CPU Scheduling

CPU Scheduling

- It is the basis of multiprgorammed OS.
- We switch CPU among processes, therefore the computer is more productive.
- In a single-processor system, only one process can run at a time.
- Others must wait for CPU to be free.
- The aim (of multiprogramming) is to have some process running at all times.
    - No idleness for CPU.
    - Maximize CPU utilization
- How?
    - A process is executed until it must wait, typically for I/O.

How?

- By keeping several processes in memory, when it has to wait; the OS takes CPU away from that process and gives it to some other process.
- Every time a process has to wait, CPU is given to someone else.

CPU - I/O Burst Cycle

- Processes alternate between two states
  - CPU Burst and I/O Burst
- CPU Burst
  - When the process is using the CPU intensively.
  - Process performs computations or executes instructions.
- I/O Burst
  - Period where process is involved in I/O activities
  - R/W to storage, network communication, etc.
  - During this time, process is not utilizing CPU
  - Process waits for data to be read or written.
    - Longer than CPU bursts, because I/O operations are generally slower than CPU.

CPU burst - I/O Burst Cycle

- Process execution begins with CPU burst.
  - This is followed by an I/O burst.
  - Followed by CPU burst... and goes on.
  - Until the final CPU burst ends with a sys request to terminate execution.
- So, we are in a cycle.
- The durations of CPU bursts are different for each process, computer.
- If we know the type of program, we can choose an appropriate CPU-scheduling algorithm. (I/O bound vs CPU bound - many short burst vs long bursts)
  - OS schedules all resources. CPU is the main resource. It needs to be scheduled as well.

CPU Scheduler

- Whenever the CPU becomes idle, OS must select one of the processes in the *ready queue* to be excuted.
- This selection is carried out by the **short-term scheduler** (CPU scheduler).
    - Selects a process and allocates CPU to it.
- Ready queue is not necessarily a FIFO queue.
    - It can be a priority queue, tree or unordered linked list.

Preemptive Scheduling

- CPU Scheduling decisions may take place under these 4 circumstances:

  1 A process switches from running to waiting state.

  2 A process switches from running to ready state.

  3 A process switches from waiting state to ready state.

  4 A process terminates

- Apparently, after 1 and 4 a new process from the queue must be selected.
- However, what to do in 2 or 3?

Non-preemptive Scheduling

- When scheduling takes place because of 1 or 4, we say that the schedulimng is **nonpreemptive** or **cooperative**.
    - Otherwise, it is **preemptive**
- In **nonpreemptive**, once the CPU has been allocated to a process, the process keeps CPU until it releases it either by *terminating* or by switching to waiting state.
    - Used by Windows 3.x. (Very old)
- New operating systems are using *preemptive*.

Preemptive scheduling

- Used to manage the allocation of CPU to processes.
- OS have the ability to interrupt a currently running process and assign CPU to another process.
    - We didn't have this in nonpreemptive.
    - There, a process runs to either completion or yields control or blocked for I/O.
- However, preemptive scheduling can result in race conditions when data are shared among several processes.
- Consider two processes:
    - When one is updating the data, it is preempted so the second process can run.
    - Second process tries to read the data, but it is in an inconsistent state.

Dispatcher

- The module which gives control of the CPU to the process which is selected by the CPU scheduler.
- It control the context switch.
- Save the state of P_0, loads P_1.
- etc.

Scheduling Criteria

- Multiple CPU-scheduling algorithms.
  - They can favor one class of processes to another.
- There are several criteria to compare CPU-scheduling algorithms.
- CPU utilization
  - Keep the CPU as busy as possible
- Throughput
  - Number of processes that complete their execution per time unit
- Turnaround time
  - Amount of time to execute a particular process
  - The interval from the time of submission of a process to the time of completion.
  - Sum of the periods spend waiting to get into memory, waiting in ready queue, execution on CPU and doing I/O.

Scheduling Criteria

- Waiting time
  - Amount of time a process has been waiting in the ready queue
- Response time
- Amount of time it takes from when a request was submitted until the first response is produced.

Aim is to maximize CPU utilization and throughput and minimize turnaround, waiting and response time.

Scheduling Algorithms

- FCFS: First come First served
  - The process that requests the CPU first is the one who gets it first.
  - Can be implemented by a FIFO queue.
  - When a process enters the ready queue, its PCB is linked onto the tail of the queue.
  - When CPU is free, it is allocated to the process at the head of the queue.
- However, the average waiting time under this policy can be long.

FCFS

In this algorithm, the order they come in is very important. This is the first example.

This is the second.

The problem here is, at the 10th second, in the second order we would have 2 processes started and done. But in the first, we have only completed half of a process.

- This is an **nonpreemptive** algorithm.

Shortest-Job-First-Scheduling (SJF)

- This algorithm associates with each process the length of the process's next CPU burst.
- When the CPU is available, the process with smallest CPU burst is given to CPU.
  - If there are multiple with same length, FCFS is used.
- For example if we have 4 processes P_0, P_1, P_2 and P_3 with burst times $[6, 8, 7, 3]$ respectively;
  - They will be scheduled as:

SJF

- It can be either preemptive or nonpreemptive.
  - i.e. if the new process has a shorter burst time, it can preempt.
  - Shortest-Remaining-Time-First-Scheduling
- The hard part is to know the length of the next CPU request.
- Used frequently in *long-term scheduling*.
- In order to use it in *short-term scheduling* we need some way to find the length of CPU bursts.
  - We can estimate.
  - We can predict the value as an **expotnential average** of the measured lengths of previous CPU bursts.

Round-Robin Scheduling

- Designed especially for time-sharing systems.
- Similar to FCFS, but with preemption.
  - So now system can switch between processes.
- **Time quantum** or **time slice**
  - Generally from 10 to 100 milliseconds.
  - Ready queue is treated as a *circular queue*.
  - CPU goes around the queue and gives 1 time quantum to each process.

RR

- Ready queue is treated like FIFO queue
- New processes are added to the tail.
- CPU picks the first process, sets a timer to interrupt after 1 time quantum, and dispatches the process.
- 2 things can happen:
  - Process may have a CPU burst of less than 1 time quantum.
  - In this case, process will release the CPU voluntarily.
  - Scheduler will proceed to next process.

RR

- Or, the process may have a CPU burst of more than 1 time quantum.
- Timer will go off, and will cause an interrupt to OS.
- A context switch will be executed
- Process will be put at the *tail* of the ready queue.
- Scheduler will select the next.
- It is a preemptive algorithm.

RR

- The performance of RR depends on the size of time quantum.
- If it is extremely large, RR becomes FCFS.
- If extremely small, there are large number of context switches.
    - Creates overhead.
- We want the time quantum to be large with respect to context switch time.
    - If Context switch is 10 % of time quantum, it means 10 % of CPU time will be spent in context switch.
    - Usually smaller than 10 microseconds.
    - Quantum time is usually between 10-100 milliseconds.

RR

Let's say that we have three processes $[P_0, P_1, P_2]$ with burst times $[24, 3, 3]$ respectively.

- In this case, if we use a time quantum of 4 seconds, process P_1 gets 4 milliseconds.
- CPU is given to P_2. It doesn't need 4 seconds, it is done in 3.
- CPU is given to P_3. Again, done in 3.
- CPU returns to P_1.

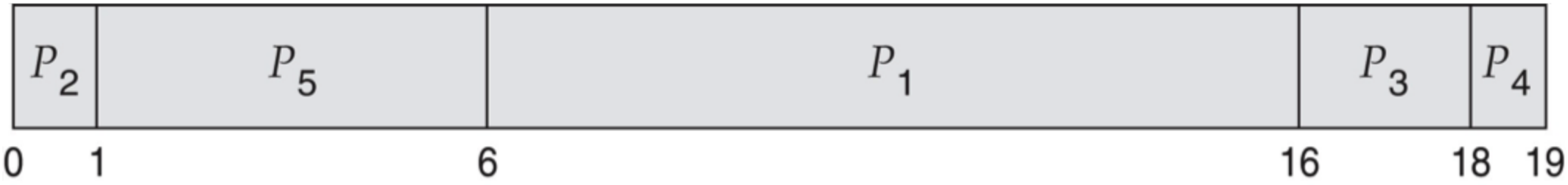
- The average waiting time can be long.

Priority Scheduling

- SJF algorithm is a special case of the general **priority-scheduling** algorithm.
- For each process, there is a priority and CPU is allocated to the process with the highest priority.
    - Those with equal priority, FCFS is used.
- Priorities are generally indicated by fixed range of numbers.
    - 0 to 7
    - 0 to 4000 etc.
- There is not a general consensus as to which one is 0. Highest or lowest?
    - Some uses one approach, some uses other.
- Can be preemptive or nonpreemptive.
    - If newly added has a higher priority, preemptive version will switch.

Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

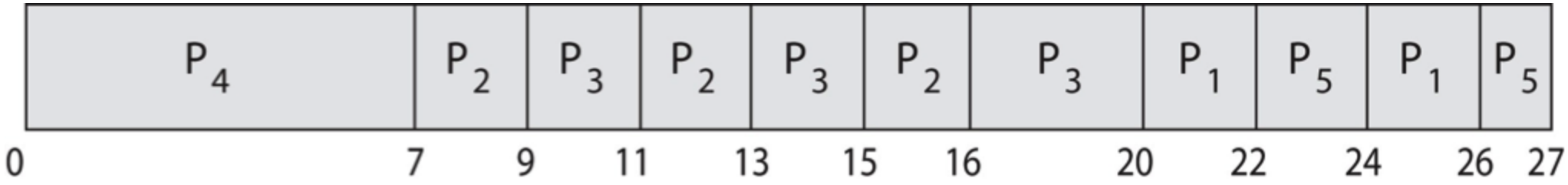


Priority Scheduling

- Problem can be **indefinite blocking** or **starvation**
- A process that is ready to run but waiting for CPU can be considered blocked.
- If all incoming processes have higher priority, a process can wait indefinitely.
- A solution to this is **aging**
  - It involves gradually increasing the priority of processes that wait for a long time.
  - If priorities range from 127 to 0, the priority of the waiting process can be increased 1 every 15 minutes.
  - Eventually it will get executed.

Priority Scheduling with RR

Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3



Multilevel queue

- There are another class of scheduling algorithms created for situations where processes are easily classifies into different groups.
    - For example **foreground** and **background**.
- These type of processes have different response-time requirements and may have different scheduling needs.
- A multilevel queue partitions the ready queue into several separate queues.
    - Processes are permanently assigned to one queue.
    - Each queue has its own scheduling algorithms
    - i.e. Foreground and background. Foreground might be scheduled with RR and the other can be FCFS.

Multilevel Queue

- However, there must be scheduling between the queues too.
- Commonly implemented as fixed-priority preemptive scheduling.
- Example:
    - System processes
    - Interactive processes
    - Interactive editing processes
    - Batch processes
    - Student processes
- Each queue has absolute priority over lower-priority queues. For example, no process in batch queue can run unless those above are empty.
- Even if a system process added to a queue while a batch process is working the system will preempt the batch process.

Multilevel Feedback Queue Scheduling

- In multilevel queues, processes cannot change queues.
- It has low scheduling overhead but inflexible.
- Here, it is possible.
- Idea is to separate processes based on their CPU bursts.
    - If a process uses too much CPU time, its moved to a lower-priority queue.
    - In addition, if a process waits too long in lower-priority queue, it can be moved to a higher-priority one.