University of Nottingham
UK | CHINA | MALAYSIA

Operating Systems and Concurrency

Lecture 4: Process 2

University of Nottingham, Ningbo China
2024

Recap

Last Lecture

- Processes have **“control structures”** associated with them (process control blocks and process tables)
- Processes can have different **states** and **transition** between them (e.g.new, ready, running, blocked, terminated)
- The operating system maintains multiple **process queues** (e.g. ready queue, event queues, etc.)
- The operating system **manages processes** on the user’s behalf (e.g. fork(), exit(), . . .)

Goals for Today

Overview

- Introduction to **process scheduling**
- Types of **process schedulers**
- **Evaluation criteria** for scheduling algorithms
- Typical **process scheduling algorithms**

Process Scheduling

Driving Factors

- The OS is responsible for managing and scheduling processes
- The aim of process scheduling is to assign processes to be executed by the processor or processors over time, with good response time, throughput, and processor efficiency.
- Process scheduling in an operating system is driven by factors such as priority, CPU utilization, throughput, turnaround time, waiting time, response time, fairness, the nature of the process (CPU-bound vs. I/O-bound), scheduling type (preemptive vs. non-preemptive), and system load.

Process Scheduling
Priority

- **Priority scheduling** involves assigning different priorities to processes, where the CPU is allocated to the process with the highest priority.
- This ensures that critical or time-sensitive tasks are handled promptly.
- Priorities can be assigned by the system or user and may be dynamic (changing based on process behavior or system state).

Process Scheduling
CPU Utilization

- The goal is to keep the CPU as busy as possible.
- Higher CPU utilization is generally desirable as it ensures that the processing power of the CPU is being used efficiently.
- Scheduling algorithms aim to reduce idle CPU time by optimizing process execution order.

Process Scheduling Throughput

- Throughput refers to the number of processes completed in a given time frame.
- Scheduling should maximize throughput by minimizing process completion time, ensuring that more processes are completed efficiently.

Process Scheduling
Turnaround Time

- Turnaround time is the **total time** taken from the submission of a process to its completion.
- Scheduling algorithms try to minimize turnaround time to improve system responsiveness and user satisfaction.
- It includes waiting time, processing time, and any time spent waiting for I/O operations.

Process Scheduling
Waiting Time

- This is the total time a process spends in the ready queue waiting to be executed.
- Reducing waiting time is crucial for efficient scheduling as prolonged waits can lead to resource underutilization and poor performance.
- Scheduling strategies aim to minimize this for improved system performance.

Process Scheduling
Response Time

- Response time measures the time from when a process is submitted until the first response or output is produced.
- This is important in interactive systems where users expect immediate feedback.
- Scheduling algorithms for interactive systems (like Round Robin) aim to minimize response time to ensure a responsive user experience.

Process Scheduling
Fairness

- A fair scheduling algorithm ensures that no process is starved of CPU time and that all processes receive equitable time for execution.
- Fairness becomes especially important in multi-user or multi-tasking systems, where different users or applications might compete for resources.

Process Scheduling

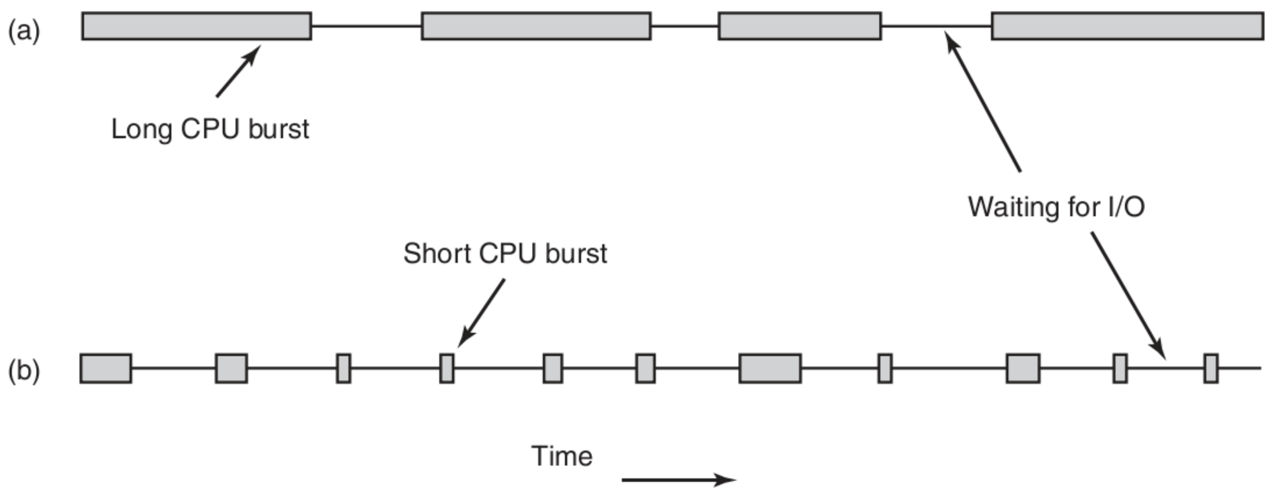
Nature of the Process

- **CPU-Bound Processes:** processes spend most of their time performing computations on the CPU. They have long CPU bursts, meaning they need a lot of CPU time before requesting I/O.
  - Example: A scientific computation or a large data processing task.
- **I/O-Bound Processes:** processes spend most of their time waiting for I/O operations to complete, using very little CPU time. They have short CPU bursts.
  - Example: Reading from a file, printing documents, or interacting with a user interface.
- **Alternating Bursts:** Processes typically alternate between bursts of CPU activity (computation) and I/O requests (like reading or writing data).

Process Scheduling

Nature of the Process

- Scheduling algorithms aim to balance between CPU-bound and I/O-bound processes.
- I/O-bound processes are usually given preference because they spend less time on the CPU, allowing for better CPU utilization during I/O waits.



Process Scheduling

System Load

- The scheduling algorithm must adapt based on the current system load
  - (e.g., the number of processes waiting to be executed). During high loads, scheduling might prioritize faster execution or processes that can free resources quickly.

Process Scheduling

Classification by Time Horizon

- Process scheduling is categorized into
  - Long-term,
  - Medium-term, and
  - Short-term scheduling,
- Each with distinct roles to optimize system performance and resource utilization.

Process Scheduling

Classification by Time Horizon

- Long term scheduling:
  - Applies to new processes and controls the degree of multiprogramming
  - Decide which processes to admit to the system
  - A good mix of CPU and I/O bound processes is favorable to keep all resources as busy as possible.
  - Happens infrequently, as it's only triggered when a new process is created or when the system's load changes significantly.

Process Scheduling

Classification by Time Horizon

- Medium term scheduling:
  - Controls swapping and the degree of multi-programming in main memory. (Memory management)
  - Occurs when the system is under memory pressure or when it’s necessary to adjust the degree of multiprogramming.
  - Efficient memory utilization, maintaining system responsiveness.

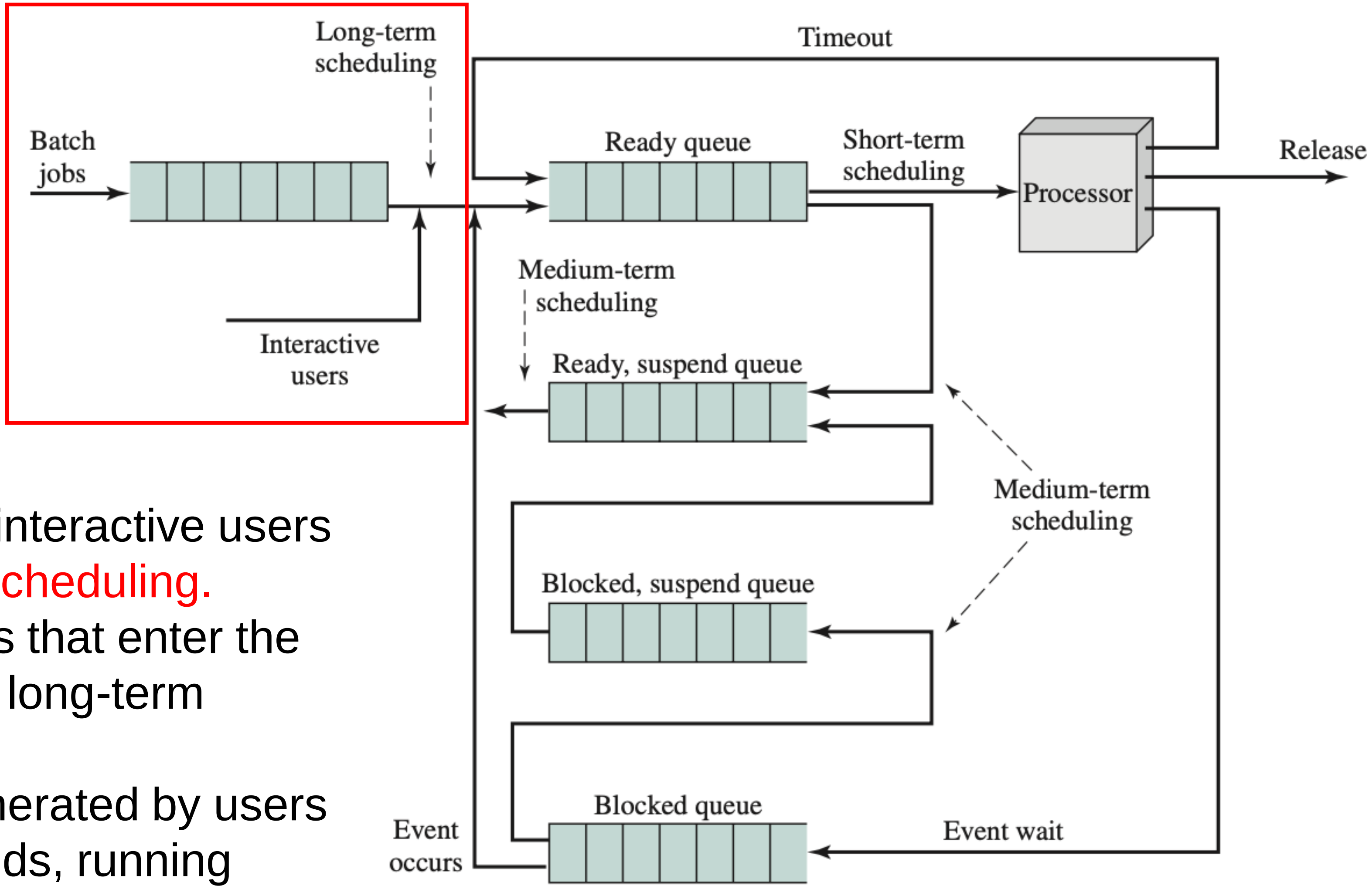
Process Scheduling

Classification by Time Horizon

- Short term scheduling (dispatcher) :
  - Manages the ready queue
  - Decide which process to run next
  - It is invoked in response to various events like:
    - Clock interrupts (e.g., when a time slice expires in preemptive systems),
    - I/O interrupts (e.g., when an I/O operation completes),
    - System calls (e.g., when a process makes a blocking call),
    - Semaphore operations (e.g., signaling when resources become available).
  - Invoked very frequently, hence must be fast
  - Minimizes response time, ensures CPU efficiency, fast dispatching.

Process Scheduling

Classification by Time Horizon (Cont.,)



- New processes from batch jobs and interactive users enter the system through long-term scheduling.
  - Batch Jobs: These are processes that enter the system as jobs, typically through long-term scheduling.
  - Interactive Users: Processes generated by users in real-time (e.g., typing commands, running interactive applications).

Figure. Queueing diagram for scheduling

Process Scheduling

Classification by Time Horizon (Cont.,)

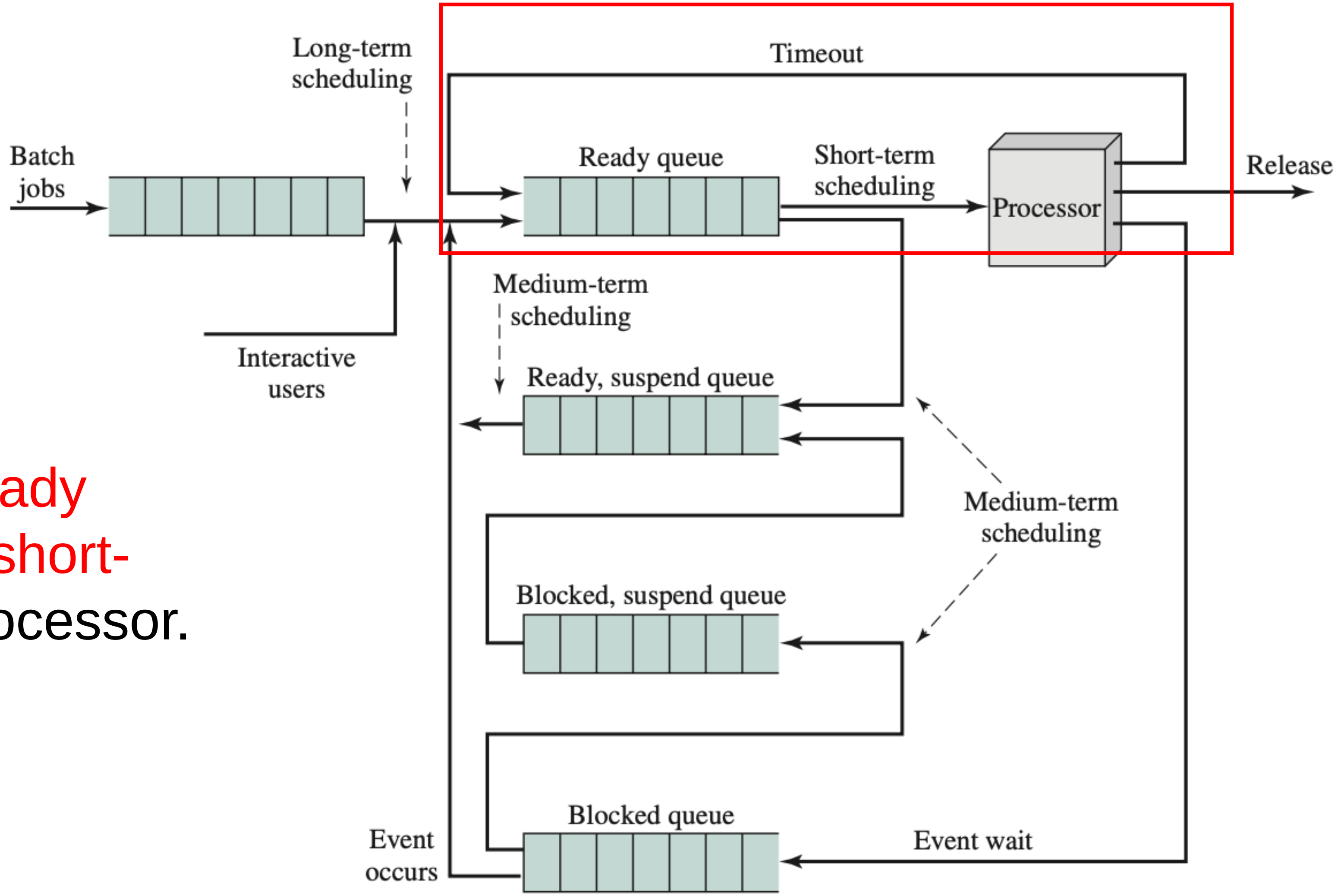


Figure. Queueing diagram for scheduling

- Processes move through the ready queue and are selected by the short-term scheduler to run on the processor.

Process Scheduling
Classification by Time Horizon (Cont.,)

- Processes may move between the blocked, suspended, and ready queues based on I/O events or memory availability, with the medium-term scheduler managing the swapping of processes in and out of main memory.

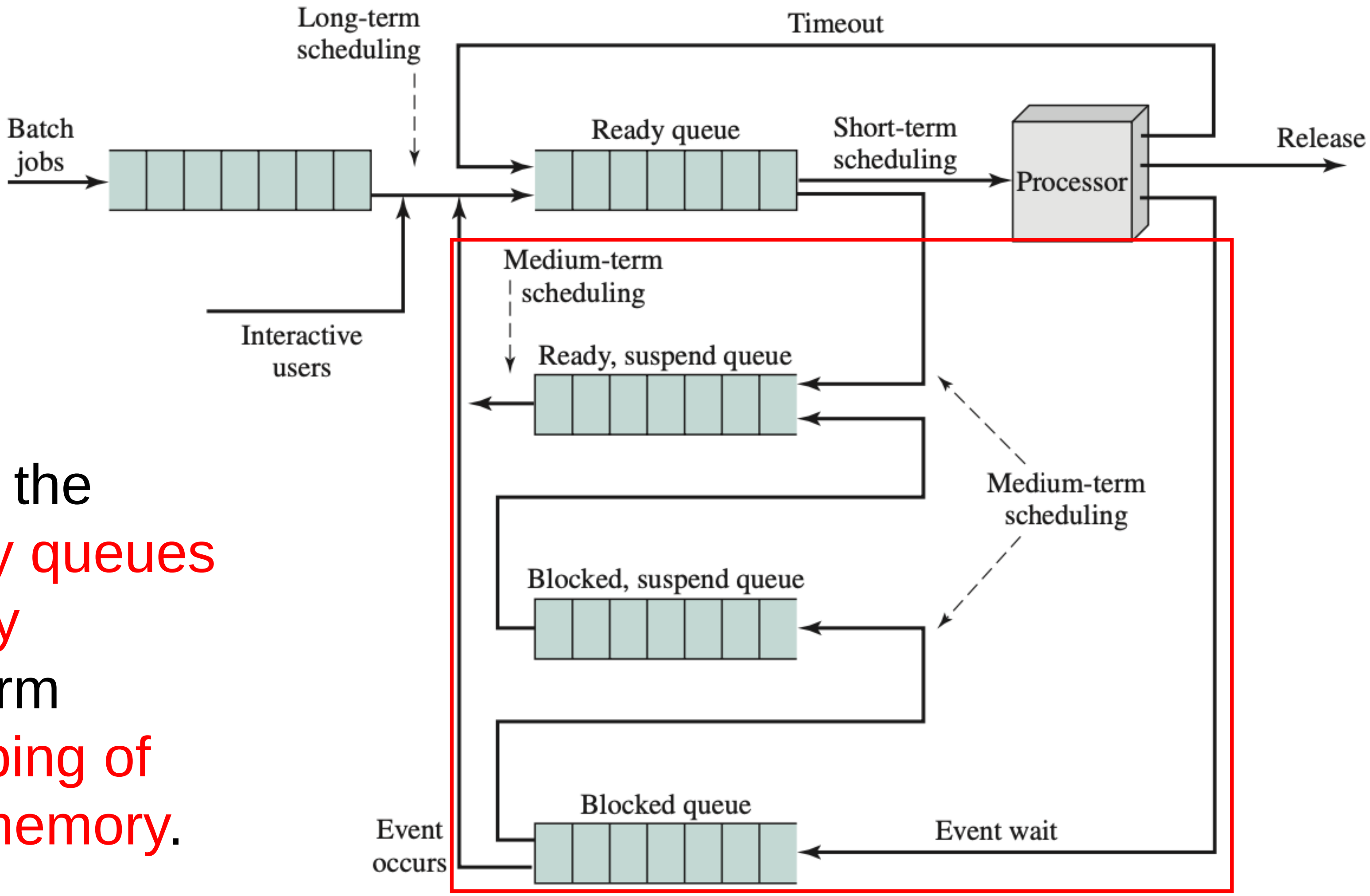


Figure. Queueing diagram for scheduling

Process Schedulers
Classification by Approach

- Process scheduling can be classified into preemptive and non-preemptive based on how processes are managed when they are running on the CPU.
  - **Non-preemptive**: processes are only interrupted **voluntarily** (e.g., I/O operation or “nice” system call – yield())
    - Windows 3.1 and DOS were non-preemptive
  - **Preemptive**: processes can be **interrupted forcefully**
    - E.g. preemptive scheduling algorithm picks a process and lets it run for a maximum of some fixed time. (clock interrupt)
    - This requires context switches, which generate **overhead**, too many of them should be avoided
    - Prevents processes from **monopolizing the CPU**
    - **Most popular** modern operating systems are preemptive.

Performance Assessment
Criteria

- **User oriented criteria**:
  - **Response time**: minimize the time between creating the job and its first execution
  - **Turnaround time**: minimize the time between creating the job and finishing it
  - **Predictability**: minimize the variance in processing times
- **System oriented criteria**: (focus on **efficient utilization** of processor)
  - **Throughput**: maximize the number of jobs processed per hour
  - **Fairness**:
    - Are processing power/waiting time equally distributed?
    - Are some processes kept waiting excessively long (**starvation**)
- Evaluation criteria can be **conflicting**, i.e., **reducing the response time** may increase context switches and may **worsen the throughput** and **increase the turn around time.**

Scheduling Algorithms

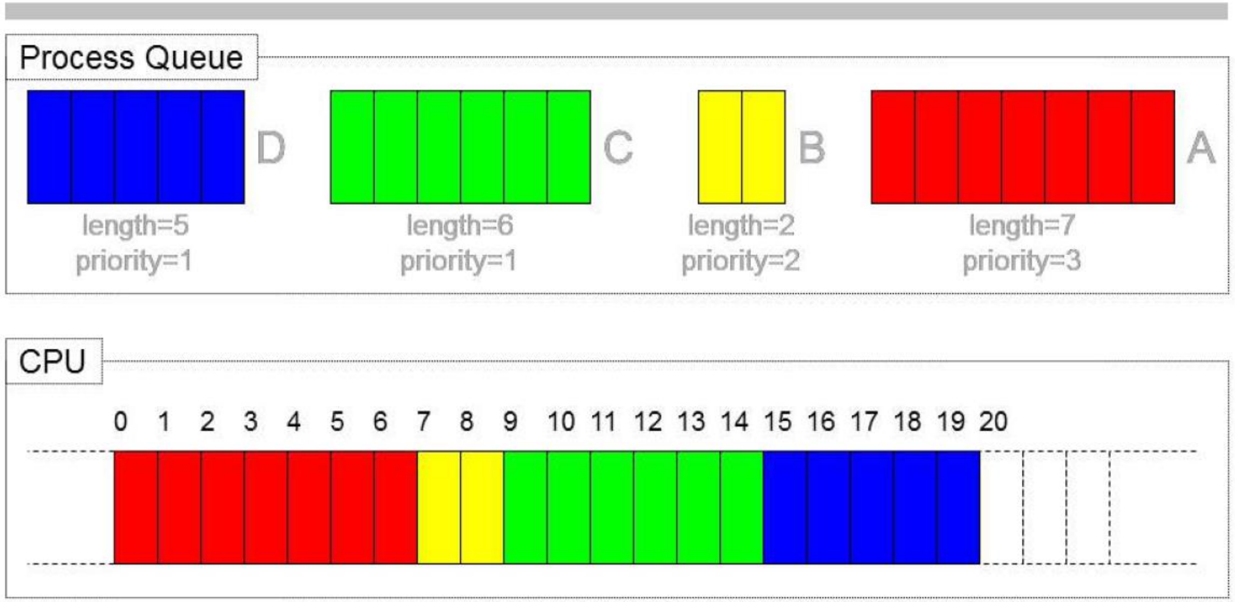
Overview

- **Algorithms** considered:
  - 1 First Come First Served (**FCFS**)(**Batch System**)
  - 2 **Shortest job first** (**Batch System**)
  - 3 **Round Robin (Interactive System)**
  - 4 **Priority queues (Interactive System)**

- Performance measures used:
  - **Average response time**: the average of the time taken for all the processes to start
  - **Average turnaround time**: the average time taken for all the processes to finish

Scheduling Algorithms
First Come First Served

- A **non-preemptive algorithm** that operates as a **strict queueing mechanism** and schedules the processes in the same order that they were added to the queue.



- Average response time = $0 + 7 + 9 + 15 = \frac{31}{4} = 7.75$
- Average turn around time = $7 + 9 + 15 + 20 = \frac{51}{4} = 12.75$

Scheduling Algorithms

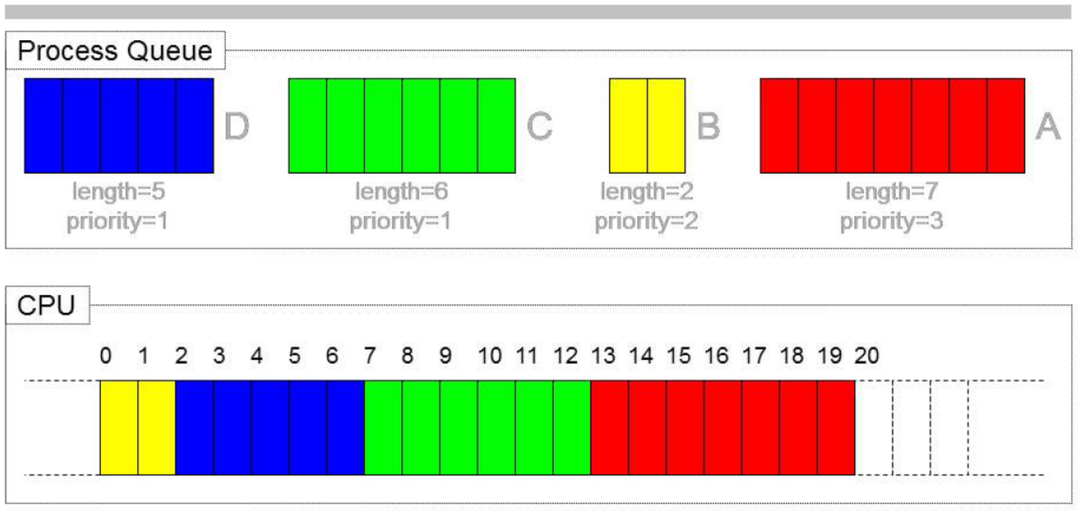
First Come First Served (Cont.,)

- Advantages: **positional fairness** and easy to implement
- Disadvantages:
  - **Favours long processes** over short ones (think of the supermarket checkout!)
  - **Favours CPU bounds processes** over I/O bounds ones.

Process	Arrival Time	Service Time (T_s)	Start Time	Finish Time	Turnaround Time (T_r)	T_r/T_s
W	0	1	0	1	1	1
X	1	100	1	101	100	1
Y	2	1	101	102	100	100
Z	3	100	102	202	199	1.99
Mean					100	26

Scheduling Algorithms
Shortest Job First

- A **non-preemptive algorithm** that starts processes in order of **ascending processing time** using a provided/known estimate of the processing
- Advantages: always result in the **optimal turn around time**



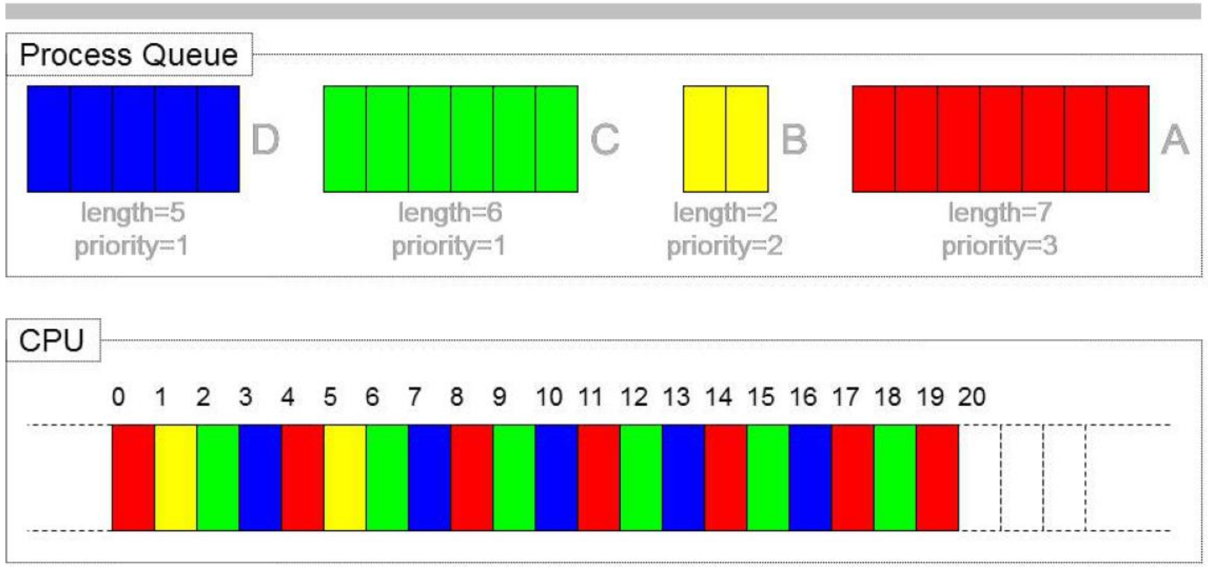
- Average response time = $0 + 2 + 7 + 13 = \frac{22}{4} = 5.5$
- Average turn around time = $2 + 7 + 13 + 20 = \frac{42}{4} = 10.5$

- Disadvantages:
  - **Starvation** might occur: when longer processes may never get CPU time if there are constantly shorter processes arriving.
  - **Fairness** and **predictability** are compromised:
    - SJF is not fair in a system where fairness is critical, as **short processes** are always favored over long ones, leading to **unfair treatment** of longer processes.
    - Users with long-running processes may not be able to predict when their processes will get CPU time,
  - **Requirement of Knowing Processing Times:**
    - SJF requires the processing time (or burst time) of each process to be known in advance or accurately estimated. This information is not always readily available or easy to estimate.

Scheduling Algorithms

Round Robin

- **A preemptive version of FCFS** that forces **context switches** at **periodic intervals** or **time slices**
  - Processes run in the order that they were added to the queue
  - Processes are forcefully **interrupted by the timer**
  - If a time slice is only **used partially**, the next process **starts immediately**



- Average response time = $0 + 1 + 2 + 3 = \frac{6}{4} = 1.5$
- Average turn around time = $6 + 17 + 19 + 20 = \frac{62}{4} = 15.5$

Scheduling Algorithms
Round Robin

- Choosing the Right Time Slice(Quantum) : is a critical parameter in the Round Robin algorithm, as it affects both system responsiveness and throughput.
  - Small Time Slice:
    - Improves response time: A small time slice (e.g., 1 ms) ensures that no process waits long for CPU time. This is beneficial for interactive systems where quick responses are crucial.
    - But increases overhead: The frequent preemptions mean more context switches, which introduces overhead and reduces CPU efficiency.
  - Large Time Slice:
    - Improves throughput: A large time slice (e.g., 1000 ms) reduces the number of context switches, allowing processes to use the CPU for longer periods.
    - But increases response time: A larger time slice increases the average time a process has to wait before it gets CPU time, especially in systems with many processes.

Scheduling Algorithms
Round Robin (Cont.,)

- Advantages
  - Improved Response Time: RR ensures better responsiveness, especially in interactive systems, because processes are not starved, and each one gets CPU time in a predictable manner.
  - Effective for General-Purpose Time-Sharing Systems: well-suited for multi-user, time-sharing environments, as it ensures fairness.
    - Each process gets an equal opportunity to run, making it ideal for systems with multiple interactive users or tasks.
  - Fairness: Every process gets an equal chance at the CPU, preventing any process from monopolizing it.
    - This fairness ensures that lower-priority processes do not starve, unlike priority-based scheduling.

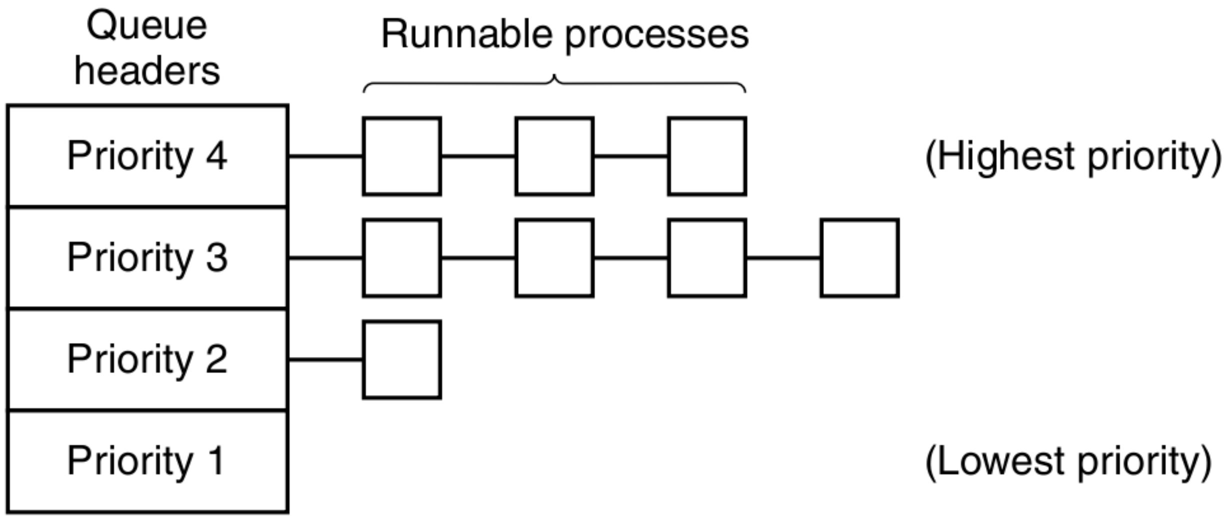
Scheduling Algorithms

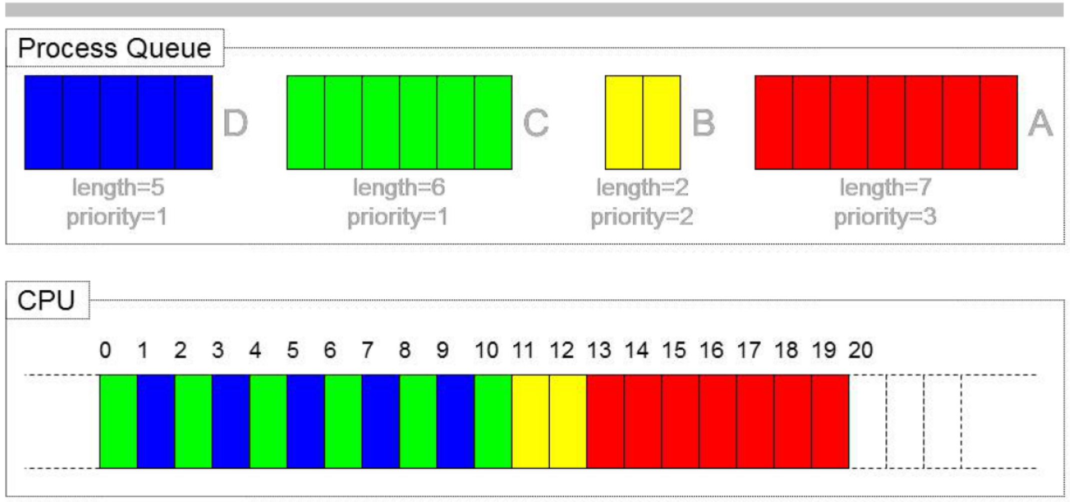
Round Robin (Cont.,)

- Disadvantage
  - **Frequent context switches** occur, especially when the time slice is short.
    - Each context switch incurs overhead (saving and restoring the state of processes), leading to reduced CPU efficiency.
  - **Favor CPU-bound processes** over I/O-bound processes.
    - Since CPU-bound processes use up their entire time slice, they are continuously moved to the end of the queue and come back for more execution. In contrast, I/O-bound processes typically require shorter bursts of CPU time and spend more time waiting for I/O. As a result, they may not get as much CPU time relative to CPU-bound processes.
  - **Can Reduce to FCFS:** If the time slice is too large, Round Robin behaves similarly to First-Come, First-Served (FCFS).
    - When the time slice is large, processes may finish within their time slice, resulting in no preemption, and the scheduler effectively becomes non-preemptive, resembling FCFS scheduling.

Scheduling Algorithms
Priority Queues

- A **preemptive algorithm** that schedules processes by priority (high to low)
  - The process priority is saved in the **process control block**





- Average response time = $0 + 1 + 11 + 13 = \frac{25}{4} = 6.25$
- Average turn around time = $10 + 11 + 13 + 20 = \frac{54}{4} = 13.5$

Scheduling Algorithms

Priority Queues

- Advantages:
  - Improved Responsiveness for Important Tasks:
    - ensures that important or time-sensitive processes (those with high priority) are executed as soon as possible, improving response time for critical tasks.
  - Flexibility:
    - Different processes can be treated differently based on their priority. For instance, real-time processes (e.g., handling urgent I/O) can be given higher priority over background tasks.
  - Suited for Real-Time Systems:
    - Real-time systems that need to guarantee the execution of high-priority tasks (e.g., embedded systems or multimedia applications) can benefit from priority scheduling as it allows these tasks to run without delay.

Scheduling Algorithms

Priority Queues

- Disadvantage
  - Potential for Starvation:
    - Low-priority processes may suffer from starvation, where they wait indefinitely if higher-priority processes continuously arrive.
    - This is particularly problematic in preemptive priority scheduling systems.
  - Response Time Increases for Lower-Priority Processes:
    - While high-priority tasks may benefit from low response times, lower-priority processes may have significantly longer response times.
    - This happens because every time a higher-priority process arrives, the lower-priority process is either preempted (in preemptive scheduling) or has to wait until all higher-priority processes finish.
  - Increased Overhead in Preemptive Priority Scheduling:
    - Frequent preemption of lower-priority processes by higher-priority ones can result in high context-switching overhead, leading to inefficiency, especially if the higher-priority processes have short CPU bursts but arrive frequently.

Summary

Take Home Message

- The OS is responsible for **process scheduling**
- Different types of schedulers exist (e.g. pre-emptive, etc.)
- Different **evaluation criteria** exist for process scheduling
- Different **algorithms** should be considered