Real-Time Systems & Fault Tolerance

Flávia Maristela (flaviamsn@ifba.edu.br)

Instituto Federal da Bahia
Especialização em Computação Distribuída e Ubíqua (ECDU)

Salvador, Setembro de 2014

Schedule

Definition

programs that interface the machine with the applications programs

William Stallings

Definition

layer of software whose job is to manage computer devices and provide user programs with a simpler interface to the hardware

Andrew Tanenbaum

Operating Systems Main Functionalities

- Manage the pieces of a complex system:
  1. Software Applications
  2. File
  3. Disk
  4. Input/Output
  5. Memory
  6. Processor
  7. ... and too many other devices and components

Operating Systems Main Functionalities

In other words:

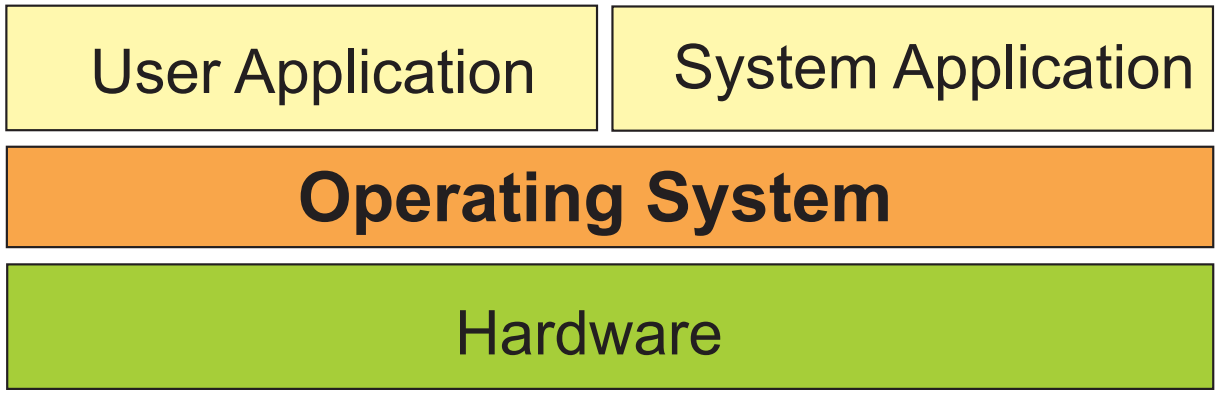
- Manage different resources (ex.: CPU, Memory, Disk)
- Improve computer performance (ex.: response time, throughput)
- Provide an architecture which makes easier programmers life! (abstract hardware layer)

Briefly, an operating system is an interface between computer applications and hardware

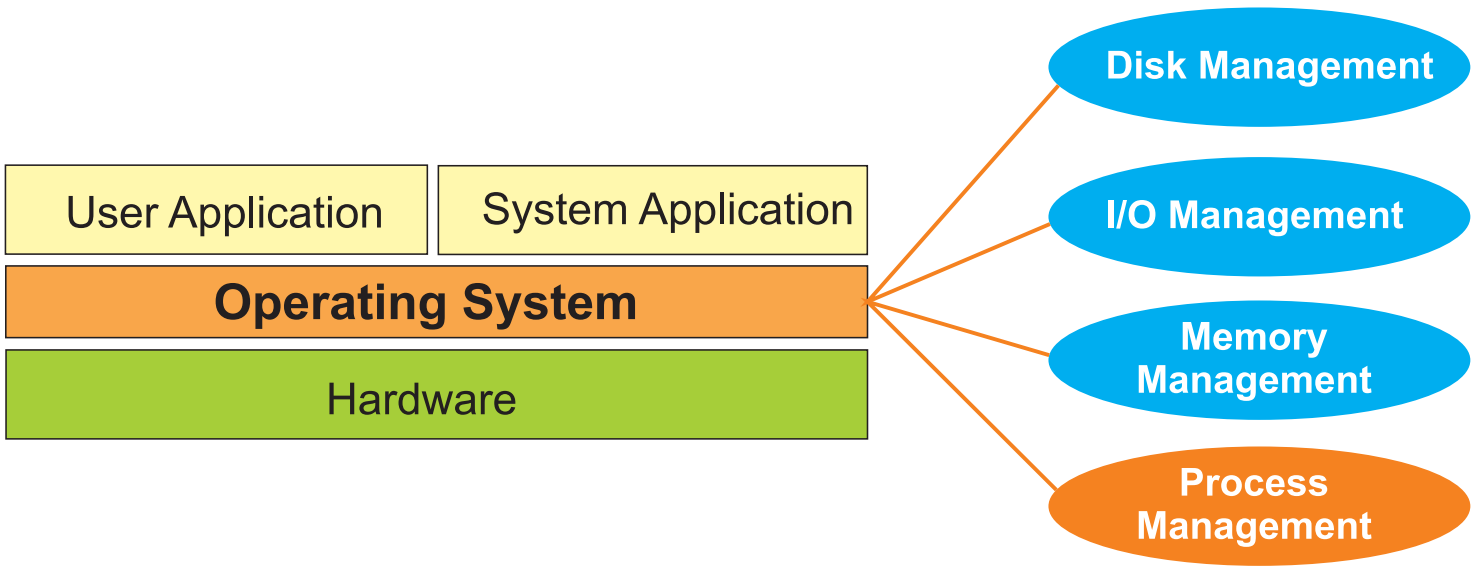
Operating Systems Goals

- Convenience
- Transparency
- Ability to Evolve
  - hardware and/or software improvements
  - new services
  - bug fixes
- Efficiency
- Response Time
- Throughput

Operating Systems Architecture



Our Focus


User Application
System Application
Operating System
Hardware
Disk Management
I/O Management
Memory Management
Process Management

- A program is a collection of running processes
- Processes are **usually** modeled as independent units
- Although independent, they need to share the same computer resources

What is a process?

Definition

A program in execution

Rômulo Oliveira, Simão Toscani, Alexandre Carissimi

Definition

An abstraction of a running program

Andrew Tanenbaum

What is a process?

Definition

An instance of a running program

Abraham Silberschatz

Definition

The entity that can be assigned to and executed on a processor

William Stallings

Understanding Processes

- When a program is executed by a single process it is called **sequential program**
- Most programs are sequential
- On the other hand, when a program is executed by several processes that cooperate, they are called **concurrent program** (sometimes parallel)
- Processes in concurrent program can **cooperate** and/or **compete**
  - Processes compete (as for competition) for hardware and/or operating system resources (processor, memory, data structure , etc.)
  - The term "*concurrent*", means that they happen at the same time

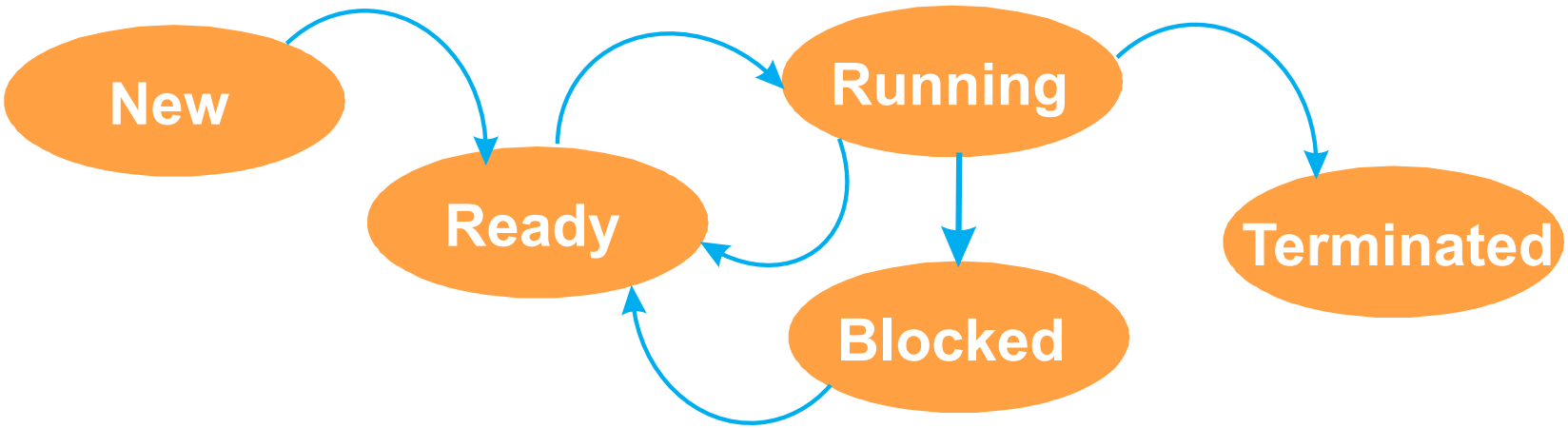
Understanding Processes

- Physical concurrency (real parallelism)
  - requires multiple CPUs
- logical concurrency (pseudo-parallelism, time-shared CPU)
  - multiple tasks share a common single resource
- Both requires:
  - shared memory
  - messages passing
  - synchronization

Process Components

- an executable program
- the associated data needed by the program
  - variables
  - work space
  - buffers
- the execution context
  - states
  - transitions

Process States and Transitions



Definition

Entities that are scheduled for execution

Andrew Tanenbaum

Definition

Execution flow within a process

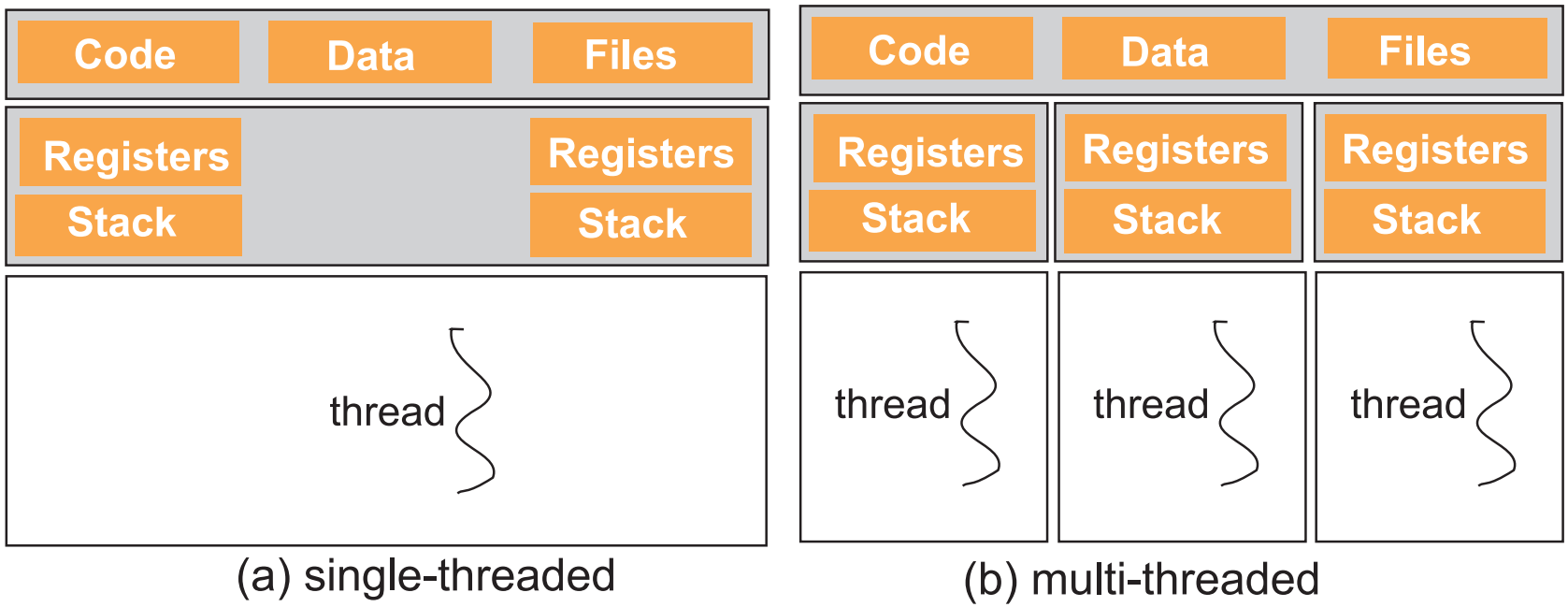
Rômulo Oliveira, Simão Toscani, Alexandre Carissimi

Definition

Basic unit of CPU utilization

Abraham Silberschatz, Peter Galvin and Greg Gagne

Threads Architecture



Some Misconceptions

- Program: a set of instructions, which implements an algorithm in a specific programming language
- Process: a program in execution
- Threads: entities that are scheduled for execution
- Job: an instance of a task

Some Misconceptions

In Real-Time Systems...

we use *tasks* and in this context, tasks = threads

The big deal on Process Management

Motivation context

- many different programs
- many different users
- limited resources
- users in a hurry!!!

The solution

RESOURCE SHARING

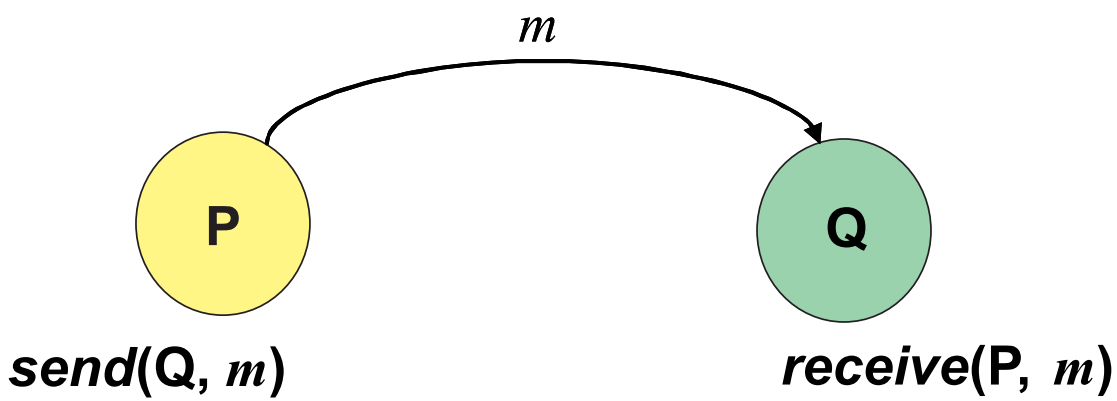
Process Management Activities

- Process creation and deletion
- Process suspension and resumption
- Process synchronization
- Provision of Mechanisms for:
  - Process communication
  - Deadlock handling

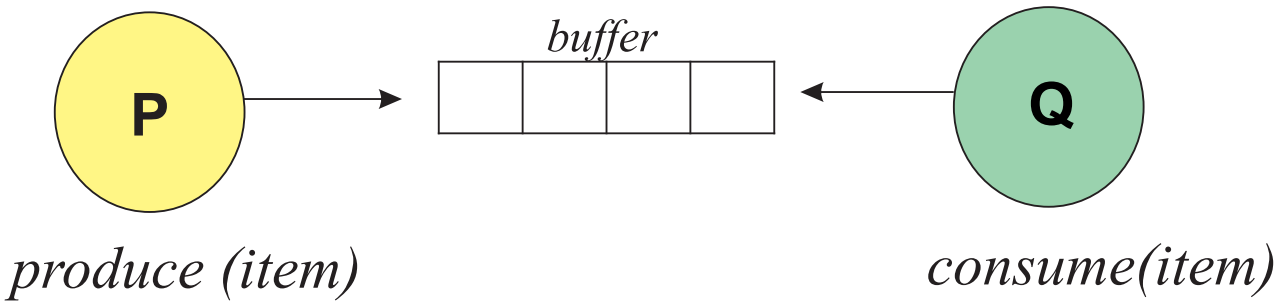
Briefly, we may say that processes **compete** and/or **cooperate**. This is caused by concurrency!

Cooperating Processes

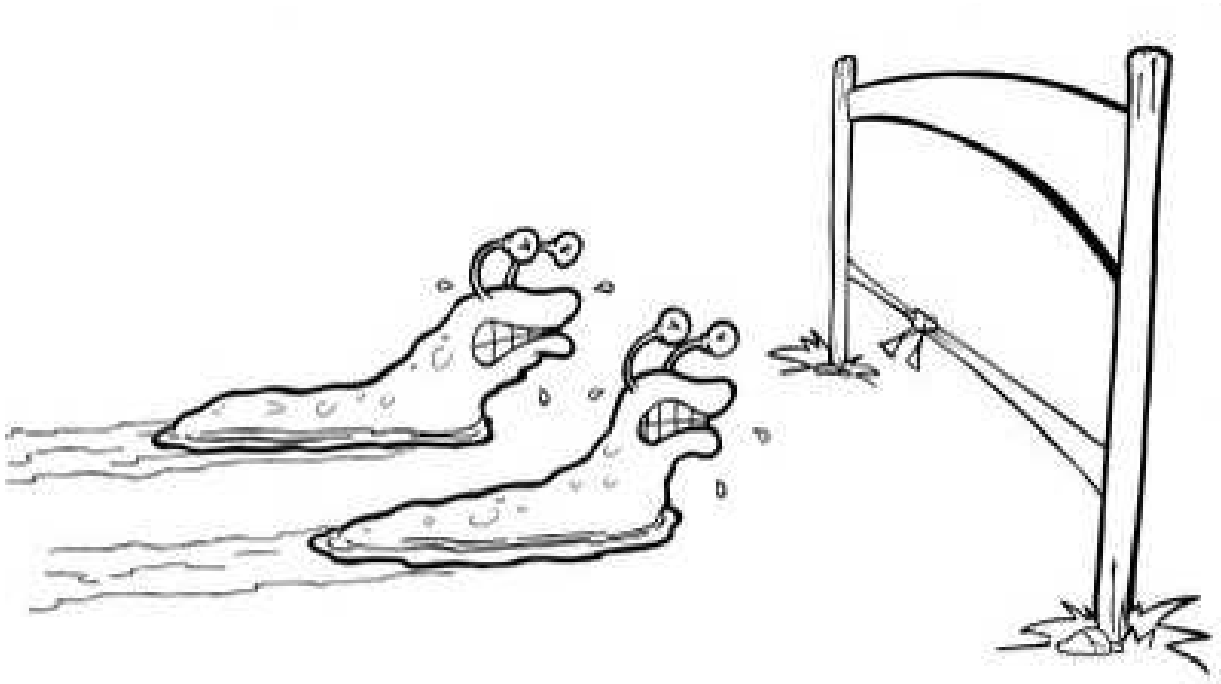
- They can comunicate by message passing or sharing the same buffer



Shared Buffer



Race Condition



Race Condition - Process Competition

$x = 0;$
$begin;$
$p_1 : \ldots;$
$x = x + 1;$
$\ldots;$
$p_2 : \ldots;$
$x = x + 1;$
$\ldots;$

- x is the shared variable
- Final result must be $x = 2;$
- Each process may have a different view of x

Race Condition

Definition

Two different processes can try to read or write data in a shared memory, and the final result depends on who is executing in that moment.

Race Condition - Process Competition

$x = 0;$
$begin;$
$p_1 : \ldots;$
$x = x + 1;$
$\ldots;$
$p_2 : \ldots;$
$x = x * 2;$
$\ldots;$

The Critical Section Problem

Definition

- *n processes all competing to use some shared data;*
- *Each process has a code segment, called* ***critical section*** *in which the shared data is accessed.*
- *The problem is to* ***ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section*** *(Mutual Exclusion).*

Interleaved execution - Parallel Processing

$$
\begin{array}{lll}
p_1: & R_1 = x; & p_2: \quad \ldots \\
& R_1 = R_1 * 2; & \qquad R_2 = x; \\
& x = R_1; & \qquad R_2 = R_2 + 1; \\
& \ldots & \qquad x = R_2;
\end{array}
$$

- x has only been incremented once and its first update ($x = R_1$) is lost.

Critical Section and Mutual Exclusion

- Assume:
  - each reading and writing of individual variables are atomic (indivisible)
  - no priorities associated with critical sections
  - relative speeds of processes are unknown
  - process may halt only outside of its critical section
- Guarantee **mutual exclusion**
  - at any time, only one process is executing within its **critical section**

Prevent **mutual blocking**

- Process outside of its *critial section* must not prevent other processes from entering its *critial section*.
- Process must not be able to repeatedly reenter its *critial section* and starve other processes (**fairness**)
- Processes must not block each other forever (**deadlock**)
- Processes must not repeatedly yield to each other

- Dining Philosophers
- Sleepy Barber
- Producer *vs.* Consumer

Possible Solutions

- Communication Primitives
- Semaphores

- Scheduling involves **two** main activities:
  - Move process to a ready "queue"after creation
  - Select a process to run from the "ready queue"
- The component responsible for scheduling system process is the **scheduler**

Scheduler Invocation Means

- The scheduler is invoked periodically according to processes attributes
  - arrival time
  - execution cost
  - deadline
  - priority

Process Attributes

- Arrival Time (Release Time*): the time at which processes arrive at CPU
- Execution Cost: the time each process will need to execute its computation
- Deadline*: maximum execution term
- Priority: function which maps execution priority order/level
- Period*: activation regularity

Scheduling Activity

- **Scheduler** is the operating system component which is responsible for scheduling activity
- **Scheduling** is the allocation (or reallocation) of tasks to CPUs according to scheduling algorithms
- **Scheduling algorithm** determines in which order tasks must be executed, which is defined according to a **priority function**

Decision Mode

- The scheduler is invoked according to **Decision mode** rules, which are implemented by scheduling algorithms

Non preemptive Decision Mode

1. Each process runs as long as possible
2. The scheduler is called when process terminates or blocks
3. Usually is not useful in time-shared or real-time systems

Decision Mode

Preemptive Decision Mode

1. Each process can stop running and CPU can be re-assigned to other one
2. The scheduler is called when process state changes or periodically (quantum-oriented)

Possible Priority Functions

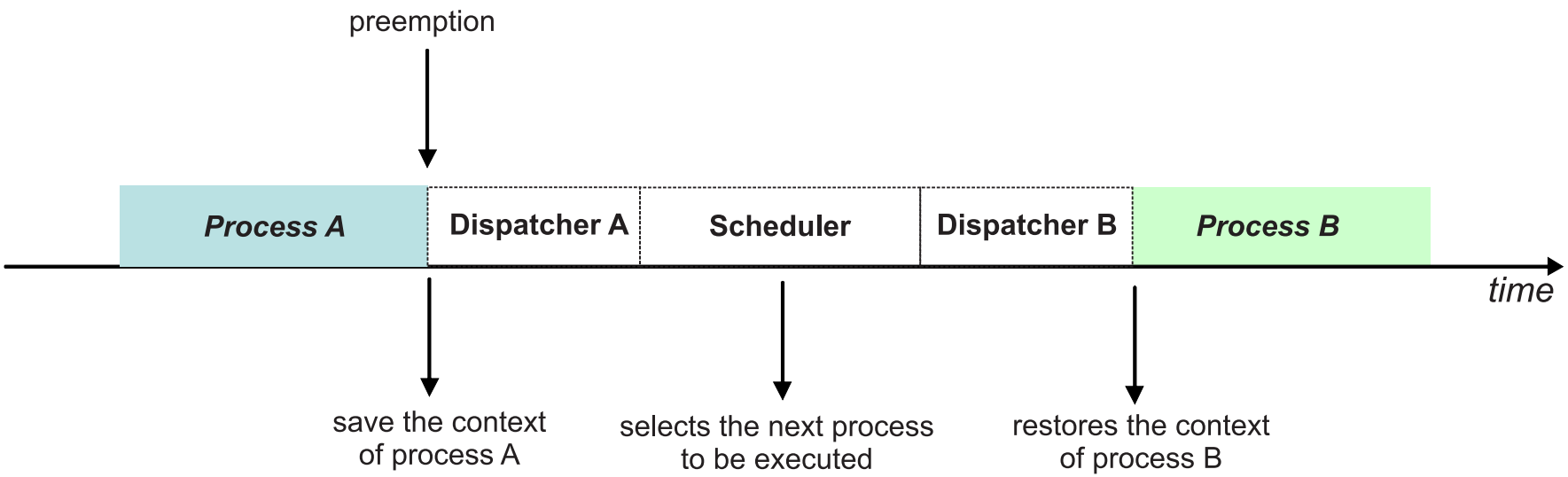
- execution cost
- priority
- arrival time
- deadline
- period
- memory requirements (batch systems)

Scheduling Algorithm Performance

- Each process has informations that allow to precisely define its actual state, also called **context**
- Context Switch

  When a process P_1 is resumed from CPU to allow P_2 execution, their context must be saved and loaded respectively.

Dispatcher Functioning



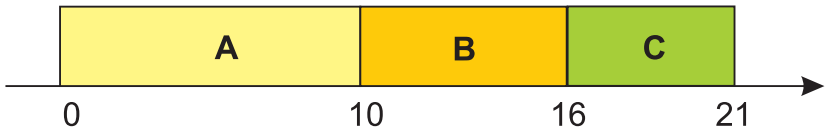
Scheduling Policies

- General Purpose Systems
  - First In First Out (FIFO)
  - Shortest Job First (SJF)
  - Priority
  - Round Robin (RR)
  - Multiple Queue

First In First Out - FIFO

- Dispatch processes according to its arrival time
- Nonpreemptive Decision Mode

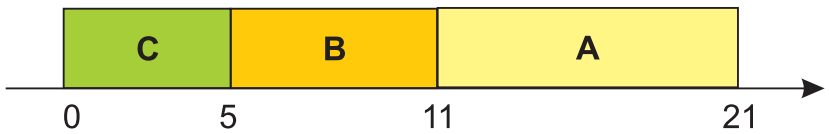
Process	Arrival time	Execution Cost
A	0	10
B	2	6
C	3	5



Shortest Job First - SJF

- Dispatch processes according to its execution time
- Lower execution time processes are dispatched first
- Nonpreemptive Decision Mode

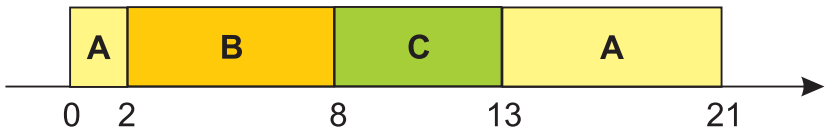
Process	Arrival time	Execution Cost
A	0	10
B	0	6
C	0	5



Priority

- Dispatch processes according to its priority
- Preemptive Decision Mode

Process	Arrival time	Execution Cost	Priority
A	0	10	3
B	2	6	1
C	3	5	2



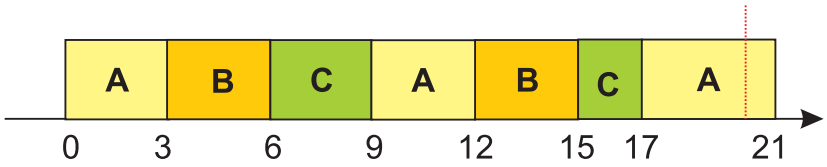
Round Robin - RR

- Performs a cycle within active processes
- Assumes a fixed size quantum **q**
- All processes have same priority
- Processes are preempted after continuously running for **q** time units
- Preemptive Decision Mode

Round Robin - RR

Process	Arrival time	Execution Cost
A	0	10
B	2	6
C	3	5

quantum = 3



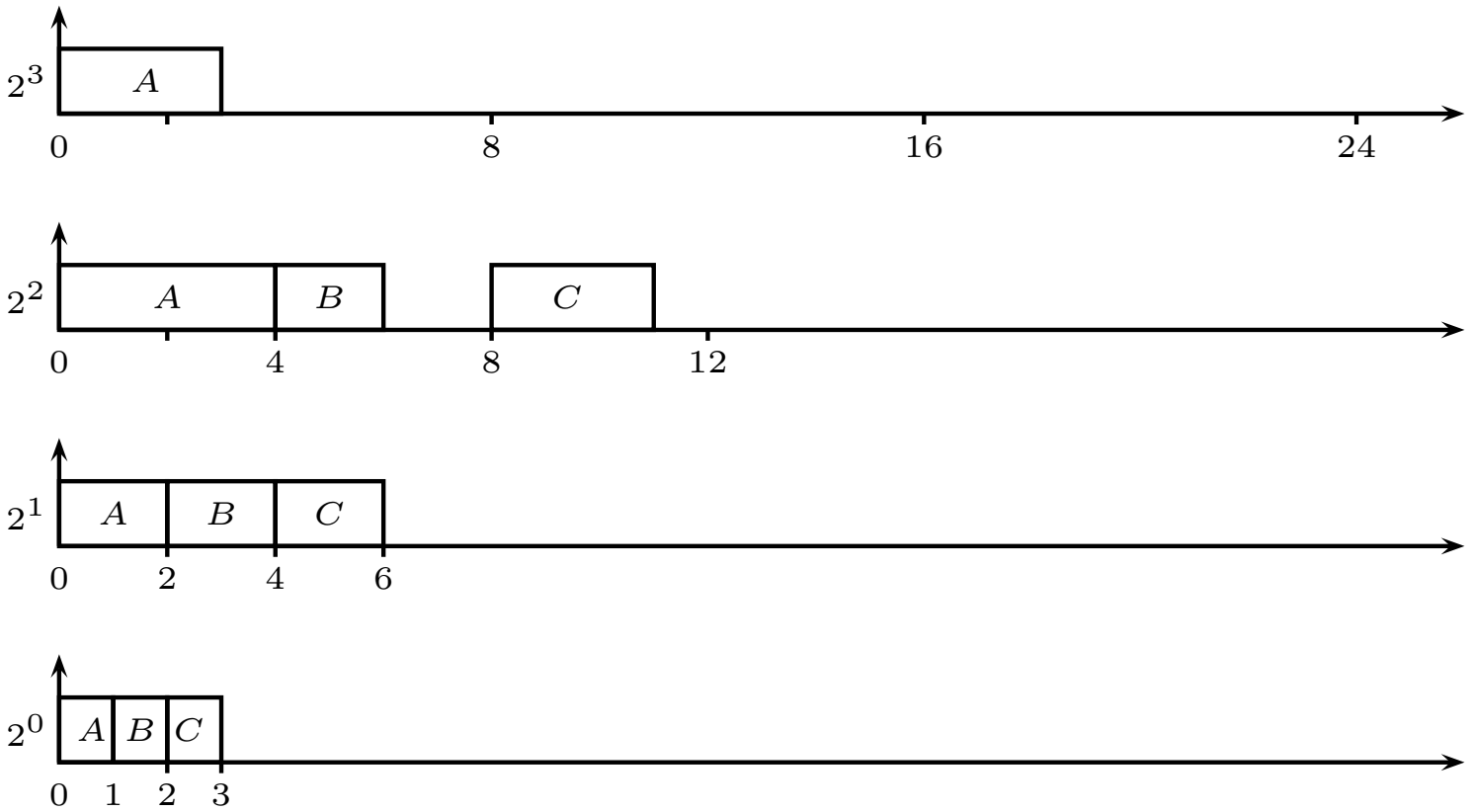
Multiple Queues

- Each Queue has a specific quantum (function $f(x) = 2^x$)
- Minimize context switch for some processes
- Cyclic regarding execution queues

Multiple Queues

Process	Arrival Time	Execution Cost
A	0	10
B	0	6
C	0	5

Tabela : Illustrative Example


2^3
A
0
8
16
24
2^2
A
B
C
0
4
8
12
2^1
A
B
C
0
2
4
6
2^0
A
B
C
0
1
2
3

Scheduling Algorithms Comparison (General Purpose Systems)

- Comparing scheduling policies is a difficult task ... sometimes, impossible!
- To have an idea of different approaches, we can measure:
  - Context switches
  - Preemptions
  - Average Response Time
  - *Tasks Response Time

QUESTIONS?