CS475: Scheduler Simulation Code Review

# Questions to answer:

1. Who did you work with?
2. How is time defined?
3. How is a process modelled?
4. How are the different scheduling algorithms implemented?
5. How is multi-processor distribution accomplished?
6. Were there any bugs or issues that you fixed?
7. Explain any additional data structures.
8. What alternative data structures were considered? Why were they not chosen?
9. What alternative data structures would you have also considered?
10. Does the class hierarchy make sense? Are there other refactoring opportunities?
11. Is memory managed correctly (e.g. are pointers correctly deallocated)?
12. Are exceptions handled correctly?

|  |  |  |
| --- | --- | --- |
| Derek, Tyler, Colin | Scott | Bao, Chris, Cyruz |
| Alex, McKenna, Julia | Elisha, Geoff, Andrew |  |
| Aaron, LJ, Oscar | Zach, Zach, Drew |  |
| Kathleen, Andrew, Jarek | Marissa, Amanda, Kat |  |
| Brady, Kionte, Jason | Eli, Brodie, Clinton |  |

1. Alex Blair, McKenna Galle, and Julia Abbott reviewing Geoff, Elisha, and Andrew
2. This group created an operating system class that controlled time through an enum called scheduler. This enumerated value is then passed into each algorithm for execution. This value is the incrementation value as each process goes through a loop. This incrementation simulates the milliseconds of time that a process executes.
3. They created a process object which is sent through the scheduler to control CPU ticks and execution. Processes are modeled as a class object that has fields such as ID, bursts, and switch times. This allows the calculation of each statistic very carefully based on what information is needed.
4. Each algorithm is its own class and it inherits from the scheduler. This allows each algorithm to utilize the same definition and variables associated with time. It also in a way simulates each algorithm running on the same machine. Also, each algorithm does not need to have time defined within.
5. They used one function to run processes and used nested loops with a higher counter to simulate multiple core FCFS. So they could re-use code for the FCFS algorithm and adapt it to a multiprocessor simulation with a higher counter in the loop such as 8 for 4 cores.
6. One of the major issues occurred in FCFS when one process finished and had a very short I/O burst, it was put back into the queue, but was also still in the execution vector. Their solution was to remove it from the vector when it got close to finishing not when it was finished. Also an issue was adapting the single runProcess function to all different algorithms and multiprocessor approaches. Also, there was no destructor for the scheduler so that was addressed. Because of this there was a small memory leak.
7. Arrays and Vectors were used for execution. The Scheduler used a queue and a map was used for a process table. Each process is stored via a pointer to a location within this map.
8. A stack was considered to sort processes differently but, first in first sorting was required so a queue was chosen to be implemented as well as a map.
9. If utilizing the file system, a heap may have been applicable.
10. Followed class hierarchy well from the UML. Everything works from the Scheduler class. One thing is that FCFS and SPN look very similar so they could probably be merged. Multicore’s number of cores is only used in the FCFS algorithm so this could be expanded to be applicable for all other algorithms.
11. Pointers are properly deleted and memory is not leaked. Pointers are only used for processes and the scheduler.
12. Exceptions are handled through a try, catch block in main. Errors were fixed when they arose as opposed to try catch.