1 Process scheduling

1.1 Context

The OS is responsible for **managing** and **scheduling processes**. It has to decide

- When to **admin** processes to the system (new -¿ ready)
- Which process to **run** next (ready -¿ run)
- When and which processes to **interrupt** (running -¿ ready)

It relies on the **scheduler** (dispatcher) to decide which process to run next, which uses **scheduling algorithm** to do so
The type of algorithm used by the scheduler is influenced by the the **type of the operating system** (e.g. real time vs. batch)

1.2 Classification by Time Horizon

Long term: applies to **new processes** and controls the degree of multiprogramming by deciding which processes to admit to the system when

- A good **mix** of **CPU** and **I/O bound processes** is favourable to keep all resources as busy as possible
- **Usually absent** in popular/modern OS

Medium term controls swapping and the degree of multi-programming
Short term: decide which process to run next

- Manages the **ready queue**
- Invoked very **frequently**, hence must be **fast**
- Usually called in response to **clock interrupts, I/O interrupts, or blocking system calls**

1.3 Classification by Approach

Non-preemptive: processes are only interrupted voluntarily (e.g., I/O operations or "nice" system call - yield()).
Windows 3.1 and DOS were non-preemptive
Preemptive processes can be **interrupted forcefully or voluntarily**

- This requires context switches which generate **overhead**, too many of them should be avoided
- Prevents processes from **monopolising the CPU**
- **Most popular** modern operating systems are preemptive

2 Performance Assesment

2.1 Criteria

User oriented criteria:

- **Response time** minimise the time between creating the job and its first execution
- **Turnaround time** minimise the time between creating the job and finishing it
- **Predictability time** minimise the variance in processing times

System oriented criteria:

- **Throughput:** maximise the number of jobs processed per hour
- **Fairness:** Are the processing power/waiting time equally distributed. Are some processes kept waiting exececessively long (**starvation**)

Evaluation criteria can **conflicting** i.e, **improving the response time** may require **more context switches**, and hence **worsen the throughout** and **increase the turn around time**

3 Scheduling algorithms

3.1 Overview

Algorithms considered

- First come first served **FCFS**/ First in first out **FIFO**
- **Shortest job first**
- **Round robin**
- **Priority queues**

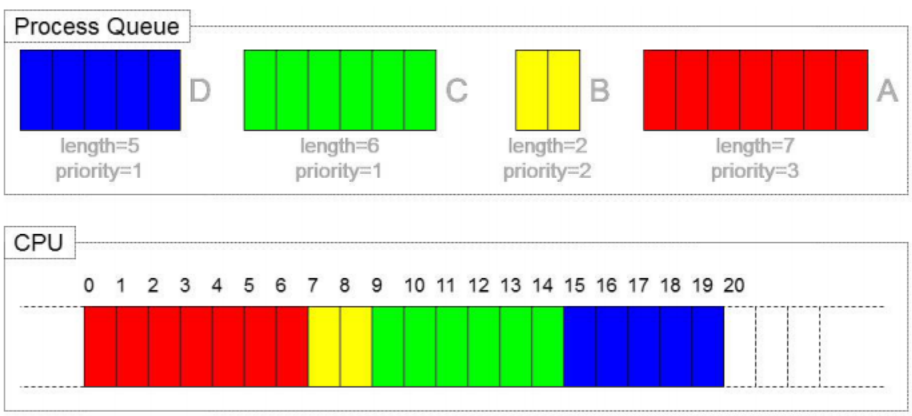
Performance measure used:

- **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

3.2 First come first served FCFS

Concept: a **non-preemtive algorithm** that operates as a **strict queueing mechanism** and schedules the processes in the same order that they were added to the queue
Advantages: **positional fairness** and easy to implement
Disadvantages:

- **Favours long processes** over short ones
- Could **compromise resource utilisation**, i.e., CPU vs I/O devices

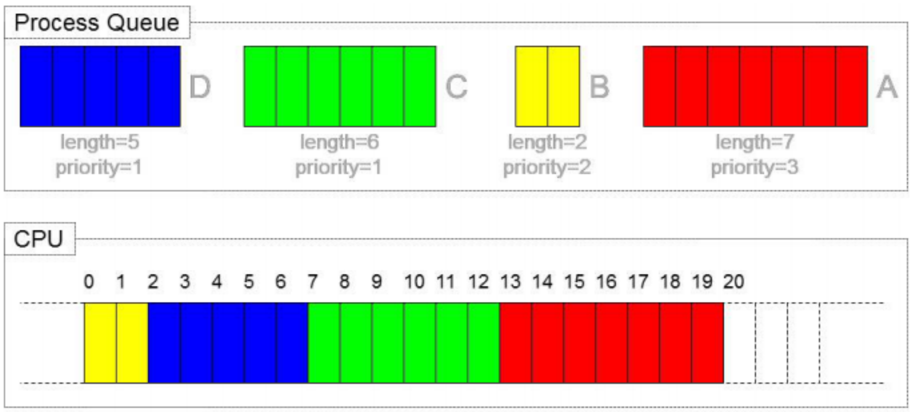


- Average response time = $0 + 7 + 9 + 15 = \frac{31}{4} = 7.75$
- Average turn around time = $7 + 9 + 15 + 20 = \frac{51}{4} = 12.75$

3.3 Shortest job first

Concept: a **non-preemtive 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**
Disadvantages:

- **Starvation** may occur
- **Fairness** and **predictability** are compromised
- **Processing times have to be known** beforehand



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

3.4 Round robin

Concept: a **preemptive version of FCFS** that forces **context switches** at **periodic intervals or time slices**

- Processes **run in order that they were added** to the queue
- Processes are forcefully **interrupted by the timer**

Should be used when it is desirable to allow long running processes to execute while not interfering with shorter ones
Advantages:

- Improved **response time**
- Effective for general purpose **time sharing systems**
- Starvation free

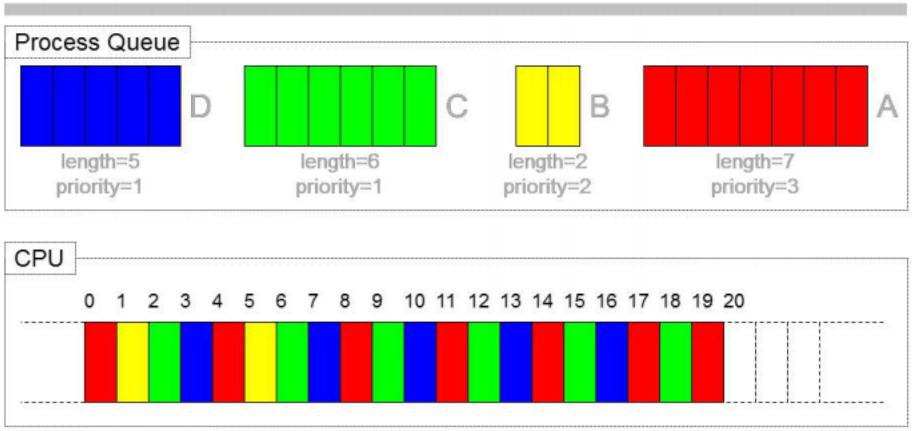
Disadvantages:

- Increased **context switching** and thus overhaul
- **Favours CPU bound processes**(which usually run long) over I/O (which do not run long). Can be prevented by working with multiple queues
- Can **reduce to FCFS**

The lenght of time **time slice** must be carefully considered
For instance, assuming a **multi-programming system** with **preemptive scheduling** and a **context switch time of 1ms**

- A **good (low) response time** is achieved with a **small time slice**(1ms) =¿ low throughput
- A **hight throughput** is achieved with a **large time slice** (1000ms) =¿ hight response time

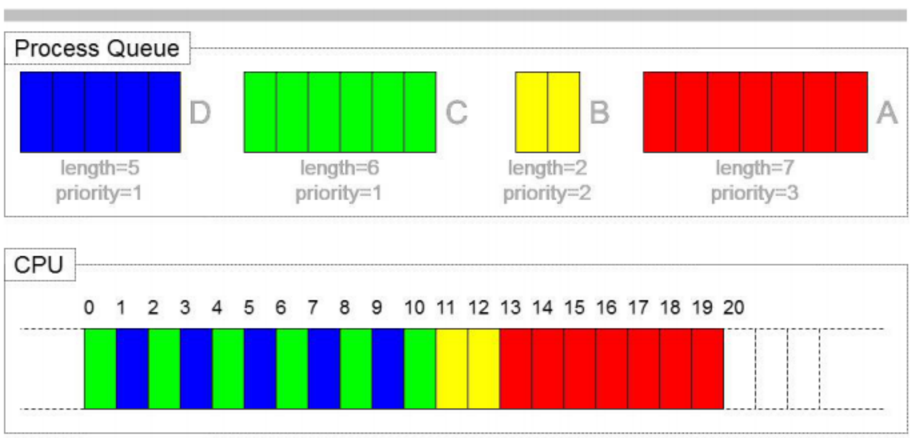
If a time slice is only **used partially** the next process **starts immediately**



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

3.5 Priority queues

Concept: A **preemptive algorithm** that schedules processes by priority (high =¿ low). The process priority is saved in the **process control block**
Advantages: can **prioritise I/O bound jobs**
Disadvantages: low priority processes may suffer from **starvation** with static priorities.



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

4 Summary

- The OS is responsible for **process scheduling**
- Different types of schedulers exist
- Different **evaluation criteria** exists for process scheduling
- Different **algorithms** should be considered