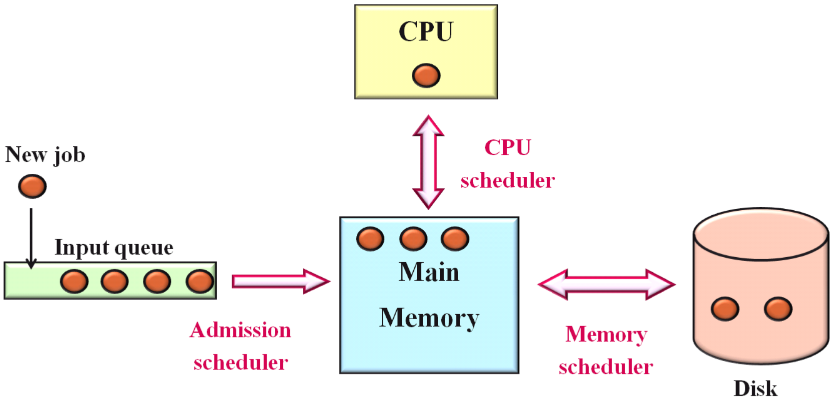
Operating Systems

III. Scheduling

Ludovic Apvrille
ludovic.apvrille@telecom-paristech.fr
Eurecom, office 470

perso.telecom-paris.fr/apvrille/OS/

Three-level Scheduling: Admission, CPU and Memory



Three-Level Scheduling: Explanations

- Admission scheduler
  - Processes are first stored into an admission queue
  - Processes are then admitted in the system
- Memory scheduler
  - Swap in / swap out
  - To be avoided: Disk storage is expensive in terms of time
- CPU scheduler
  - See next slides

CPU Scheduling Criteria

All systems

- **Fairness**: Giving each process a fair share of the CPU
- **Policy enforcement**: No process should be able to overcome the rules of the system
- **Balance**: Keeping all parts of the system busy

Cloud systems

- **CPU utilization**: Keep the CPU busy
- **Throughput**: Maximize the number of processes that are completed per time unit (hour)
- **Turnaround Time**: Minimize time between submission and termination

Scheduling Criteria (Cont.)

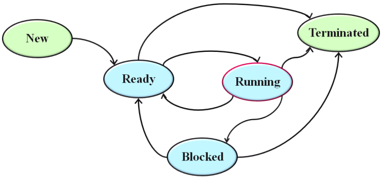
Interactive systems

- **Response time**: Respond to interactions as fast as possible
- **Proportionality**: Meet users' expectations

Real-time systems

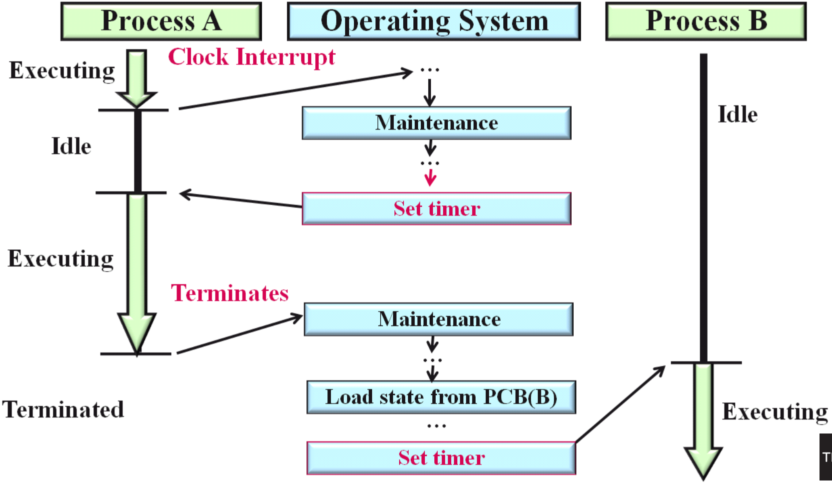
- **Meeting deadlines**: Ensure tasks will be completed before a given time
- **Latency**: Minimize the latency between an input event and its corresponding output
- **Predictability**: Know in advance whether time constraints can always be met, or not

When to Schedule?

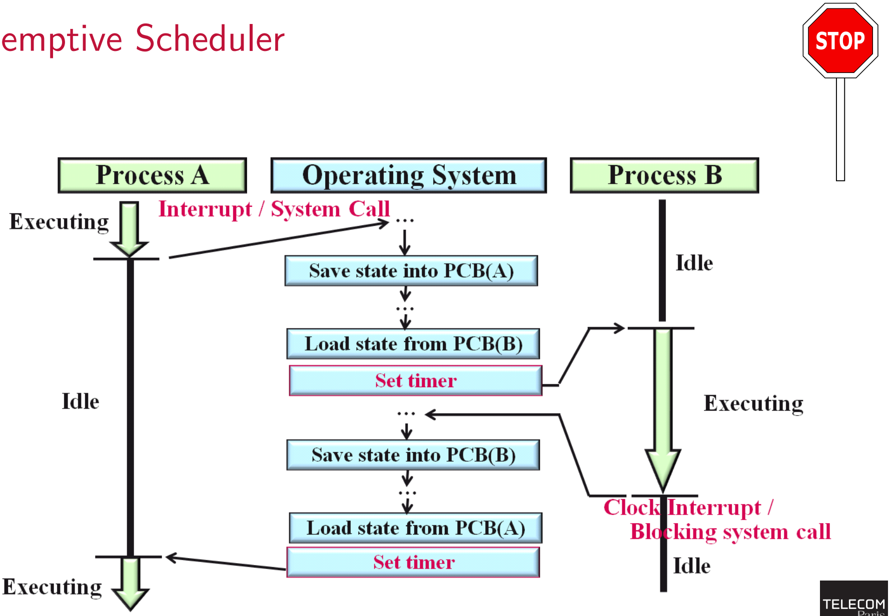


- At creation of a process
  - Run parent or child?
- At termination of a process
- When a process blocks on I/O, semaphore, etc.
- When I/O interrupt occurs
- At hardware clock interrupt
  - Non-preemptive scheduling
    - The same process is given the CPU until it blocks or voluntarily releases the CPU
    - Only choice if no timer is available
    - Windows 3.1
  - Preemptive scheduling
    - The same process can run only for a pre-defined time-interval
    - The OS sets a timer to the time-interval

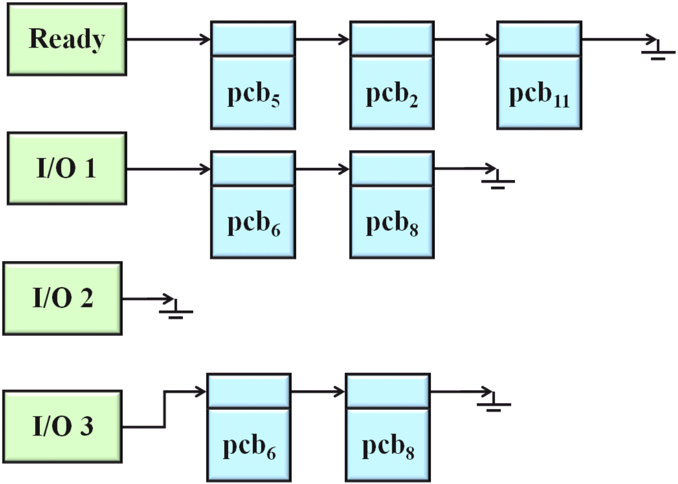
Non-Preemptive Scheduler



Preemptive Scheduler



Scheduling Implementation



Presented Scheduling Algorithms

For batch systems

- First-Come First-Served
- Shortest Job First
- Shortest Remaining Time Next

For interactive systems

- Round-Robin scheduling
- Priority-based scheduling
- Group-based scheduling
- Fair-share scheduling
- Lottery scheduling

For multiprocessors systems

Presentation of three different approaches

For RT systems

Attend lectures on RTOS to know more!

First-Come, First-Served (FCFS)

Processes are assigned the CPU in the order they request it

- Single queue of ready processes
- Easy to program, fair
- Non-preemptive scheduling

Example: *Average Wait Time* for various sets of processes

Process	p1	p2	p3
Duration	24	3	3
Arrival time (@)	0	0	0

Arrival order	AWT=?
p1, p2, p3	$\frac{0+24+27}{3} = 17$
p2, p3, p1	
p3, p2, p1	

Shortest Job First (SJF)

Scheduler selects the job with the shortest computation time

- Easy to implement
- Non-preemptive
- Optimal when processes are all ready simultaneously
  - Minimum AWT

But ... How to predict the next CPU burst time???

1. Specified amount
2. Exponential average

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$

- τ: predicted value
- t: measured computation time

Shortest Job First: Example

Example ♯1: Same arrival time

Process	p1	p2	p3	p4
Duration	8	4	4	4
@	0	0	0	0

Algo	AWT=?
FCFS (p1, p2, p3, p4)	
SJF	

Example ♯2: Various arrival times

Process	p1	p2	p3	p4	p5
Duration	2	4	1	1	1
@	0	0	3	3	3

Algo	AWT=?
FCFS	
SJF	

Can you find a better scheduling???

Shortest Remaining Time Next (SRTN)

Preemptive version of *Shortest Job First*

Memo: *SJF* is non-preemptive

Example: Comparison between *SJF* and *SRTN*

Process	p1	p2
Duration	10	1
@	0	1

Algo	Scheduling	AWT=?
SJF	p1 for 10, p2 for 1	$\frac{0+9}{2} = 4.5$
SRTN	p1 for 1, p2 for 1, p1 for 9	$\frac{1+0}{2} = 0.5$

Round-Robin (RR)

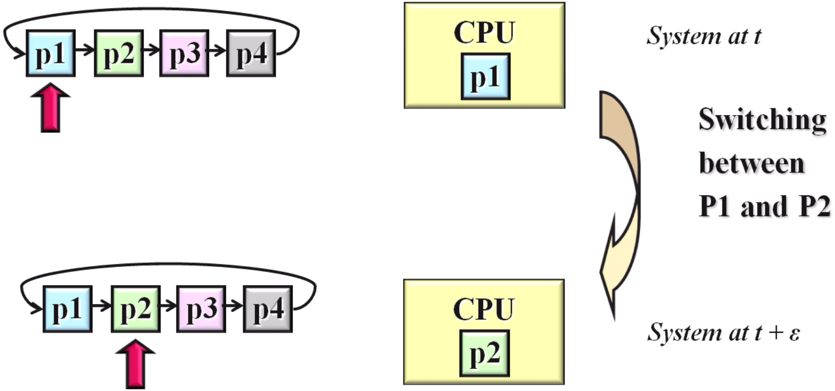
Each process is assigned a time quantum

- If the process is still running at the end of its quantum, it is preempted
  - Next process is assigned the CPU
- Widely used, easy to implement, fair

How to set the value of the quantum???

- Quantum too short? Quantum too long?
- What happens if:
  - $quantum = \epsilon$?
  - $quantum = \infty$?
- Typical quantum: 10 to 50 ms

Round-Robin: Explanations



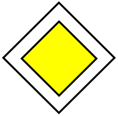
Round-Robin: Example

Process	p1	p2	p3
Duration	24	3	3
@	0	0	0
Quantum	4	4	4

p1	p2	p3	p1	p1	p1	p1	p1
0	4	7	10	14	18	22	26

AWT=???

Priority-Based Scheduling



Limitations of RR

- RR assumes all processes are of equal importance
  - For example, sending of an email vs. playing music

Priority-based scheduling: Priorities are assigned to processes

- Static priority or dynamic priority
- The process with the higher priority (may be the lower value) is chosen
  - RR between processes of the same priority

Priority-Based Scheduling: Dynamic Priorities

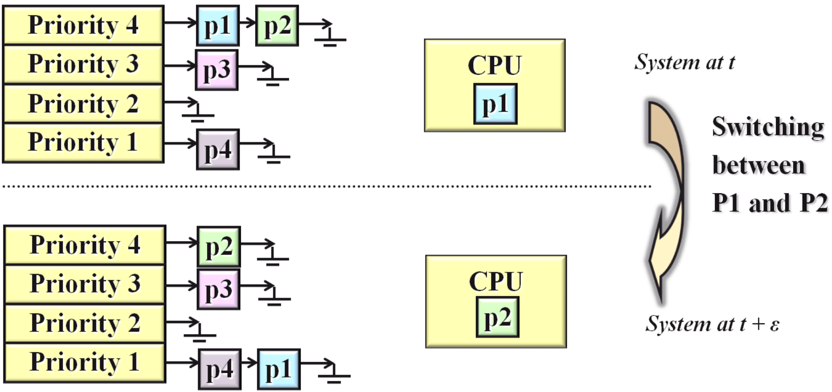
Priorities might be re-evaluated

- Avoids high-priority processes to jeopardize the CPU (starvation of low-priority processes)
- For example, priority can be set to $priority_{n+1} = \frac{quantum}{t_n}$ where t_n is the last computation time

Example: 10 ms quantum

- Process used 1ms $\rightarrow$ new priority $= 10$
- Process used 5ms $\rightarrow$ new priority $= 2$
- Process used 10ms (i.e., all quantum) $\rightarrow$ new priority $= 1$

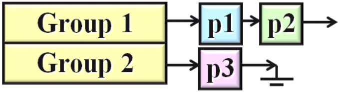
Priority-Based Scheduling (Cont.)



Group-based

Scheduling policy is applied according to groups of processes

Example: group 1 may have a quantum of 20, and group 2 a quantum of 10



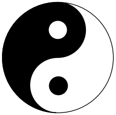
Intra-group scheduling

All algorithms previously presented can be used

Inter-group scheduling

- Fixed-priority preemptive scheduling
  - Highest priorities for foreground processes (interactivity)
- Time-slice between groups
  - e.g., 80% for group 1, 20% for group 2

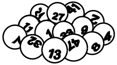
Fair-Share Scheduling



Idea: Fairness between users is taken into account first

- For example, in RR, a user with 9 processes gets more CPU than a user with 2
- Allocates a fraction of CPU to users
  - One user = one scheduling group
- Other possible notions of fairness
  - Resources, etc.

Lottery Scheduling



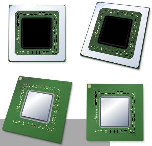
Idea: Lotto tickets are given to processes

- A random ticket is picked up, the winner gets a quantum of time
- The number of tickets received by a process is equivalent to its importance
- Processes can exchange tickets for cooperation
- Highly responsive

Example

A video server with three video streams at 10, 20, 25 frames / sec, respectively.
→ Each process is given a number of tickets equals to the frame rate i.e 10, 20 and 25

Multiprocessor Scheduling



One scheduling queue for each processor

- Symmetric system

A common ready-queue for all processors

- Asymmetric multi-processing
- Danger: synchronization problems because several processors could access to the scheduling data structures at the same time

Only one ready-queue accessed by only one processor

- No synchronization issues
- Drawback?

Windows NT Scheduling

See https://docs.microsoft.com/en-us/windows/desktop/procthread/scheduling

Basics

- Priority-based preemptive round-robin scheduling
- Raises the priority of interactive and I/O bound processes
- CPU cycle-based scheduling (since Vista)

A process runs until ...

- It is preempted by a higher priority process
- It terminates
- Its time slice expires (currently "approximately 20 ms")
- It calls a blocking system call

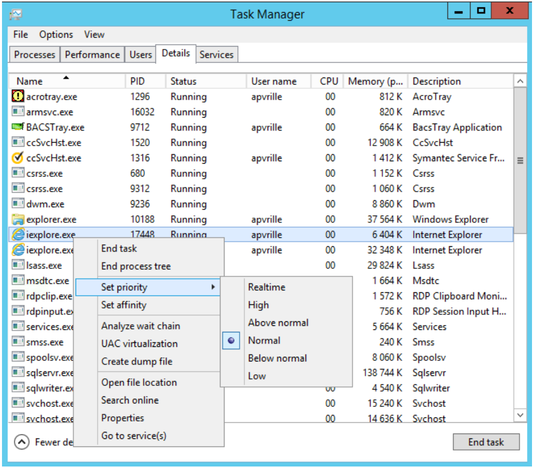
Windows NT: Priority Classes

Classes

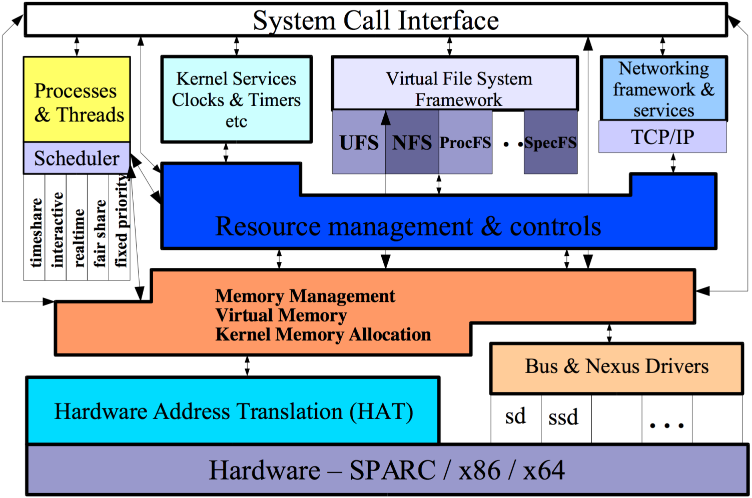
Priorities

	Real Time	High	Above Normal	Normal	Below Normal	Idle
Time-critical	31	15	15	15	15	15
Highest	26	15	12	10	8	6
Above-normal	25	14	11	9	7	5
Normal	24	13	10	8	6	4
Below-Normal	23	12	9	7	5	3
Lowest	22	11	8	6	4	2
Idle	16	1	1	1	1	1

Windows NT: Changing Classes of Processes



Solaris 10: Kernel Architecture



source = *Oracle/Sun Microsystems*

Linux: Priority-Based Scheduler

Scheduling classes

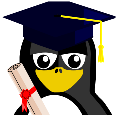
- Real-time processes: **SCHED_FIFO**, **SCHED_RR**
- Interactive and batch processes: **SCHED_OTHER**, **SCHED_BATCH**
- Low-priority processes: **SCHED_IDLE**
- One active queue for each of the 140 priorities and **for each processor**
  - Cross-CPU scheduling regularly performed (e.g., every 200 ms)

SCHED_OTHER

Round-Robin time-sharing policy with **dynamic priorities**

- Processes running for a long time are penalized

Improvements Since Kernel 2.6.23: "Completely Fair Scheduler"



- "Out of balance" tasks are given time to execute
  - Out of balance task = task has not been given a fair amount of time relative to others
- Time quantum depends upon the time balance of the task w.r.t. other tasks
- The amount of time provided to a given task is called the *virtual runtime*
  - Group-based: the virtual runtime can also be computed for a group
- Priorities are used as a decay factor for the time a task is permitted to execute

Android

- Roughly, the scheduler is based on the Linux one
  - → Fair scheduling approach
- **BUT**: fairness according to **Groups of processes**
  - Foreground/Active, visible, service, background, empty
- To reclaim resources, Android may kill processes according to their running priority