Process Scheduling

OPS Lecture 4, G53OPS/G52OSC

Geert De Maere
(Jason Atkin – OSC)
Geert.DeMaere@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2015

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()`

```
while (TRUE) {                                  /* repeat forever /*/
    type_prompt( );                             /* display prompt on the screen */
    read_command(command, params);              /* read input line from keyboard */

    pid = fork( );                              /* fork off a child process */
    if (pid < 0) {
        printf("Unable to fork0);               /* error condition */
        continue;                               /* repeat the loop */
    }

    if (pid != 0) {
        waitpid (–1, &status, 0);               /* parent waits for child */
    } else {
        execve(command, params, 0);             /* child does the work */
    }
}
```

Figure: Use of the fork() and exec() system calls (Tanenbaum)

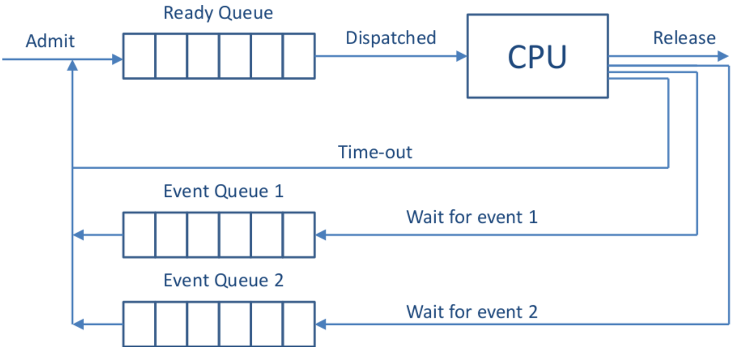
Goals for Today

Overview

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

Process Scheduling
Context

- The OS is responsible for **managing** and **scheduling processes**
  - Decide when to **admit** processes to the system (new $\rightarrow$ ready)
  - Decide which process to **run** next (ready $\rightarrow$ run)
  - Decide which when to **interrupt processes** (running $\rightarrow$ ready)
- It relies the **scheduler** (dispatcher) to decide which process to run next through the use of a **scheduling algorithm**
- The type of algorithm used by the scheduler is influenced by the **type of operating system** (e.g., real time vs. batch)



Ready Queue
Admit
Dispatched
CPU
Release
Time-out
Event Queue 1
Wait for event 1
Event Queue 2
Wait for event 2

Process Schedulers
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
  - **Usually absent** in popular modern OS
  - A good **mix** of **CPU** and **I/O bound processes** is favourable to keep all resources as busy as possible
- **Medium term**: controls swapping and the degree of multi-programming
- **Short term**: decide which process to run next
  - Usually called in response to **clock interrupts**, **I/O interrupts**, or **blocking system calls**
  - Invoked very **frequently**, hence must be **fast**
  - Manages the **ready queue**

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

Process Schedulers

Classification by Approach

- **Non-preemptive**: processes are only interrupted voluntarily (e.g., I/O operation or "nice" system call – `yield()`)
  - Windows 3.1 and DOS were non-preemtive
- **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

Performance Assessment
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**: minimise the variance in processing times
- **System oriented criteria**:
  - **Throughput**: maximise 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

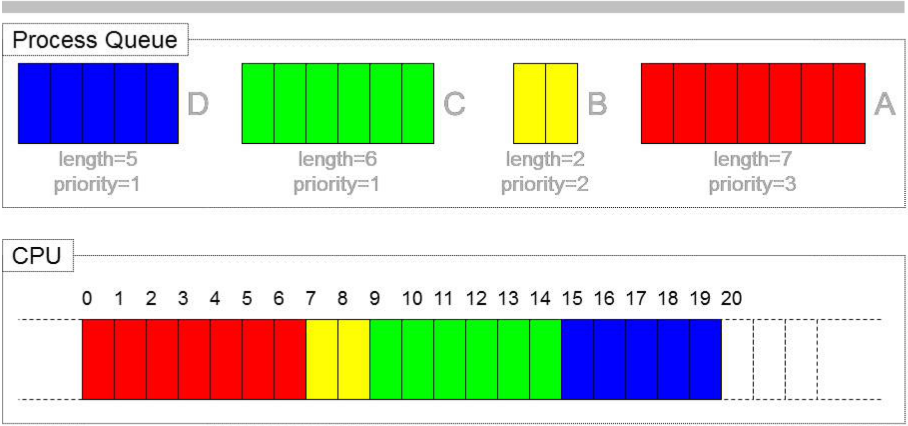
- **Algorithms** considered:
  1. First Come First Served (**FCFS**)/ First In First Out (FIFO)
  2. **Shortest job first**
  3. **Round Robin**
  4. **Priority queues**
- Performance measures used:
  - **Average response time**: the average of the time taken for all the processes to start
  - **Average turnaround time**: the average time taken for all the processes to finish
- Images/animations by Jon Garibaldi!

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

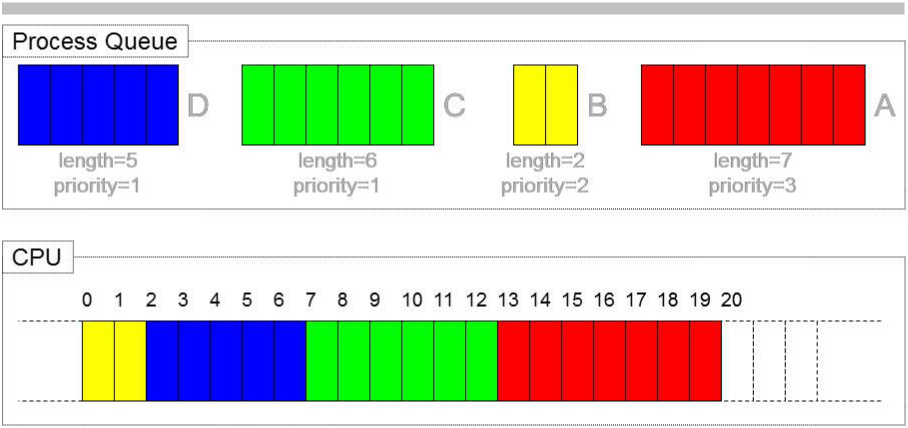
First Come First Served

- 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 (think of the supermarket checkout!)
  - 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 = $7 + 9 + 15 + 20 = \frac{51}{4} = 12.75$

- 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** might occur
  - **Fairness** and **predictability** are compromised
  - **Processing times have to be known** beforehand or estimated by, e.g. using exponential averages

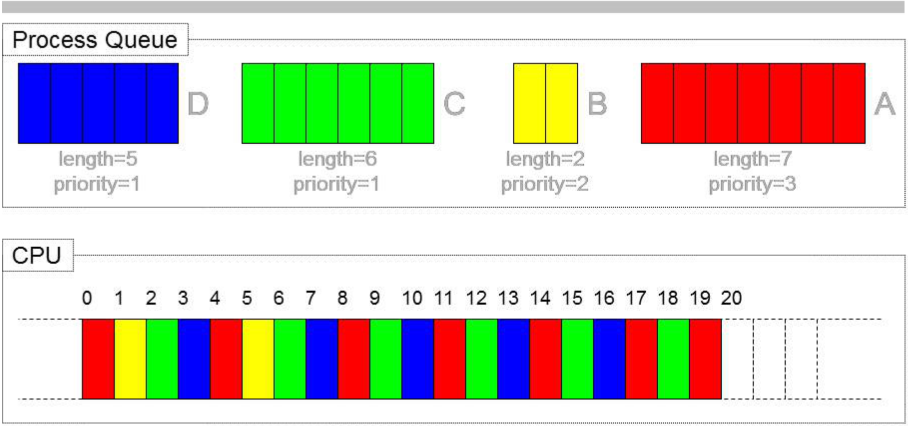


- Average response time = $0+2+7+13=\frac{22}{4}=5.5$
- Average turn around time = $2+7+13+20=\frac{42}{4}=10.5$

- Concept: 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**
- Advantages:
  - Improved **response time**
  - Effective for general purpose **time sharing systems**
- Disadvantages:
  - Increased **context switching** and thus overhead
  - **Favours CPU bound processes** (which usually run long) over I/O processes (which do not run long) - how can this be prevented?
  - 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)?

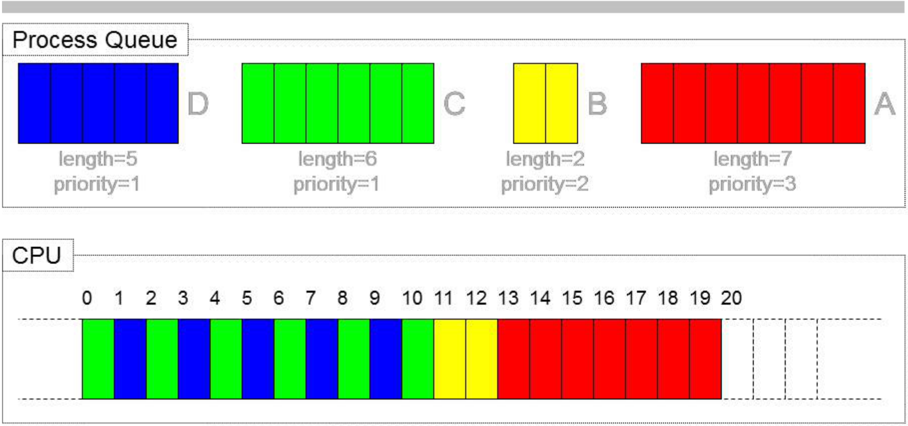
- The **length** of the **time slice** must be carefully considered!
- For instance, assuming a **multi-programming system** with **preemptive scheduling** and a **context switch time** of 1ms:
  - E.g., a **low response time** is achieved with a **small time slice** (e.g. 1ms) $\Rightarrow$ low throughput
  - E.g., a **high throughput** is achieved with a **large time slice** (e.g. 1000ms) $\Rightarrow$ low response time



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

- Concept: A **preemptive algorithm** that schedules processes by priority (high $\rightarrow$ low)
  - The process priority is saved in the **process control block**
  - Priorities can be assigned dynamically
- Advantages: can **prioritise I/O bound jobs**
- Disadvantages: low priority processes may suffer from **starvation** (with static priorities)

Scheduling Algorithms
Priority Queues



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

Algorithm	Response	Turnaround
FCFS	7.75	12.75
SJF	5.5	10.5
RR	1.5	15.5
PQ	6.25	13.5

Table: Algorithm Comparison

- **Shortest job first** always has the **lowest turn around time**
- **Round Robin** always has the **shortest response time**

Exam 2013-2014: Illustrate the use of round robin, shortest job first, and highest priority first scheduling algorithms for the listed processes (+ calculate average turn around/response time)

- 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