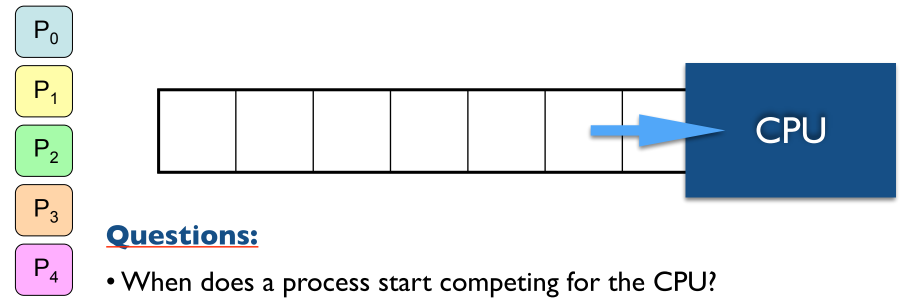
CPU Scheduling

CSCI 315 Operating Systems Design

Department of Computer Science

Notice: The slides for this lecture have been largely based on those accompanying the textbook *Operating Systems Concepts*, 9th ed., by Silberschatz, Galvin, and Gagne. Many, if not all, the illustrations contained in this presentation come from this source.

Basic Concepts



Questions:

- When does a process start competing for the CPU?
- How is the queue of ready processes organized?
- How much time does the system allow a process to use the CPU?
- Does the system allow for priorities and preemption?
- What does it mean to maximize the system's performance?

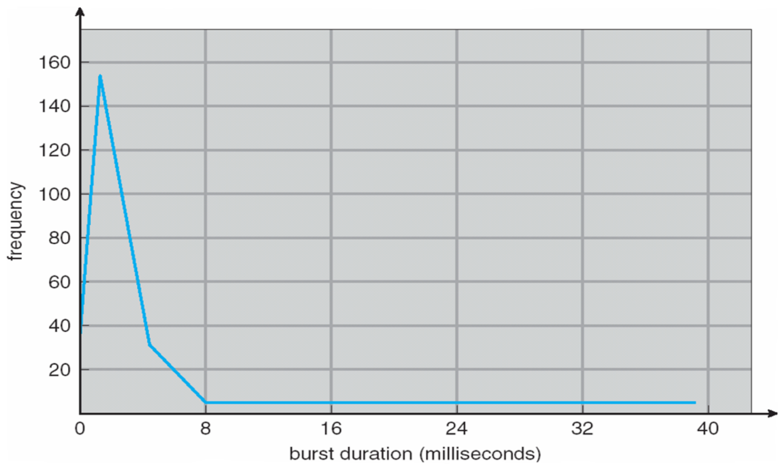
Basic Concepts

- You want to maximize **CPU utilization** through the use of multiprogramming.
- Each process repeatedly goes through cycles that alternate CPU execution (a CPU burst) and I/O wait (an I/O wait).
- Empirical evidence indicates that CPU-burst lengths have a distribution such that there is a large number of short bursts and a small number of long bursts.

Alternating Sequence of CPU and I/O Bursts

- Goal: maximize CPU utilization with multiprogramming
- Process execution consists of **cycles** of CPU execution and I/O wait
- A **CPU burst is** followed by an **I/O burst**
- The probability distribution of CPU bursts is an important concern

Histogram of CPU-burst Times

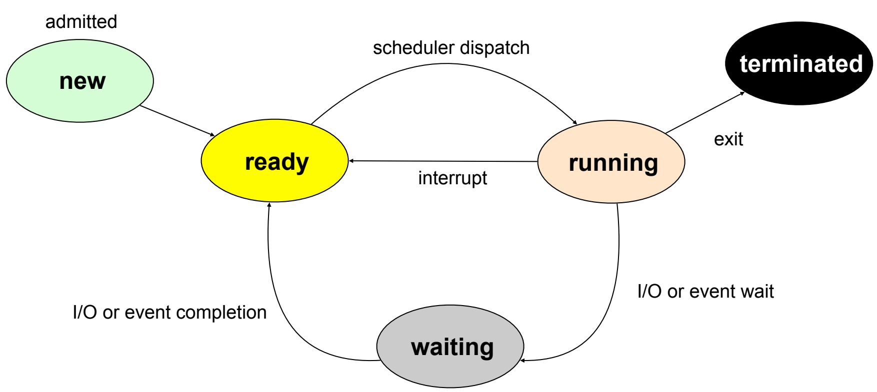


CPU Scheduler

- AKA *short-term scheduler*.
- Selects from among the processes in memory, which are ready queue and has the dispatcher give the CPU to one of them.

- The schedule needs to execute when a process:
  1. Switches from running to waiting state,
  2. Switches from running to ready state,
  3. Switches from waiting to ready,
  4. Terminates.

Process State Transition Diagram



Preemptive Scheduling ?

- In **cooperative** or **nonpreemptive** scheduling, when a process takes the CPU, it keeps it until the process either enters waiting state or terminates.

- In **preemptive scheduling**, a process holding the CPU may lose it. Preemption causes context-switches, which introduce overhead. Preemption also calls for care when a process that loses the CPU is accessing data shared with another process or kernel data structures.

Dispatcher

- The **dispatcher** module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context,
  - switching to user mode,
  - jumping to the proper location in the user program to restart that program.
- The **dispatch latency** is the time it takes for the dispatcher to stop one process and start another running.

Scheduling Criteria

These are **performance** metrics such as:

- **CPU utilization** – high is good; the system works best when the CPU is kept as busy as possible.
- **Throughput** – the number of processes that complete their execution per time unit.
- **Turnaround time** – amount of time to execute a particular process.
- **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, **not** output (for time-sharing environment).

It makes sense to look at the **averages** of these metrics.

Optimizing Performance

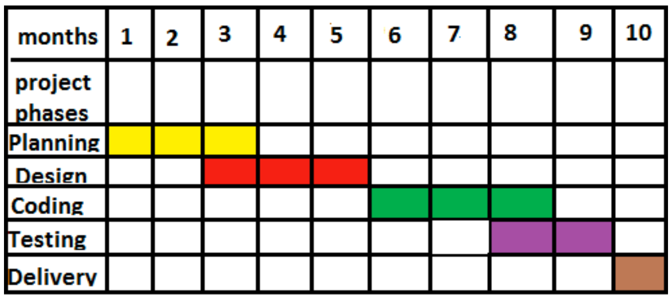
- **Maximize** CPU utilization
- **Maximize** throughput

- **Minimize** turnaround time
- **Minimize** waiting time
- **Minimize** response time

Scheduling Algorithms

Gantt Chart

A Gantt chart is **a type of bar chart** that illustrates a project schedule (circa 1910). Modern Gantt charts also show the dependency relationships between activities and current schedule status.



https://en.wikipedia.org/wiki/Gantt_chart

First-Come, First-Served (FCFS)

Process	Burst Time
P_1	24
P_2	3
P_3	3

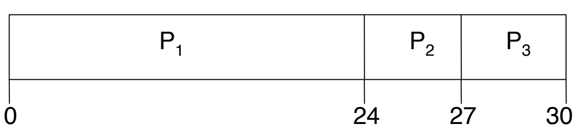
- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The **Gantt Chart** for the schedule is:

- Waiting times: ?
- **Average waiting time:** ?

First-Come, First-Served (FCFS)

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The **Gantt Chart** for the schedule is:



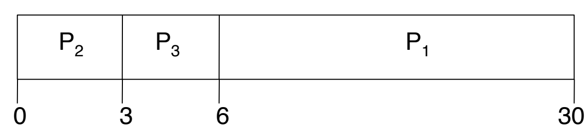
- Waiting times: $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- **Average waiting time:** (0 + 24 + 27) / 3 = 17

FCFS

Suppose that the processes arrive in the order

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6$; $P_2 = 0$, $P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case.
- **Convoy effect:** all process are stuck waiting until a long process terminates.

Shortest-Job-First (SJF)

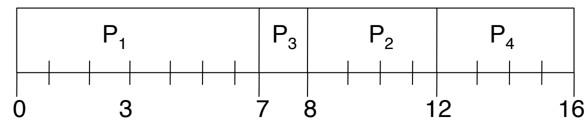
- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - **Nonpreemptive** – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - **Preemptive** – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the ***Shortest-Remaining-Time-First (SRTF)***.
- SJF is **optimal** – gives minimum average waiting time for a given set of processes.

Question: Is this practical? How can one determine the length of a CPU-burst?

Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

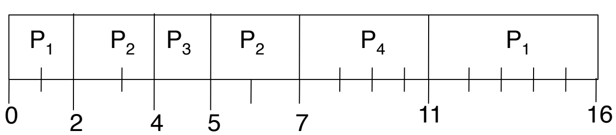


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

Determining Length of the Next CPU-Burst

Sorry, no crystal ball.

But… we can use some smart math!

Determining Length of the Next CPU-Burst

- We can only ***estimate*** the length.
- This can be done by using the length of previous CPU bursts, using exponential averaging:

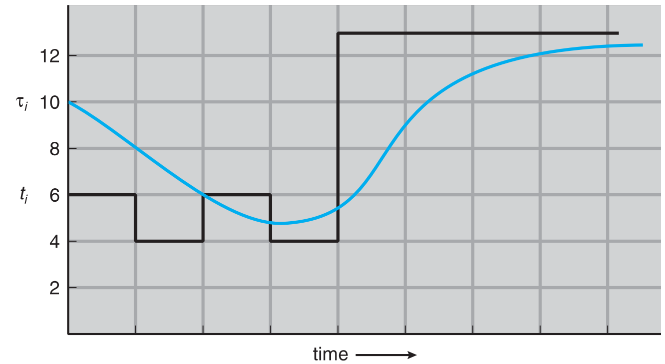
t_n = measured length of the n^{th} CPU burst

T_n = estimated length of the n^{th} CPU burst

$$T_{n+1} = \alpha \, t_n + (1-\alpha) \, T_n$$

α = weight value, where $0 \leq \alpha \leq 1$

Prediction of the Length of the Next CPU-Burst



CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

Priority Scheduling

- A priority number (integer) is associated with each process.
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive
- SJF is a priority scheduling where priority is the predicted next CPU-burst time.
- Problem: **Starvation** – low priority processes may never execute.
- Solution: **Aging** – as time progresses increase the priority of the process.

Round Robin (RR)

- Each process gets a small unit of CPU time (time ***quantum***), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Performance:
  - q too large $\Rightarrow$ devolves into FCFS
  - q *too small* $\Rightarrow$ excessive context switching; q must be large with respect to context switch, otherwise overhead is too high

RR with Time Quantum = 20

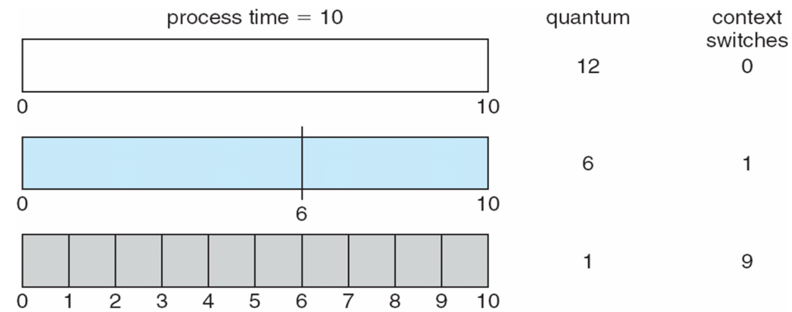
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3	
0	20	37	57	77	97	117	121	134	154	162

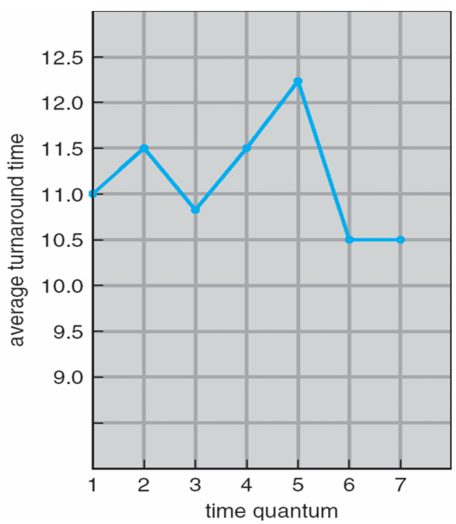
- Typically, higher average turnaround than SJF, but better *response.*

Time Quantum and Context Switch Time



Question: What considerations influence the choice of value for the quantum?

Turnaround Time Varies with the Time Quantum

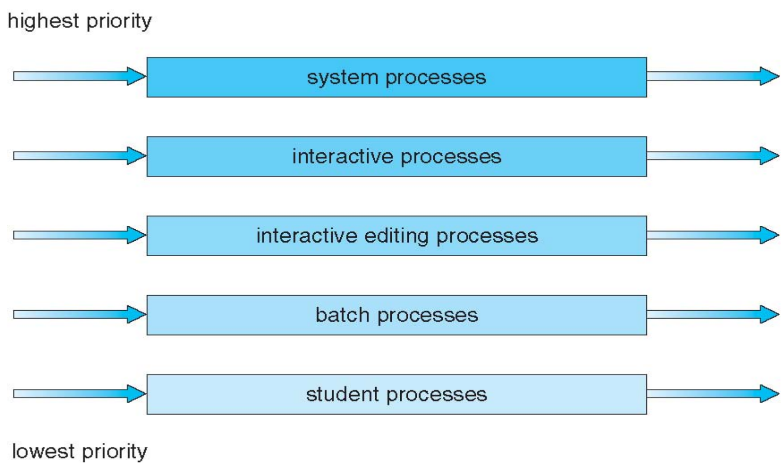


process	time
P_1	6
P_2	3
P_3	1
P_4	7

Multilevel Queue

- Ready queue is partitioned into separate queues:
  - foreground (interactive)
  - background (batch)
- Each queue has its own scheduling algorithm.
  - foreground: RR
  - background: FCFS
- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR.
  - 20% to background in FCFS .

Multilevel Queue Scheduling



Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way.
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues,
  - scheduling algorithms for each queue,
  - method used to determine when to upgrade a process,
  - method used to determine when to demote a process,
  - method used to determine which queue a process will enter when that process needs service.

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – time quantum 8 milliseconds
  - Q_1 – time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1.
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.

Multilevel Feedback Queues