
DevNas

The Description of the Project

In this project, I built a simulator that simulates processor execution for processes and used multithreading to simulate how the CPU act, it will execute processes by their arrival time and their priority.

The scheduler, the clock, and the processors run on different threads, and if any thread depends on other threads it will wait for the other thread to continue working.

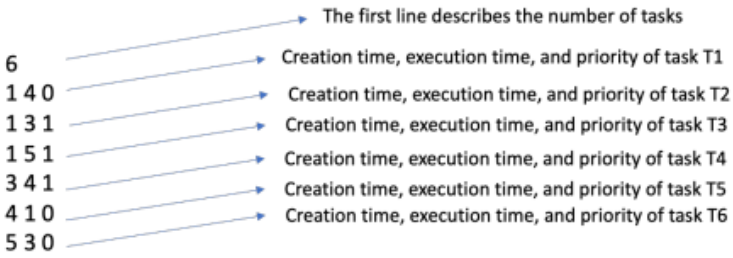
The Flow of the system

After the OS starts running, the tasks will be created, and each task has a different arrival time, execution time, and priority, every created task will be stored in the system queue by its priority waiting to be executed when it's turn.

The scheduler will choose the peak of the queue and assign it to the available processor to start execution, after the processor finishes with the task it will return to IDLE.

The system will be synchronized by the Clock, each cycle worth 1s.

Test Case



Assuming two processors and 10-cycle simulation, the below diagram describes one possible simulation output for the above input task file.

Clock cycle	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Processor P1	T3					T1				T5
Processor P2	T2			T4				T6		

Explanation:

- At cycle 1, tasks T1, T2, and T3 are created and added to the queue, then the schedular dequeues T2 and T3 and assigns them to processor P1 and P2, respectively. Note that the scheduler prioritized T3 and T2 over T1 because they have high priority. Also, note that it would have been also okay if T3 was assigned to P2 and T2 was assigned to P2, i.e., a task can be assigned to any processor.

• At cycle 2, no tasks were created and T3 and T2 are still using the processors. T1 is still waiting in the queue.

• At cycle 3, T4 is created and added to the queue. Therefore, waiting tasks are T1 and T4.

• At cycle 4, T5 is created and added to the queue. Therefore, waiting tasks are T1, T4 and T5. Moreover, T2 has finished execution and therefore processor P2 becomes available. The scheduler assigns T4 to P2 because it has the highest priority.

• At cycle 5, T6 is created and added to the queue. Therefore, waiting tasks are T1, T5 and T6.

• At cycle 6, T3 finished its execution and processor P1 becomes available. The scheduler assigns T1 to P1 because all waiting tasks have low priority but T1 has the longest execution time.

• At cycle 7, T1 and T4 are still using the processors, while T5 and T6 are waiting in the queue.

• At cycle 8, T4 finished its execution and processor P2 becomes available. The scheduler assigns T6 to P2 because all waiting tasks have low priority but T6 has the longest execution time.

• At cycle 9, T1 and T6 are using the processors, while T5 is waiting in the queue.

• At cycle 10, T1 finished its execution and processor P1 becomes available. The scheduler assigns T5 to P1 because it is the only task in the queue.

Classes

Main Class

Description

The main class is responsible for getting the number of processors and the max cycles from the user, and it starts the simulator class.

Simulator Class

Description

After this class starts, it will create instance of all the runnable classes and start their threads, then the class will be waiting to get notified by the clock class so it can add newly created

Tasks to the queue, at the end it will notify the Scheduler class to start assigning tasks.

Clock Class

Description

This class is responsible for notifying the simulator each cycle, each cycle will be 1 second by making the clock thread sleep 1 second.

Scheduler Class

Description

The scheduler class will be waiting to get notified by the simulator class, after it get notified the scheduler will start assigning tasks to the idle processors from the waiting queue.

Task Class

Description

This class represented the tasks that been executed by the processors, it contains creation time, execution time and priority. Task class implements Comparable interface that has compareTo method so it will be sorted after entering the priority queue.

Processor Class

Description

The processor class get tasks assigned by the scheduler, and each processor can run one task at

A time, so whenever the processor is running a task it can't get assigned to another task.

Conclusion

This was a quick explanation for my implementation, I included a folder with test cases to try the code yourself and a video of the code running, feel free to check it out.