OS Process Scheduler Simulation

Adam Cassidy | Operating Systems Spring 2022

System Requirements:

Standard C++ environment on any machine

Process Entity Support:

PCB Class : holds all critical information about a given process
proc_blocks[] : holds pointers to PCBs for every process

Multicore Process Distribution Approach:

I decided to go with a bit of a custom tweak to the Load Sharing approach. Instead of one large ready queue for all processes, I decided to implement a generated “priority level” for each process and have 3 queues: one for each priority level of high, medium, and low. These are just standard queues using FCFS priority within them.

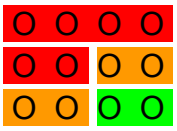
Each round robin cycle, half of the cores are assigned to high importance processes, one third are assigned to medium importance processes, and the remaining are assigned to low importance processes. See “Multilevel Queue Configuration”.

Uniprocessor Multiprogram Scheduling:

This project is entirely implemented to use the round robin technique to manage multiprogram scheduling. However, in order to simulate FCFS scheduling we can simply set the round robin reset time to the maximum cpu burst time.

Multilevel Queue Configuration:

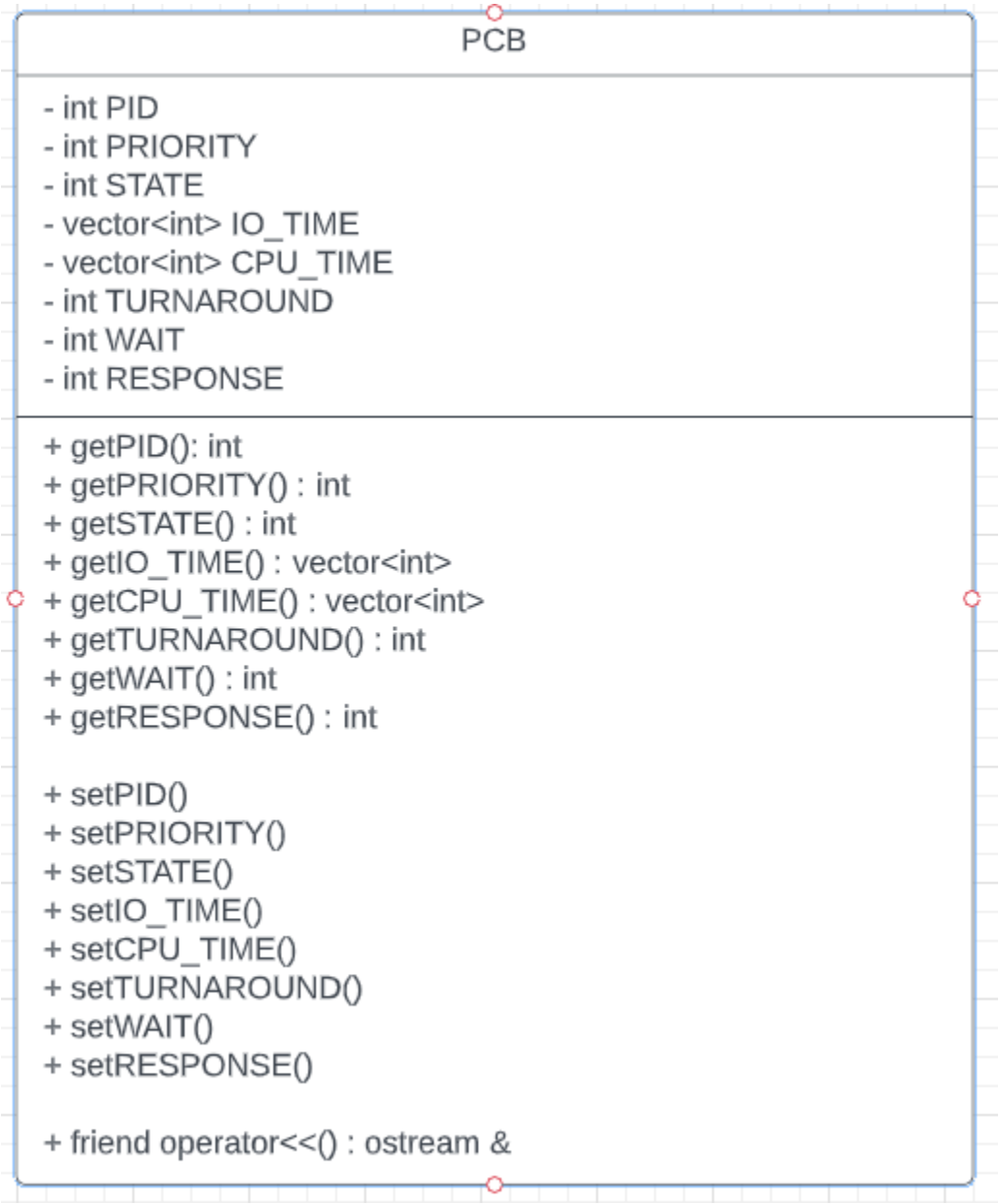
Consider this diagram with an example of 12 cores.
Available
Cores:



high priority queue of process ID’s : queue<int> high_queue;
medium priority queue of process ID’s : queue<int> med_queue;
low priority queue of processs ID’s : queue<int> low_queue;

Based on priority specified in PCB (initialized in generation), queues will be have a PID added onto them when a process “enters” that queue. The PCB state will also change to “ready”.

PCB UML:



General Timeline:

Before spring break: Decide on something simpler than initially planned.

During spring break: Lay out all details of how the sim will work and choose what data structures to use. Try to have rough working version before returning.

By due date: Finish analysis and find which algorithms work best.