Processes Scheduling and Threads

OPS Lecture 5, G53OPS/G52OSC

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

University Of Nottingham
United Kingdom

2015

Recap

Last Lecture

- **Types of schedulers**: preemptive/non-preemptive, long/medium/short term)
- Performance **evaluation criteria**
- Scheduling **algorithms**: FCFS, SJF, Round Robin

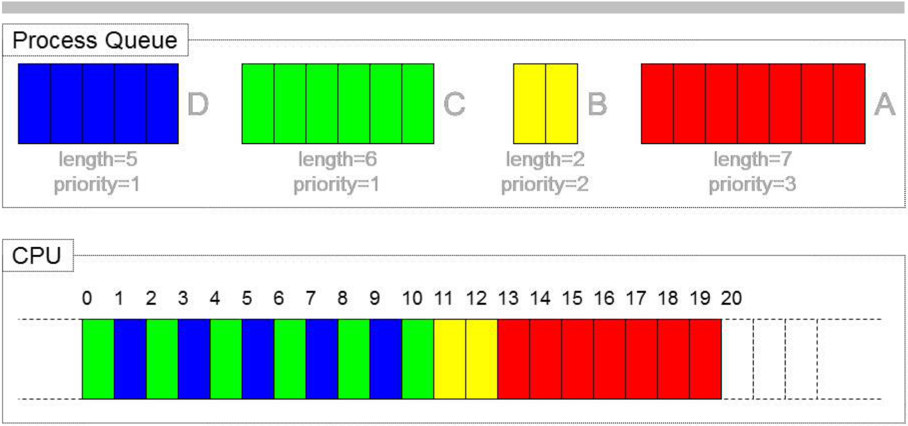
Goals for Today

Overview

- Priority queues and **multi-level feedback queues**
- Scheduling in **Window 7** + Demo
- **Threads** from an OS perspective

- Concept: A **preemptive algorithm** that schedules processes by priority (high $\rightarrow$ 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)

Scheduling Algorithms
Priority Queues



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

- Priority queues are usually implemented by **using multiple queues** (e.g. for foreground or background processes)
- Every queue can have its own **scheduling algorithm** and different algorithms can be used for **individual queues** (e.g., round robin, FCFS)
- **Feedback queues** allow **priorities to change dynamically**, i.e., jobs can **move between queues**, e.g.:
  - Move to **lower priority queue** if too much CPU time is used (prioritise I/O and interactive processes)
  - Move to **higher priority queue** to prevent **starvation** and avoid **inversion of control**

Exam 2013-2014: Explain how you would prevent starvation in a multi-level queue scheduling algorithm

Multi-level Feedback Queues

Moving Beyond Priority Queues

- Defining characteristics of feedback queues:
  - The **number of queues**
  - The **scheduling algorithms** used for the individual queues
  - **Migration policy** between queues
  - Initial **access** to the queues
- Feedback queues are highly **configurable** and offer significant flexibility

- An **interactive system** using a **pre-emptive scheduler** with **(dynamic) priority levels**
  - Two priority classes with 16 different priority levels exist
    - "**Real time**" processes/threads have a **fixed priority level**
    - "**Variable**" processes/threads can have their priorities **boosted temporarily**
- A **round robin algorithm** is used within the queues

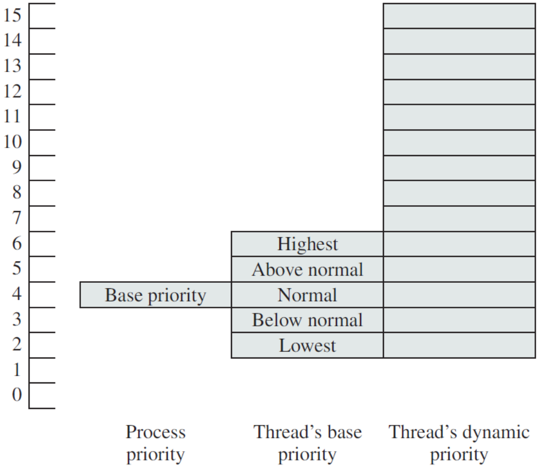


Figure: Priorities in Windows 7 (Stallings, 7^{th} edition)

- Priorities are based on the **process base priority** (0-15) and **thread base priority** ($\pm$2 relative to the process priority)
- A thread's **priority dynamically changes** during execution between its base priority and the maximum priority within its class
  - **Interactive I/O bound processes** (e.g. keyboard) receive a **larger boost**
- Boosting priorities prevents priority inversion: i.e. it frees up **resources held by lower priority processes**

- Code available on Moodle
- Examples:
  1. Six threads, equal priority, 1 core/processor
  2. Three threads, different priorities, 1 core/processor
  3. Three threads, different priorities, 1 core/processor, admin privileges
  4. Two threads, both high priority, 1 core/processor
  5. Two threads, both high priority, 2 cores/processors

- A process consists of two **fundamental units**
  - **Resources**: all related resources are grouped together
    - A logical address space containing the process image (program, data, heap, stack)
    - Files, I/O devices, I/O channels, . . .
  - **Execution trace**, i.e., an entity that gets executed
- A process can **share its resources** between **multiple execution traces** that are **interleaved**, i.e., multiple threads running in the same resource environment

- Every thread has its own **execution context** (e.g. program counter, stack, registers)
- All threads have **access** to the process' **shared resources**
  - E.g. files, one thread opens a file, all threads of the same process can access the file
  - Global variables, memory, etc. ($\Rightarrow$ synchronisation!)
- Some CPUs (hyperthreaded ones) have direct **hardware support** for **multi-threading**
  - They can offer up to 8 hardware threads per core

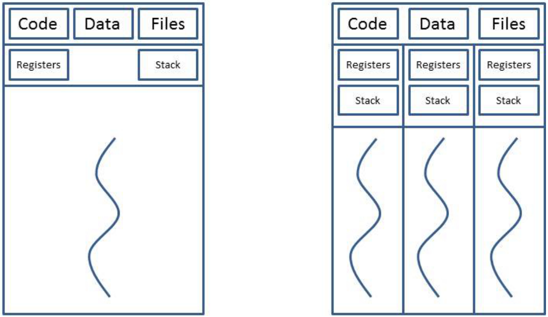


Figure: Single threaded process (left), multi-threaded process (right)

- Similar to processes, threads have:
  - **States** and **transitions** (new, running, blocked, ready, terminated)
  - A **thread control block**
- Threads incur **less overhead** to create/terminate/switch (address space remains the same for threads of the same process)

Processes	Threads
Address space	Program Counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	Local vars
Signals and signal handlers	
Accounting information	

Table: Shared resources left, private resources right

- **Inter-thread communication** is faster than **interprocess** communication (shared memory - processes often have to rely on messaging)
- **No protection boundaries** are required in the address space (threads are cooperating, belong to the same user, and have a common goal)
- **Synchronisation** has to be considered carefully!

- Multiple **related activities** apply to the **same resources**, these resources should be accessible/shared
- Processes will often contain multiple **blocking tasks**
  - I/O operations (thread blocks, **interrupt** marks completion)
  - Memory access: pages faults are result in blocking
- Such activities should be carried out in **parallel/concurrently**
- **Application examples**: webservers, make program, spreadsheets, word processors, processing large data volumes

- **User** threads
- **Kernel** threads
- **Hybrid** implementations

- Multi-level feedback queues and Windows Scheduling
- Threads vs. processes