Scheduling

A.Y. 2017-18

ACSO Tutoring

MSc Eng. Michele Zanella

Process Scheduling

Goals:

- **Multiprogramming**: having some process running at all times, to maximize CPU utilization
- **Time sharing**: switching the CPU among processes so frequently that users can interact with each program while it is running

Scheduler: it is in charge of selecting an available process for program execution on the CPU

CPU-I/O Burst Cycle

- Process execution consists of a **cycle** of CPU execution and I/O wait.
- Process execution begins with a **CPU burst** and is followed by an **I/O burst**.

Preemptive scheduling

- Scheduling decision when:
  - A process swtiches from RUNNING to WAIT state
  - A process terminates
  - A process swtiches from RUNNING to READY
  - A process swtiches from WAITING to READY
- **Non-Preemptive**: once the CPU has been allocated to a process, the process keeps the CPU until it releases it by terminating or by switching to the WAIT state.

Dispatcher

1. Switching context
2. Switching to user mode
3. Jumping to the propoer location in the user program to restart that program

- It should be as fast as possible

Scheduling Criteria

- **CPU utilization**: keeping the CPU as busy as possible
- **Throughput**: number of processes that are completed per time unit
- **Turnaround time:** how long it takes to execute the process. Sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU and doing I/O
- **Waiting time:** Amount of time that a process spends waiting in the ready queue
- **Response time:** the time from the submission of a request until the first response is produced (it takes to start responding, not to output the response)

Scheduling Techniques: FCFS

First-Come, First-Served

- The process that requests the CPU first is allocated the CPU first
- FIFO queue

Pro	Cons
Easy to implement	Long average waiting
	Lower CPU and device utilization
	Non preemptive
	Troublesome for time-sharing systems

Scheduling Techniques: FCFS

Example:

Process	Burst Time
P1	24
P2	3
P3	3

P1 W_t = 0ms

P2 W_t = 24ms

P3 W_t = 27ms

Average W_t = (0+24+27)/3 = 17

Scheduling Techniques: FCFS

Example:

Process	Burst Time
P2	3
P3	3
P1	24

P2 W_t = 0ms

P3 W_t = 3ms

P1 W_t = 6ms

Average W_t = (0+3+6)/3 = 3!!

Scheduling Techniques: SJF

Shortest Job First

- Associates with each process the length of the process's next CPU burst.
- The CPU is assigned to the process that has the smallest next CPU burst

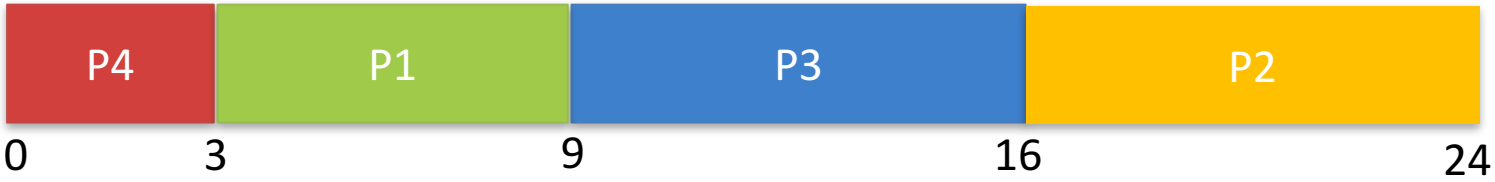
Pro	Cons
Optimal (minimum average waiting time)	Difficult to know the length of the burst
For long-term scheduler	
Preemptive or Non-preemptive	

- Preemptive version -> **Shortest-remaining-time-first**
- Predicting next CPU burst: $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$

Scheduling Techniques: SJF

Example:

Process	Burst Time
P1	6
P2	8
P3	7
P3	3



P4 W_t = 0ms

P1 W_t = 3ms

P3 W_t = 9ms

P2 W_t =16ms

Average W_t = (0+3+9+16)/4 = 7!!

Scheduling Techniques: Priority

Priority Scheduling

- A priority is associated with each process and the CPU is allocated to the process with the highest proprity (low number).
- Internal or external proprity definition

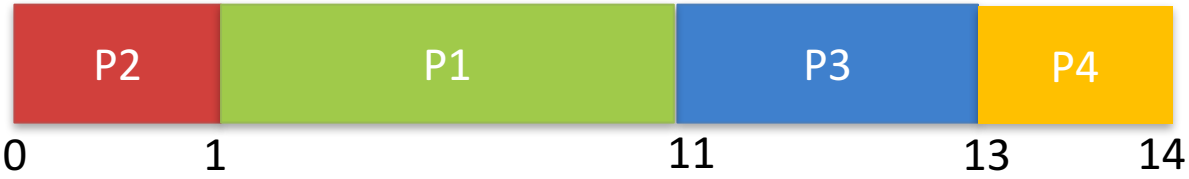
Pro	Cons
Preemptive or Non-preemptive	Starvation

- Solution for starvation -> **aging**: gradually increasing the priority of processes that wait in the system for a long time.

Scheduling Techniques: Priority

Example:

Process	Burst Time	Priority
P1	10	2
P2	1	1
P3	2	3
P4	1	4



P2 W_t = 0ms

P1 W_t = 1ms

P3 W_t = 11ms

P4 W_t = 13ms

Average W_t = (0+1+11+13)/4 = 6.75ms!!

Scheduling Techniques: RR

Round-Robin Scheduling

- A small unit of time (**time quantum**) is defined**.**
- Designed for time-sharing systems.
- Circular ready queue.
- The CPU is allocated to each process fro a time interval of up to 1 time quantum

Pro	Cons
Preemptive	Long average waiting time
Known maximum waiting time	Turnaround time depends on the quantum
	Performance depends on the size of quantum

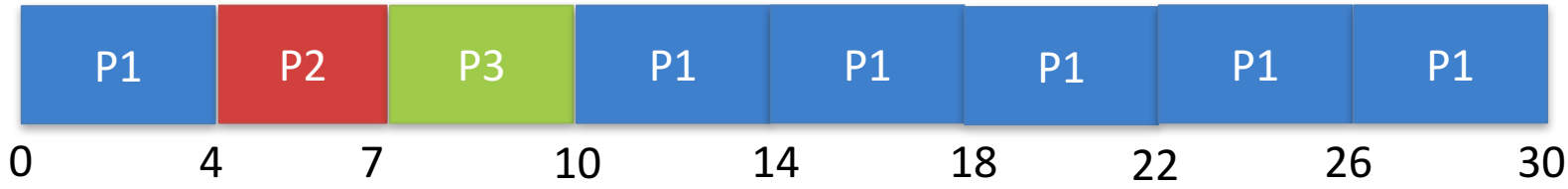
Scheduling Techniques: RR

- Two cases:
  1. The process may have the CPU burst of less than 1 time quantum -> The process itself releases the CPU
  2. The timer goes off and causes an interrupt to the OS -> A context switch is executed and the current process is put in the ready queue
- The maximum waiting time for a process is: $(n-1)\ x\ q$
- We want the time quantum to be large with respect to the context-switch time

Scheduling Techniques: RR

Example: Q=4ms

Process	Burst Time
P1	24
P2	3
P3	3



P1 W_t = (10-4) = 6ms

P2 W_t = 4ms

P3 W_t = 7ms

Average W_t = (6+4+7)/3 = 5.66ms

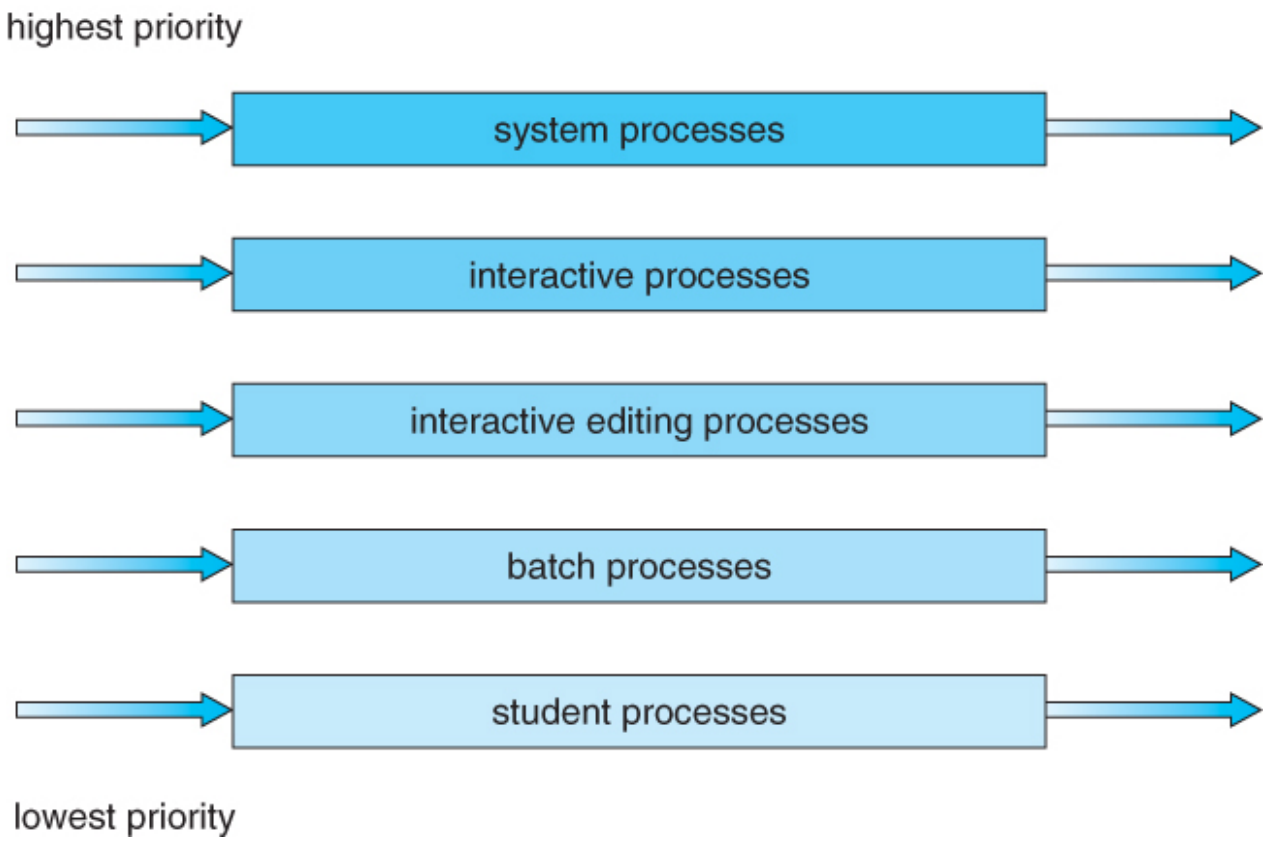
Scheduling Techniques: Multi-level

Multi-level Scheduling

- Process are classified into different groups
- The ready queue is partinioned into several separate queues
- Process are permanently assigned to one queue on some properties (e.g., memory, priority, type…)
- Scheduling among the queues is required (fixed-priority preemptive)

Pro	Cons
Low overhead	Starvation
	Fixed queue per process

Scheduling Techniques: Multi-level



Scheduling Techniques: Multi-level Feedback

Multi-level Feedback Queue Scheduling

- The process is allowed to move between queues.
- Separating processes according to the characteristics of their CPU burst
- It is defined by some parameters:
  - Number of queues
  - Scheduling algorithm for each queue
  - Method used to determine when to upgrade a process to a higher-priority queue
  - Method used to determine when to demote a process to a lower-priority queue
  - Method used to determine which queue a process will enter when that process needs service
- One of the most complex algorithm

Scheduling Techniques: Thread Scheduling

- **User-level:** Managed by the thread library
- **Kernel-level:** scheduled by the OS

- User-level threads must be mapped to an associated kernel-level thread (many-to one, many-to-many)

- **Process Contention Scope (PCS):** competition for the CPU takes place among threads belonging to the same process
  - Priority-based, priorities set by the programmer, preemptive
- **System Contention Scope (SCS)**: competition for the CPU takes place among all threads in the system

Scheduling Techniques: Pthread Scheduling

- PTHREAD_SCOPE_PROCESS: schedules threads using PCS scheduling
- PTHREAD_SCOPE_SYSTEM: schedules threads using SCS
  It creates and bind a Ligthweight Process (LWP) for each user-level thread on many-to-many systems, effectively mapping threads using the one-to-one policy

- Two functions to get and set the contention scope policy:
  - `pthread_attr_setscope(…,int scope)`
  - `pthread_attr_getscope(…, int scope)`

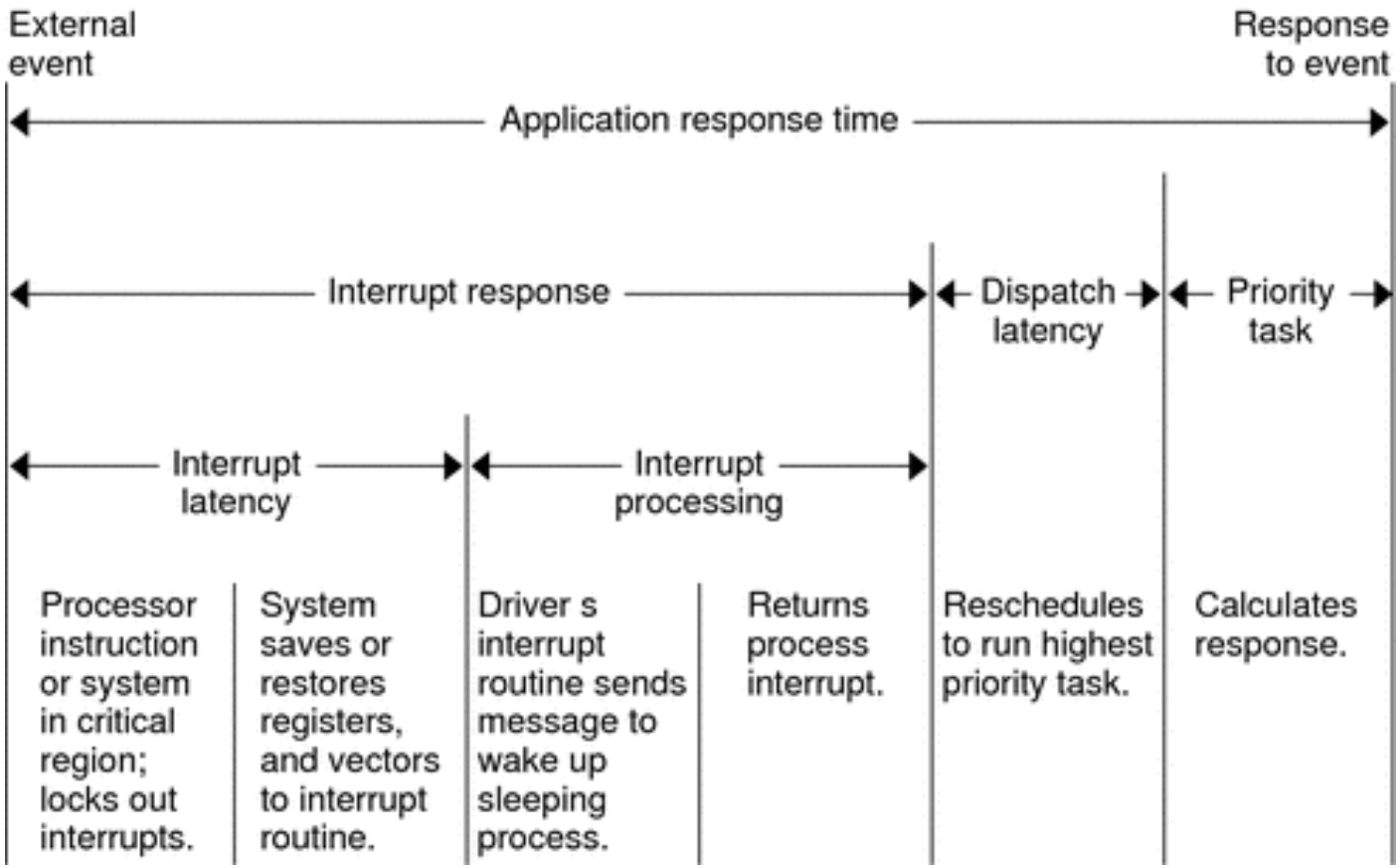
Real-Time CPU Scheduling

- **Soft real-time system:** it provides no guarantee as to when a critical real-time process will be scheduled. It guarantees only that the process will be given preference over non-critical processses.
- **Hard real-time system:** a task must be serviced by its deadline; service after the deadline has expired is the same as no service at all

Real-Time CPU Scheduling: minimizing latency

- RT system is typically waiting for an event in real-time to occur
- Event -> the system must respond to and service it as quickly as possible
- Goal -> Minimizing the latency
  - **Event latency:** the amount of time that elapses from when an event occurs to when it is serviced
  - **Interrupt latency:** period of time from the arrival of an interrupt at the CPU to the start of the routine that services the interrupt (Interrupt Service Routine, ISR) -> must be bounded
  - **Dispatch latency:** the amount of time required for the scheduling dispatcher to stop one process and start another -> preemptive kernels

Real-Time CPU Scheduling: minimizing latency



Real-Time CPU Scheduling: Priority-based

Priority-based Scheduling

- Providing a preemptive, priority-based scheduler only guarantees soft real-time functionality
- Notion of **periodic processes**
  - They require the CPU at constant intervals (periods).
  - It has fixed processing time *t*, a deadline *d* by which it must be serviced by the CPU, a period *p*
  - **Rate** is 1/*p*
- **Admission control**: the scheduler can admit the process, guaranteeing that it will complete on time, or reject the request as impossible

Real-Time CPU Scheduling: Rate-Monotonic

Rate-Monotonic Scheduling

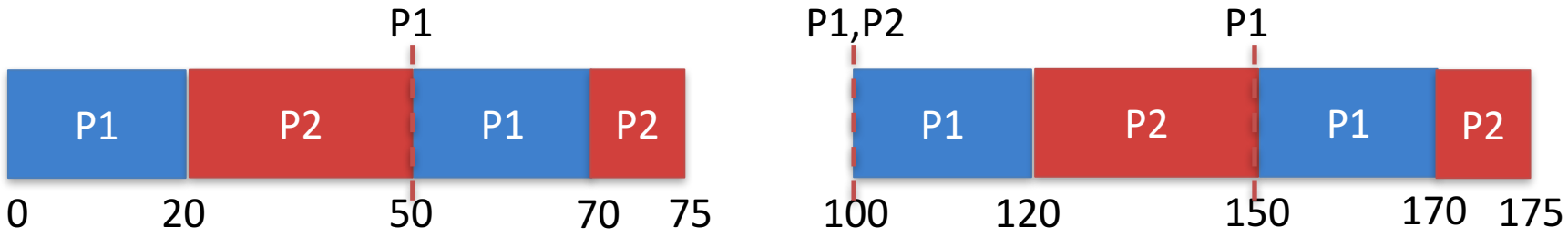
- It schedules periodic tasks using a static priority policy
- Preemptive
- Priority is the inverse of the period
- We assume that the processing time of a periodic process is the same for each CPU burst.
- Optimal: if a set of processes cannot be scheduled by this algorithm, it cannot be scheduled by any other that assigns static priorities
- It is not possible fully to maximize CPU resources.
  The worst-case CPU utilization for scheduling N processes is:

$$N(2^{\frac{1}{N}} - 1)$$

Scheduling Techniques: Rate-Monotonic

Example: Deadline: next period

Process	Period	Process Time
P1	50	20
P2	100	35



Real-Time CPU Scheduling: EDF

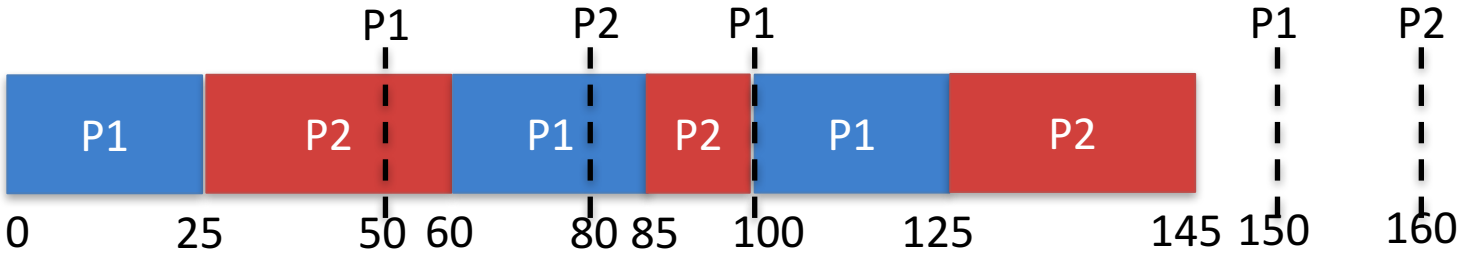
Earlier-Deadline-First Scheduling

- Dynamically assigns proprities according to deadline
- The earlier the deadline, the higher the priority
- It does not require periodic processes nor processes with a costant amount of CPU time per burst.
- Requirement: a process announces its deadline to the scheduler when it becomes runable
- Theoretical optimality -> In practice we have context switching cost!

Scheduling Techniques: EDF

Example: Deadline: next period

Process	Period	Process Time
P1	50	25
P2	80	35



Real-Time CPU Scheduling: Proportional Share

Proportional Share Scheduling

- It allocates T shares among all applications.
- It ensures that the application will have *N/T* of the total processor time
- It must work in conjunction with an admission-control policy
- A request is admitted only if sufficient shares are available.

Real-Time CPU Scheduling: POSIX

- Extensions for real-time computing
  - SCHED_FIFO: it schedules threads accoring to FCFS policy
  - SCHED_RR: it uses a round-robin policy

Exercise 2 (Thread & Parallelism)
27/02/2014

Condition	*loc* in TH_1	*loc* in TH_2
After stat. **A**	*EXISTS*	*CAN EXIST*
After stat. **C**	*CAN EXIST*	*EXISTS*
After stat. **D**	*EXISTS*	*CAN EXIST*
After stat. **E**	*EXISTS*	*DOESN'T EXIST*

Condition	*ONE* {0,1}	*TWO* {0,1,2}	*global* {0,1}
After stat. **A**	*0*	*0 / 1 / 2*	*0 / 1*
After stat. **B**	*0*	*0 / 1*	*0*
After stat. **C**	*0 / 1*	*0 / 1 / 2*	*0 / 1*
After stat. **D**	*0 / 1*	*0*	*0*

Exercise 2 (Thread & Parallelism)
27/02/2014

Situation	TH_1	TH_2	TH_3	*global*
1	*wait(TWO)*			*0 / 1*
2			*Second wait(TWO)*	*0*

Exercise 4 (Process State)
22/02/2017

Task name		IDLE	P	S	Q	*TH1*	*TH2*
	PID	1	2	3	4	*5*	*6*
	TGID	1	2	3	4	*3*	*3*
S – sem_init	0	READY	READY	EXEC	WAIT (read)		
S – pthread_create	*10*	*READY*	*READY*	*EXEC*	*WAIT (read)*	*READY*	
Interrupt from RT_clock	*20*	*READY*	*EXEC*	*READY*	*WAIT (read)*	*READY*	
Interrupt from DMA, all blocks transferred	*30*	*READY*	*READY*	*READY*	*EXEC*	*READY*	
Interrupt from ck	*40*	READY	READY	READY	READY	EXEC	N.E.
TH_1 – sem_wait	*50*	*READY*	*READY*	*EXEC*	*READY*	*WAIT (wait S)*	
S – pthread_create	*60*	*READY*	*READY*	*EXEC*	*READY*	*WAIT (wait S)*	*READY*
S – exit	*70*	*READY*	*EXEC*	*NOT EXIST*	*READY*	*NOT EXIST*	*NOT EXIST*
P– waitpid	*80*	*READY*	*WAIT (waitpid)*	*NOT EXIST*	*EXEC*	*NOT EXIST*	*NOT EXIST*
Q - exit	*90*	*READY*	*EXEC*	*NOT EXIST*	*NOT EXIST*	*NOT EXIST*	*NOT EXIST*