
University of
Nottingham
UK | CHINA | MALAYSIA
Keeping each other safe on campus
Wear a face
covering
when indoors
nottingham.ac.uk/coronavirus

Admin

Git & Linux Servers

- Git access:
  - Any issues should now be resolved
  - Please make sure that you log on to the Git server
- Linux servers:
  - Tutorials are available in the Labs section on how to access the servers
  - The file systems are shared
- EoI dates:
  - Submission - 21st of October 2021 (CV, written EoI, pitch)
  - Pitch: 26th of October 2021 (schedule TBC)

Operating Systems and Concurrency

Processes 2: Process Scheduling

COMP2007 (G52OSC)

Geert De Maere
(Alexander Turner)
{Geert.DeMaere, Alexander.Turner}@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2021

Recap

Last Lecture

- Processes have **“control structures”** (process control blocks and process tables)
- Processes have **states** and **transition** (e.g. new, ready, running, blocked, terminated)
- Operating systems have **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

Context

- The OS **manages** and **schedules** processes
  - New $\rightarrow$ ready: when to **admit** processes to the system
  - Ready $\rightarrow$ running: decide which process to **run** next
  - Running $\rightarrow$ ready: when to **interrupt** processes
- The **scheduler** (dispatcher) decides which process to run next (using a **scheduling algorithm**)
- The **type of operating system** determines which **algorithms** are appropriate (e.g., real time vs. batch)

Process Schedulers

Classification by Time Horizon

- **Long term**: admits **new processes** and controls the **degree of multiprogramming**
  - A good **mix** of **CPU** and **I/O bound processes**
  - **Usually absent** in popular modern OS
- **Medium term**: controls swapping (and degree of multi-programming)
- **Short term**: which process to run next
  - Manages the **ready queue**
  - Runs **frequently** – must be **fast**
  - Called following **clock interrupts** or **blocking system calls** (e.g. I/O interrupts)

Process Schedulers

Classification by Time Horizon

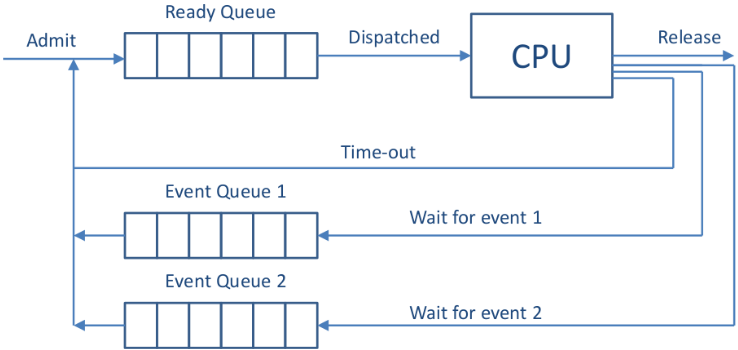


Figure: Queues in OS

Exam 2013-2014: Where do the process schedulers fit in with the state transitions?

Process Schedulers

Classification by Approach

- **Non-preemptive**: processes are “interrupted” **voluntarily**:
  - For instance, I/O operation or “nice” system call – `yield()`
  - Windows 3.1 and DOS were non-preemtive
- **Preemptive**: processes are **interrupted forcefully** or **voluntarily**
  - This requires **context switches** which generate (overhead)
  - Prevents processes from **monopolising the CPU**
  - **Most popular** modern operating systems are preemptive

Performance Assessment

Criteria

- **User oriented criteria**:
  - **Response time**: time between creating the job and its first execution
  - **Turnaround time**: time between creating the job and finishing it
  - **Predictability**: variance in processing times
- **System oriented criteria**:
  - **Throughput**: number of jobs processed per hour
  - **Fairness**:
    - Equally distributed processing / waiting time?
    - Are processes **starved** (wait excessively long)?

Performance Assessment

Conflicts

- Evaluation criteria can be **in conflict**:
  - **Improving the response time** requires **more context switches**
  - More context switches **worsen the throughput** and **increase the turnaround time**

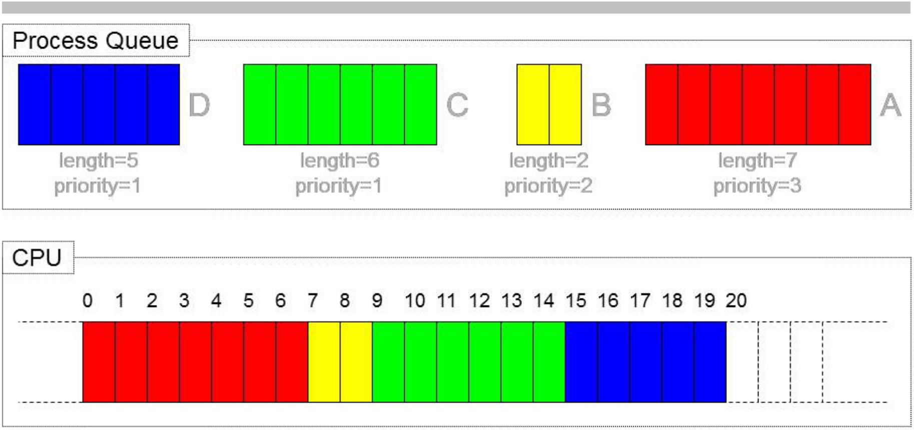
Scheduling Algorithms

Overview[1]

- **Algorithms** considered:
  1. First Come First Served (**FCFS**)/ First In First Out (**FIFO**)
  2. **Shortest job first**
  3. **Round Robin**
  4. **Priority queues**
- 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

[1] Images / animations by Jon Garibaldi

- Concept: a **non-preemtive algorithm** that operates as a **strict queueing mechanism**
- Advantages: **positional fairness** and easy to implement
- Disadvantages:
  - **Favours long processes** over short ones
  - Could **compromise resource utilisation** (CPU vs. I/O devices)

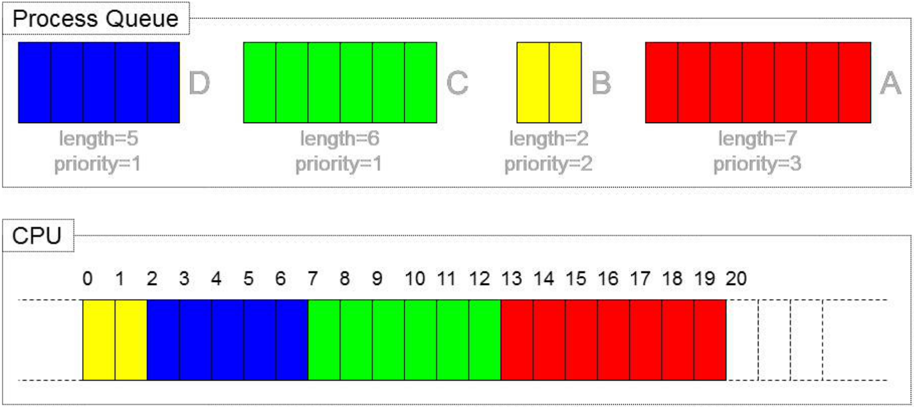


- Average response time = $0 + 7 + 9 + 15 = \frac{31}{4} = 7.75$
- Average turnaround time = $7 + 9 + 15 + 20 = \frac{51}{4} = 12.75$

- Concept: A **non-preemtive algorithm** that starts processes in order of **ascending processing time**
- Advantages: always result in the **optimal turnaround time**
- Disadvantages:
  - **Starvation** might occur
  - **Fairness** and **predictability** are compromised
  - **Processing times have to be known** beforehand

Scheduling Algorithms

Shortest Job First

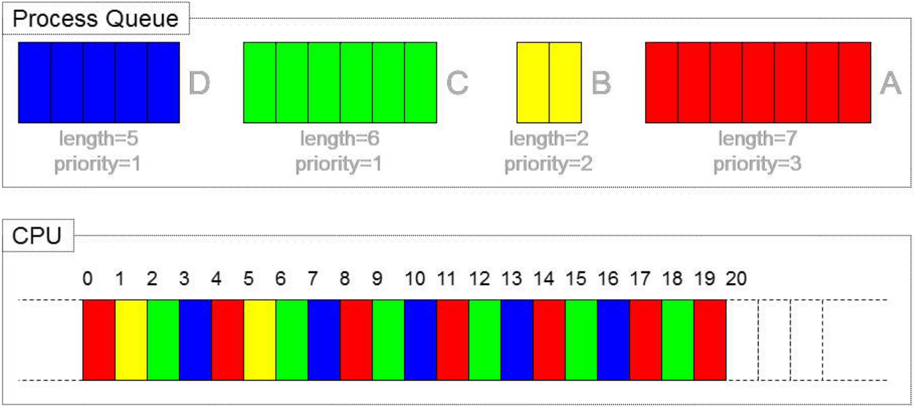


- Average response time = $0+2+7+13=\frac{22}{4}=5.5$
- Average turnaround time = $2+7+13+20=\frac{42}{4}=10.5$

- Concept: a **preemptive version of FCFS**
  - Processes **run in the order that they were added**
  - Forces **context switches** at **periodic intervals** (**time slice** - **interrupts**)
- Advantages:
  - Improved **response time**
  - Effective for general purpose **interactive/time sharing systems**
- Disadvantages:
  - Increased **context switching** and thus overhead
  - **Favours CPU bound processes** over I/O processes
  - Can **reduce to FCFS**

Exam 2013-2014: Round Robin is said to favour CPU bound processes over I/O bound processes. Explain why may this be the case (if this is the case at all)?

- The **length** of the **time slice** must be carefully considered!
- For instance, assuming a **multi-programming system** with **preemptive scheduling** and a **context switch time** of 1ms:
  - E.g., a **good (low) response time** is achieved with a **small time slice** (e.g. 1ms) $\Rightarrow$ low throughput
  - E.g., a **high throughput** is achieved with a **large time slice** (e.g. 1000ms) $\Rightarrow$ high 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 turnaround time = $6 + 17 + 19 + 20 = \frac{62}{4} = 15.5$

Scheduling Algorithms

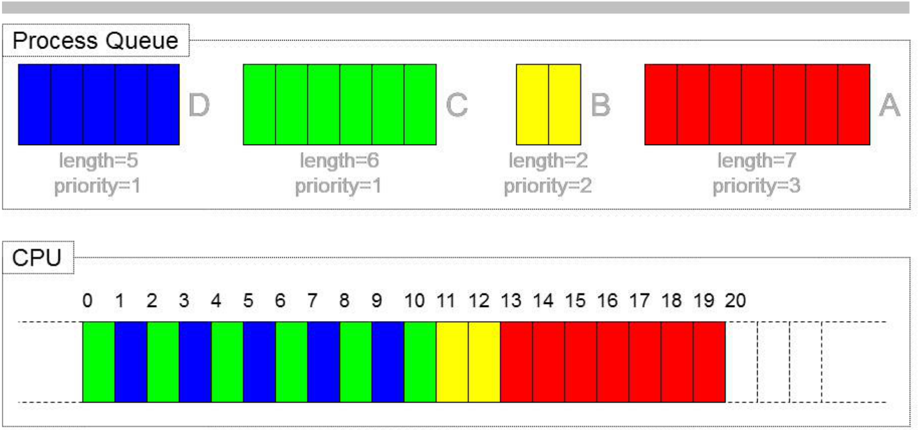
Priority Queues

- Concept: A **preemptive algorithm** that schedules processes by priority (high $\rightarrow$ low)
  - A **round robin** is used **within the same priority levels**
  - 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)

Exam 2013-2014: Out of the following four scheduling algorithms, which one can lead to starvation: FCFS, shortest job first, round robin, highest priority first? Explain your answer.

Scheduling Algorithms

Priority Queues



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

Multi-level Feedback Queues

Moving Beyond Priority Queues

- Different **scheduling algorithms** can be used for the **individual queues** (e.g., round robin, SJF, FCFS)
- **Feedback queues** allow **priorities to change dynamically**, i.e., jobs can **move between queues**:
  - Move to **lower priority queue** if too much CPU time is used (prioritise I/O and interactive processes/threads)
  - Move to **higher priority queue** to prevent **starvation** and avoid **inversion of control**

Exam 2013-2014: Explain how you would prevent starvation in a priority queue algorithm

Inversion of Control

Illustration

```
Process A (low)        Process B (high)       Process C (high)
...
request X
receive X
...                    RUN
...                    request X
...                    blocked                RUN
...                    ...                    ...
```

- Give the **order in which the processes are scheduled** when using **priority queues**, together with the **times at which they will start, end, and are interrupted** (all processes are available at the time of scheduling)
- You can assume a time slice of 15 milliseconds
- Calculate the **average response and turnaround time**

	FCFS Position	CPU burst time	Priority
Process A	1	67	1 (high)
Process B	2	37	1 (high)
Process C	3	14	2 (low)
Process D	4	16	2 (low)

- Solution:
  - Sequence: A(15) $\Rightarrow$ B(15) $\Rightarrow$ A(15) $\Rightarrow$ B(15) $\Rightarrow$ A(15) $\Rightarrow$ B(7) $\Rightarrow$ A(15) $\Rightarrow$ A(7) $\Rightarrow$ C(14) $\Rightarrow$ D(15) $\Rightarrow$ D(1)
  - Average response time = (0 + 15 + 104 + 118) / 4
  - Average turnaround time = (82 + 104 + 118 + 134) / 4
- Note: we ignore **context switch time**

Summary

Take Home Message

- Summary:
  - The OS is responsible for **process scheduling**
  - Different types of schedulers exist (e.g. pre-emptive, short term, etc.)
  - Different **evaluation criteria** exist for process scheduling
  - Different **algorithms** should be considered
- Reading:
  - Tanenbaum: section 2.4
  - Silberschatz: section 5.1 - 5.4