toc:

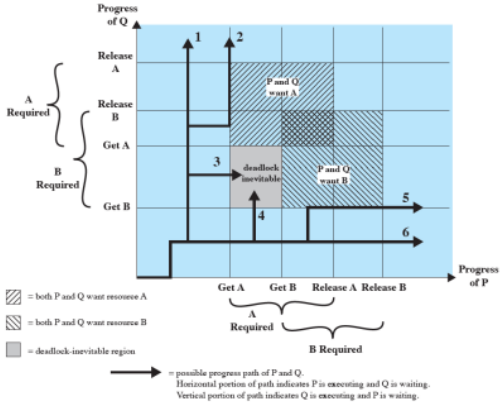
Where we are...

- Computer System Overview
- Operating System Structures
- Process Description and Control
- Threads
- Synchronization and Mutual Exclusion
- Deadlock and Starvation
- CPU Scheduling

Recap

- **All four conditions must hold** for a deadlock to occur.

Possibility of Deadlock	Existence of Deadlock
1.Mutual exclusion 2.No preemption 3.Hold and wait	1.Mutual exclusion 2.No preemption 3.Hold and wait 4.Circular wait



Generally speaking, we can deal with the deadlock problem in one of three ways:

- Ensure that the system will *never* enter a deadlock state:
  - Deadlock prevention.
  - Deadlock avoidance.
- Allow the system to enter a deadlock state, detect it, and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system.
  - Used by most OSes including Linux and Windows.

Basics

Kernel threads are ht eones being scheduled.

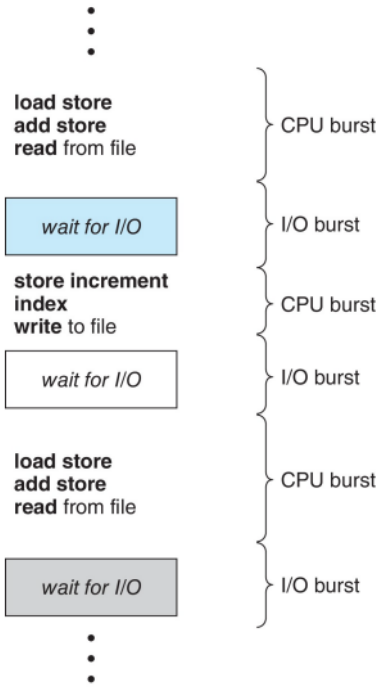
“Process scheduling” and “thread scheduling” are often used interchangeably.

We say CPU to mean CPU core as a computation unit

Maximum CPU utilization obtained with multiprogramming

CPU-I/O Burst Cycle

- Process execution consists of a cycle of CPU execution and I/O wait.
- Process execution begins with a CPU burst followed by an I/O burst, and so on.
  - CPU burst: Scheduling process state in which the process executes on CPU.
  - I/O burst: Scheduling process state in which the CPU performs I/O.

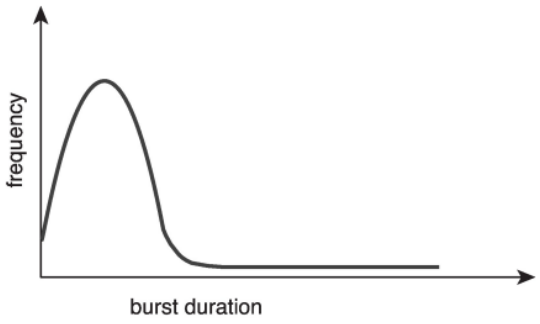


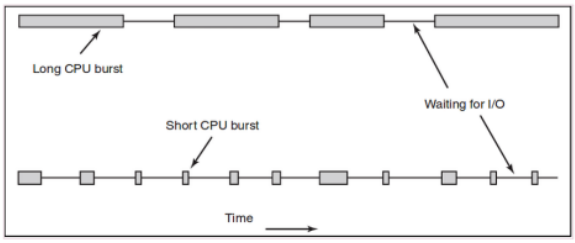
We alternate working (cpu) and wait (i/o) bursts.

We end on an I/O burst as the process is ending

Histogram of CPU-burst Times

- CPU burst distribution is of main concern.
  - Large number of short bursts: An I/O-bound program typically has many short CPU bursts.
  - Small number of longer bursts: A CPU-bound program might have a few long CPU bursts.





I/O and CPU bound comparison.

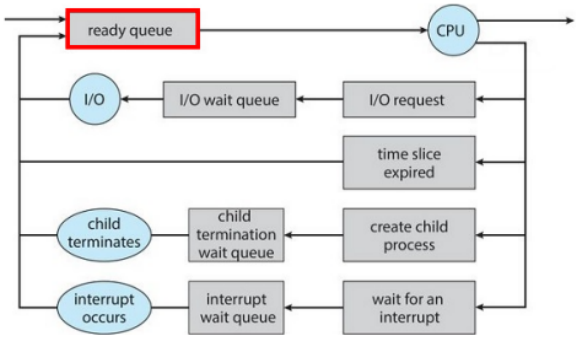
On the histogram:

- above is cpu-bound
- below is i/o bound

CPU Scheduler

- The **CPU scheduler** selects from among the processes in ready queue, and allocates a CPU core to one of them.
  - Queue may be ordered in various ways.

- The records in the queues are generally process control blocks (PCBs) of the processes.



Queueing-diagram representation of process scheduling.

Reminder: the whole process itself isn't in the queue, just the pcb (which has all the info and points to where the process actually is)

Preemptive and Nonpreemptive Scheduling

- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state (I/O or child process)
  2. Switches from running to ready state (interrupts)
  3. Switches from waiting to ready (I/O completion or child finish)
  4. Terminates (last instructions)

- If only do 1 and 4, the scheduling is **nonpreemptive**, or cooperative:
  - Once the CPU has been allocated to a process, the process keeps the CPU until it releases it either by terminating or by switching to the waiting state.

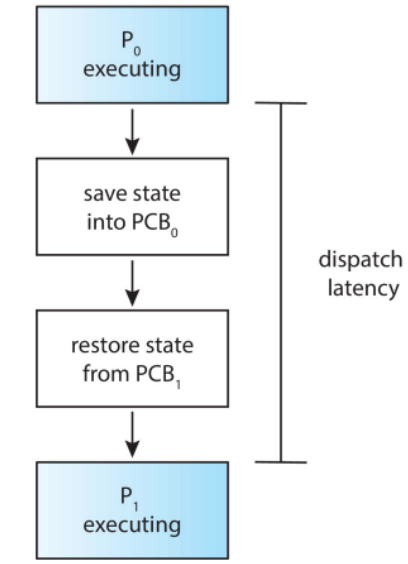
- All other scheduling is **preemptive** (all modern OSs use this).

Preemptive scheduling makes decisions at all these times.

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the CPU scheduler. It involves:
  - Switching context.
  - Switching to user mode.
  - Jumping to the proper location in the user program to resume that program.

- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running.



The role of the dispatcher.

Knowledge Check

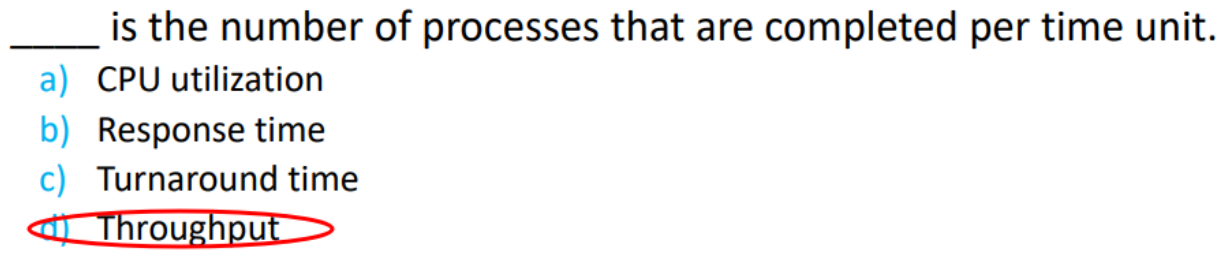
- which of the following is true of cooperative scheduling
  - b. A process keeps the CPU until the releases the CPU either by terminating or by switching to the waiting state.
- In preemptive scheduling, the sections of code affected by interrupts must be guarded from simultaneous use
  - True
  - we want to ensure mutual exclusion
  - we want to make sure no other process is accessing at the same time

Scheduling Criteria

- cpu util
  - keep as busy as possible
  - maximize this
- throughput
  - num of processes that complete their execution per time unit
  - maximize this
- turnaround time
  - amount of time to execute a particular process
  - minimize this
- waiting time
  - aggregate amount of time a process has been waiting int eh ready queue
  - will be affected by scheduling algo
  - minimize this
- response time
  - time between request submission and first response for interactive systems
  - minimize this

All numbers in milliseconds (ms) unless stated otherwise.

For now we assume we have 1 cpu.



Scheduling Algos

Algorithms to decide which of the processes in the ready queue is to be allocated to the CPU's core:

1. First-Come, First-Served Scheduling (FCFS)
2. Shortest-Job-First Scheduling (SJF)
3. Round-Robin Scheduling (RR)
4. Priority Scheduling
5. Multilevel Queue Scheduling
6. Multilevel Feedback Queue Scheduling

We will be using a Gantt chart.

- A bar chart that illustrates a particular schedule, including the start and finish times of each of the participating processes.
- All times are in ms unless otherwise specified.

First- Come, First-Served (FCFS) Scheduling

Process	Burst Time
P1	7
P2	3
P3	12
P4	5
P5	9

- Suppose processes arrive in the order P_1 , P_2 , P_3 , P_4 , P_5, then the Gantt Chart for the FCFS is:

we're assuming these all arrived at (more or less) the same time since there is no `arrival time` column in the chart. We assume in order of number just because it's convenient here.

A nonpreemptive algorithm:

- Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O.

FCFS can be easily implemented with a FIFO queue.

different orders will result in different wait times but FCFS can't account for that

Suppose processes arrive in the order: P_1 , P_2 , P_3.

Process	Burst Time
P_1	24
P_2	3
P_3	3

Waiting time for P_1 = 0; P_2 = 24; P_3 = 27

Average waiting time: (0 + 24 + 27)/3 = 17

Suppose that the processes arrive in the order: P_2 , P_3 , P_1

The Gantt chart for the schedule is:

Process	Burst Time
P_1	24
P_2	3
P_3	3

Waiting time for P_1 = 6; P_2 = 0, P_3 = 3

Average waiting time: (6 + 0 + 3)/3 = 3

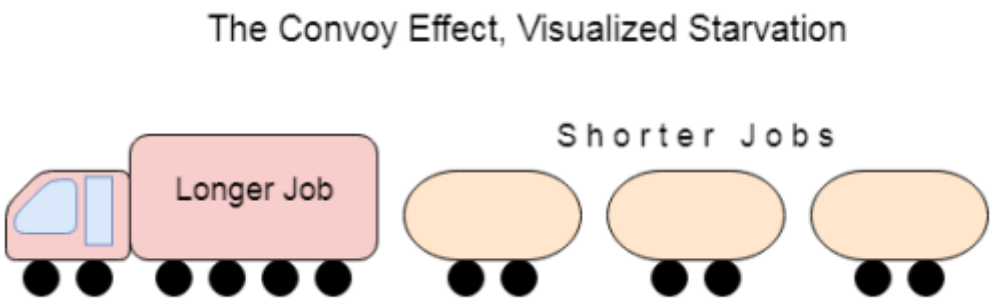
Much better than previous case!

Convoy effect:

- short process behind long process
- consider one cpu-bound and many i/o-bound processes

FCFS is not good for interactive systems

- important that each process get a share of the cpu at regular intervals



Prof stops at slide 27.

Multi-Processor Scheduling

Real-Time CPU Scheduling

Algo Eval