR is used at a lower level for Time Sharing System goals. In other words, it manages processes within specific priority queues or for short-term scheduling decisions that benefit from time slices. RR's strengths in fairness and ensuring all processes get a chance to run come into play, and it plays a supporting role in most operating systems, particularly for ensuring responsiveness in interactive environments.

However, RR is avoided as a sole algorithm due to its limitations, such as context switch overhead and potential drawbacks for long or I/O-bound processes. Other algorithms might be better suited for these situations. For instance, if there are long-running processes constantly arriving, shorter processes might suffer from starvation as they keep getting preempted before completing their execution, even if they have equal priority. RR does not take into account every characteristic of different processes, and it could fail to provide good performance even with high CPU utilization.

Team 08
s192865 KATOUZIAN Pouria
s201354 Gerard Manon

Feedback

- Difficulty: This project was significantly harder than past projects which we have done for other courses. We encountered numerous files containing uncommented functions, making comprehension a challenge. There was a significant disconnect between the course material and the project's difficulty level.
- Amount of work: It took us more than three weeks to understand the purpose of different files and their relationships, all in order to define distinct sub-problems. Each team member spent over four days coding due to the project's complexity and the numerous tasks required.
- Other: Requiring this type of project is unfair given our restricted time and lack of expertise. Such a project would be more appropriate for advanced operating system courses, intended for individuals with a deeper understanding of operating systems than we currently possess. This project demanded extensive testing to prevent memory leaks and unforeseen issues, which were time-consuming. However, the provided examples in the assignment were insufficient and very simple, as they failed to encompass all the requested algorithms (–age, preemptive, multiqueue and multi-core together,...).