Process Scheduling algorithms simulation

For your simulator, you will implement the following CPU scheduling algorithms:

- First Come First Serve
- Shortest Job First Preemptive
- Shortest Job First Non-Preemptive
- Priority Preemptive
- Priority Non-Preemptive
- Round Robin
- Multi-level queue
- Multi-level feedback queue

Process States

In our OS simulation, there are five possible states for a process, which are:

- NEW - The process is being created, and has not yet begun executing.
- READY - The process is ready to execute, and is waiting to be scheduled on a CPU.
- RUNNING - The process is currently executing on a CPU.
- WAITING - The process has temporarily stopped executing, and is waiting on an I/O request to complete.
- TERMINATED - The process has completed.

The Ready Queue

On most systems, there are a large number of processes, but only one or two CPUs on which to execute them. When there are more processes ready to execute than CPUs, processes must wait in the READY state until a CPU becomes available. To keep track of the processes waiting to execute, we keep a ready queue of the processes in the READY state.

Since the ready queue is accessed by multiple processors, which may add and remove processes from the ready queue, the ready queue must be protected by some form of synchronization--for this project, it will be a mutex lock.

Process Control Block

Think of a good way to implement PCB.

Scheduling Processes

schedule() is the core function of the CPU scheduler. It is invoked whenever a CPU becomes available for running a process. schedule() must search the ready queue, select a runnable process, and call the context_switch() function to switch the process onto the CPU.

NOTE : There is a special process, the idle process, which is scheduled whenever there are no processes in the READY state.

CPU Scheduler Invocation

There are few basic events which will cause the simulator to invoke scheduling algorithm:

1. **yield -** A process completes its CPU operations and yields the processor to perform an I/O request.
2. **wake_up -** A process that previously yielded completes its I/O request, and is ready to perform CPU operations. wake_up() is also called when a process in the NEW state becomes runnable.
3. **preempt -** When using a Round-Robin or Static Priority scheduling algorithm, a CPU-bound process may be preempted before it completes its CPU operations.
4. **terminate -** A process exits or is killed
5. **Context Switch –** Whenever any process come out of cpu temporarily after its turn like in case of round-robin.
6. **Idle** - The CPU scheduler also contains one other important function: idle(). idle() contains the code that gets by the idle process. In the real world, the idle process puts the processor in a low-power mode and waits.

Round Robin

- To specify a timeslice when scheduling a process, use the timeslice parameter of context_switch(). The simulator will automatically preempt the process and call your preempt() handler if the process executes on the CPU for the length of the timeslice without terminating or yielding for I/O.
- Run your Round-Robin scheduler with timeslices of 800ms, 600ms, 400ms, and 200ms. Compare the statistics at the end of the simulation. Show that the total waiting time decreases with shorter timeslices. However, in a real OS, the shortest timeslice possible is usually not the best choice. Why not?

Test Processes

For this simulation, we will use a series of processes, some CPU-bound and some I/O-bound. For simplicity, we have labelled each starting with a "C" or "I" to indicate CPU-bound or I/O-bound.

This is a sample input in case of priority scheduling algorithm.

PID	Process Name	CPU / I/O-bound	Priority	Start Time
0	Iapache	I/O-bound	8	0.0 s
1	Ibash	I/O-bound	7	1.0 s
2	Imozilla	I/O-bound	7	2.0 s
3	Ccpu	CPU-bound	5	3.0 s
4	Cgcc	CPU-bound	1	4.0 s
5	Cspice	CPU-bound	2	5.0 s
6	Cmysql	CPU-bound	3	6.0 s
7	Csim	CPU-bound	4	7.0 s

- For this project, priorities range from 0 to 10, with 10 being the highest priority. Note that the I/O-bound processes have been given higher priorities than the CPU-bound processes.
- Also, for demonstration purposes, the simulator executes much slower than a real system would. In the real world, a CPU burst might range from one to a few hundred milliseconds, whereas in this simulator, they range from 0.2 to 2.0 seconds.

- Be sure to update the state field of the PCB. The library will read this field to generate the Running, Ready, and Waiting columns, and to generate the statistics at the end of the simulation.
- Four of the five entry points into the scheduler (idle(), yield(), terminate(), and preempt()) should cause a new process to be scheduled on the CPU. In your handlers, be sure to call schedule(), which will select a runnable process, and then call context_switch(). When these four functions return, the library will simulate the process selected by context_switch().
- context_switch() should take a timeslice parameter, which is used for preemptive scheduling algorithms. Since FCFS is non-preemptive, use -1 for this parameter to give the process an infinite timeslice.