CSCI 340
Lecturer: Dr. Simina Fluture

Topics: CPU - Scheduling algorithms
NonPreemptive algorithms (FCFS, SJF, Priority)
Preemptive Algorithms (Priority, Shortest Remaining Time)

Read: Class Notes
Textbook: Scheduling Algorithms (6.3.1, 6.3.2, 6.3.3)

Criteria for making scheduling decisions:
I/O boundness of a process.
CPU boundness of a process.
process priority
past history of a process
- how often the process has been preempted
- elapsed time since its creation.
- predicted future behavior (the process will get blocked or not)

Waiting Time: the amount of time that the process spends waiting in the ready queue to use the processor.
Turnaround Time: the amount of the elapsed time between when a process is created and the moment that the process exits the running state by terminating.
Response Time: the amount of time it takes to start responding but not the time that it takes to output that response.

Goals:
- maximize the throughput. (# jobs/unit of time)
- maximize I/O utilization.
- maximize CPU utilization.
- minimize the response time.
- minimize turnaround time

Modes of scheduling:
nonpreemptive - processes are allowed to remain on the CPU until they terminate, block themselves
(FCFS, the nonpreemptive SJF, priority scheduling)

preemptive - the scheduler can forcibly remove a process from the CPU.

Nonpreemptive Scheduling Algorithms

First-Come First-Served (FCFS)

The ready queue contains the PCB of the process and it is treated like a FIFO queue.
Newly created processes and those becoming ready are added to the rear of the queue. The next process to be run is the one in front of the queue.

+) easy to understand and implement
-) the waiting time might be too long
possibility of the convoy effect

PID	Service Time	Arrival Time
1	3	0
2	6	2
3	4	4
4	5	6
5	2	8

Example:

Shortest-Job-First-Scheduling (SJF)

Suppose that the service time is known in advance.
The processes in the ready list, at the time when the CPU becomes available, are sorted in increasing order by the service time (next CPU burst) and scheduled for the CPU in that order.
The SJF may be either preemptive or non preemptive. A nonpreemptive algorithm will allow the currently running process to finish its CPU burst.

Example:

+) the SJF is proved to be the optimal scheduling algorithm with respect to the *average waiting time* of the ready processes.
SJF algorithm can be used for long term scheduling in a batch system by using as the length of the next CPU request the process' time limit that a user specifies when the job is submited.
-) SJF may penalize processes with high service time requests - starvation
SJF cannot be implemented at the level of short-term CPU scheduling. It is hard to predict the length of the next CPU burst.
One approach is to approximate SJF scheduling by predicting the value of the next CPU burst time.

Priority scheduling algorithms (6.3.3 from the book)

The processes are allocated to the CPU on the basis of their assigned priorities.
Priority scheduling can be either **preemptive** on **nonpreemptive**.
-) indefinite blocking or starvation.
Solution for the starvation of low-priority processes is *aging*.

Topics: Scheduling Preemptive algorithms
- **Round Robin**
- **Virtual Round Robin**
- **Multilevel Queue scheduling**
- **Multilevel Feedback Queue scheduling**

Preemptive Scheduling Algorithms:

In the discussion of nonpreemptive algorithms, the cost of context switching between processes is ignored. In preemptive scheduling systems, the cost of context switching can become a significant factor in computing optimal schedules.

SJF as a preemptive scheduling policy:

Preemptive SJF scheduling is sometimes called **shortest remaining time first**.
The process in the ready list whose remaining time to completion is the shortest is scheduled next.

Example:

Round-Robin Scheduling

A fixed quantum of time *Q* is set. A process is not allowed to use the CPU for a time larger than Q. If the CPU burst of a process exceeds Q, the process will be preempted.
Ready processes are maintained in a FIFO queue. When a process is preempted, it is placed at the rear of the queue. The process in front is scheduled next.
The basic premise is to equally distribute the processing time among all processes that request it.

Turnaround time depends on the size of the time quantum.
The **average waiting time** under the RR policy, is often quite long.

Setting the value of Q

I/O bounded processes
CPU bounded processes
Mix of CPU bounded processes with I/O bounded processes:

Multilevel Queue Scheduling

A multilevel queue-scheduling algorithm partitions the ready queue into separate queues.
Each separate queue has its own policy.
In addition there is a scheduling algorithm between queues implemented as a priority preemptive algorithm. Each queue has priority over lower-priority queues.

Another possibility is to time slice between the queues. Each queue gets a specific portion of the CPU time.

We can consider the interactive or the I/O processes having a higher priority than the CPU bounded processes.

Example: higher priority	system processes
	interactive processes
	I/O bounded process
lower priority	batch processes (CPU bounded processes)

Multilevel Feedback Queue Scheduling
allows a process to move between queues. It can be moved to a lower-priority queue (if the process uses too much of the CPU) or to a higher-priority (if the process has a large waiting time).
The following multilevel scheduling algorithm establishes a preference for shorter jobs. Scheduling is done on a preemptive basis, and a dynamic priority mechanism is used.

When a process first enters the system, it is placed in q0. When it returns to the Ready list, it is placed in ql. After each subsequent execution it is demoted to the next lower priority queue.
A short process will complete without migrating very far downward. Newer, shorter processes are favored over older, longer processes.
Within each queue, except the lowest priority one, a Round Robin policy is used. The lowest-priority queue is treated in a FCFS fashion.

There are many flavors that can be used in implementing a Multilevel Feedback Queue Scheduling algorithm.
Let's first define our policy:
We will consider a version that uses preemption in the same fashion as the Round Robin, at periodic intervals of time, and the preemption times vary according to the queue.
When a process first enters the system, it is placed in q0. When it returns to the Ready list, it is placed in ql. After each subsequent execution it is demoted to the next lower-priority queue.
Only when q0 is empty will the scheduler execute processes in ql.
If a new process arrives in q0 while a process is running in ql, the process in ql will be preempted ONLY AFTER it finishes its time quantum.