Processor Management

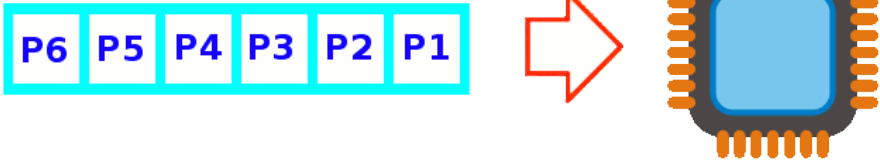
Johan Wahlström

ECM1413 Computers and the Internet

Preliminaries

- A process is a program in execution
- Processes change their state over the process lifecycle
  - ready: The process is waiting to be executed
  - running: The process is being executed
- Processor scheduling: How to allocate processors (cores) to processes?

Scheduling Algorithms

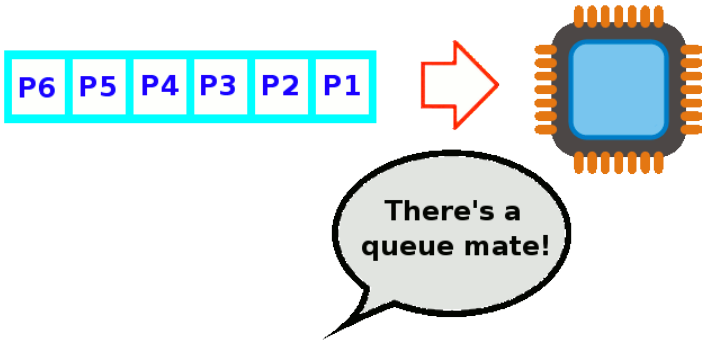


Among the most common scheduling algorithms are:

1. first come first served
2. round robin
3. shortest process next
4. multilevel queuing

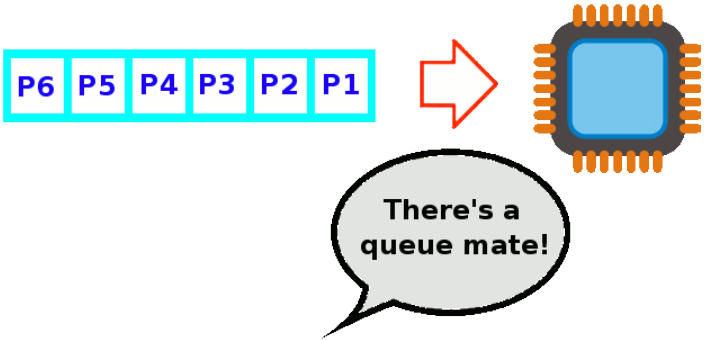
Initially, we will assume that only one processing core is available.

(1) First Come First Served



The first-come-first-served scheduling algorithm allocates the processor on the basis of creation time (like you queue at a shop with a single queue).

(1) First Come First Served – Pros



There is no unnecessary switching between processes.

You will eventually always provide processing time to a given process.

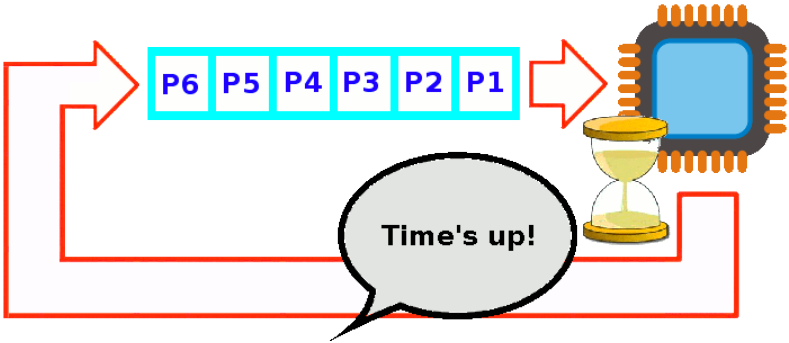
(1) First Come First Served – Cons

The average waiting time[1] is often long (P_1 is slow, but is still served first if it arrives first).

Fire come first served does not consider how quick a process is.

[1] The waiting time is the total time spent in the "ready" state.

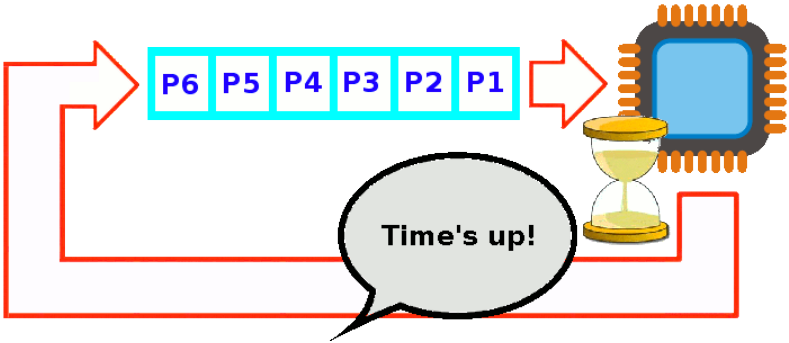
(2) Round Robin



The round robin scheduling algorithm shares the processor on the basis of equality.

Identical to first come first served, except that no process can occupy the processor longer than a predefined time length (the time quantum).

(2) Round Robin (ctd)



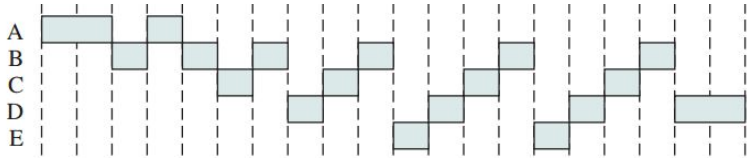
After a arriving at a processor, a process will either

- be interrupted and placed at the end of the (circular) queue
- complete before it runs out of time

New processes get priority ahead of processes that have already been given execution time.

(2) Round Robin (example)

Round-robin
(RR), $q = 1$



- When does A finish?
- When is B initially placed in the queue?
- When is E initially placed in the queue?

(2) Round Robin – Pros and Cons

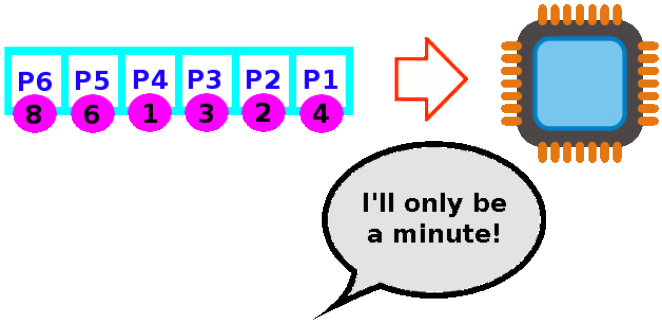
Pros:

- Distributes resources in a "fair" manner
- A quick process can pass through relatively quickly

Cons:

- Long average waiting time when processes require multiple time quanta
- Performance depends heavily on time quantum
  - Long time quantum: round robin = FCFS
  - Time quantum should be long in comparison to context-switch time
  - Most processes should finish within one time quantum

(3) Shortest Process Next



The shortest process next scheduling algorithm shares the processor on the basis of shortest predicted execution time.

Shortest process next can be either

- Preemptive (processes may be interrupted before they are done)
- Non-preemptive (processes may not be interrupted before they are done)

(3) Shortest Process Next – Pros and Cons

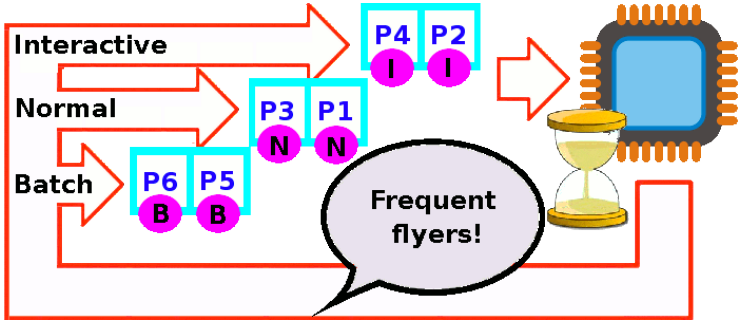
Pros:

- Gives the minimum average waiting time for a given set of processes

Cons:

- Execution time has to be estimated
  - Standard method: compute exponential average of previous processes of the same type
- Longer processes may have to wait a very long time

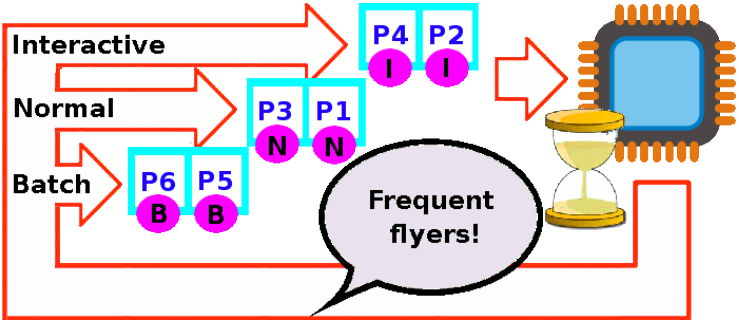
(4) Multilevel Queuing



Design choices:

- How map processes to queues?
- How determine relative priority of queues?
- Which scheduling algorithm to use within each queue?

(4) Multilevel Queuing – Mapping Processes to Queues



Example: Queues for

- interactive processes (processes with lots of I/O)
- other processes (e.g., system services)
- background processes (e.g., automatic updates, automatic backups)

(4) Multilevel Queuing – Relative Priority of Queues

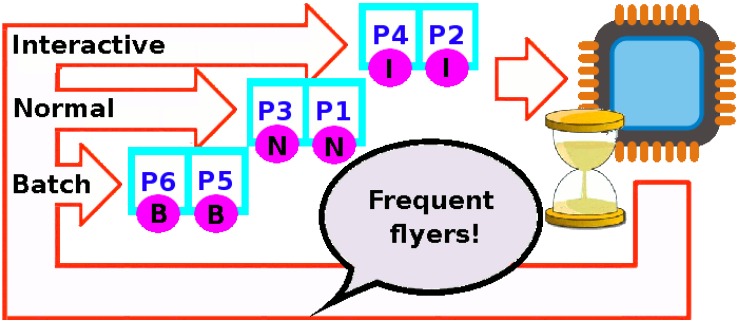
Alternative 1: Fixed-priority scheduling.

Example: interactive processes always get priority over background processes.

Alternative 2: Time-slicing.

Example: 80% of the CPU time to interactive processes, 10% to system processes, 10% to background processes.

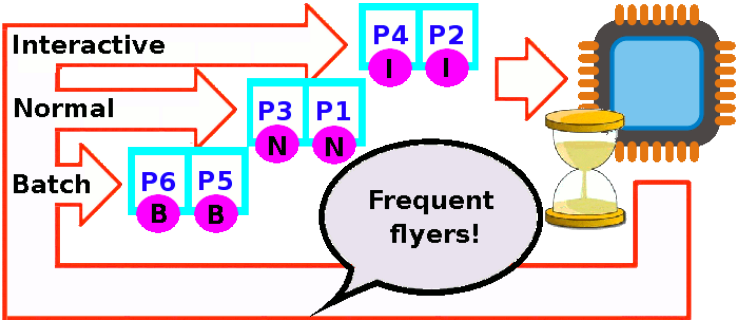
(4) Multilevel Queuing – Scheduling Algorithms Within Queues



Example:

- Round robin algorithm for interactive processes
- Round robin algorithm for all other processes
- First come first served for background processes

(4) Multilevel Queuing – Pros and Cons



Pros:

- Complex, can accommodate a range of different performance objectives

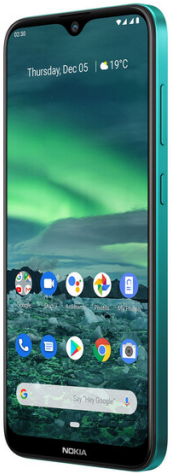
Cons:

- Complex, difficult to calibrate

Scheduling Criteria: User-Oriented Criteria

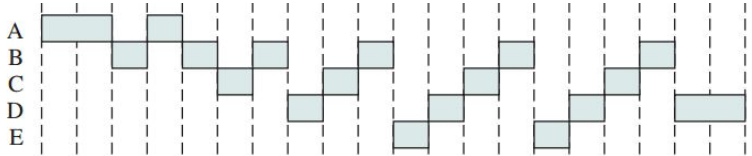
Focus on performance as perceived by an individual user.

- Turnaround time
  - The time between the submission of a process and its completion
- Response time
  - The time between the submission of a process and the first response from the process



Scheduling Criteria: User-Oriented Criteria (ctd)

Round-robin (RR), $q = 1$

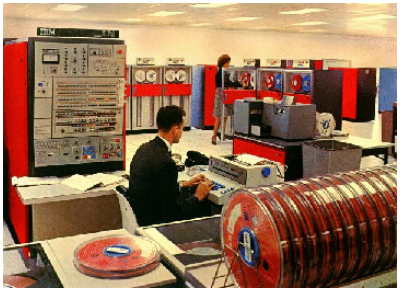


- What is the turnaround time of process B?
- What is the turnaround time of process C?

Scheduling Criteria: System-Oriented Criteria

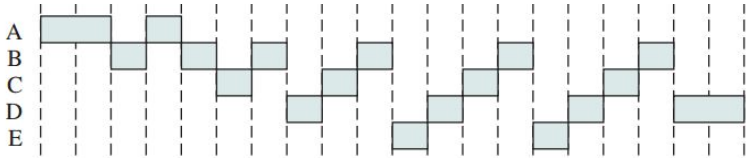
Focus on efficient utilization of the processor.

- Throughput
  - The number of completed processes per unit time
- Processor utilization
  - The percentage of time that the processor is busy



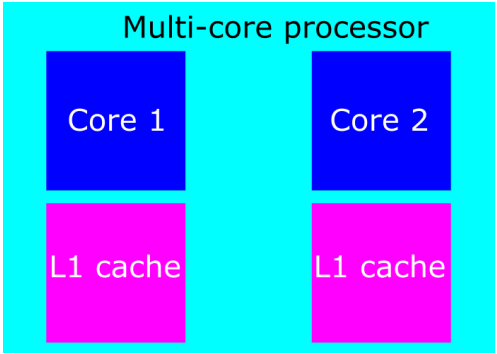
Scheduling Criteria: System-Oriented Criteria (ctd)

Round-robin (RR), $q = 1$



What is the throughput over the studied time interval?

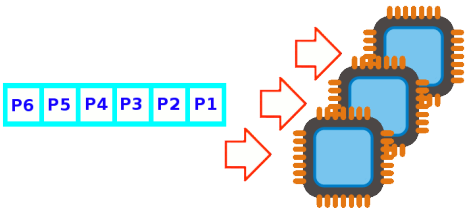
Preliminaries



- The term multiprocessor includes multi-core processors
- Each processor/core has its own cache memory

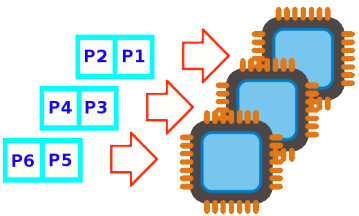
Two Approaches to Multi-Processor Scheduling

A common ready queue



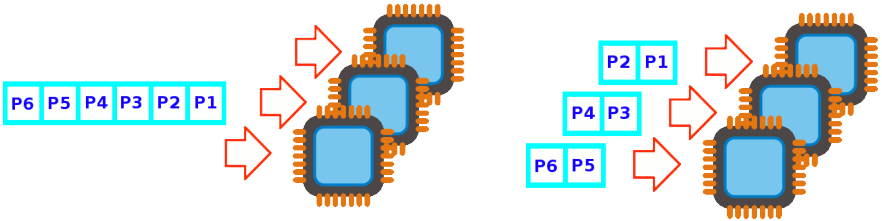
When a processor becomes available, it chooses a new process from a common queue.

Private queues



When a processor becomes available, it chooses a new process from its own private queue.

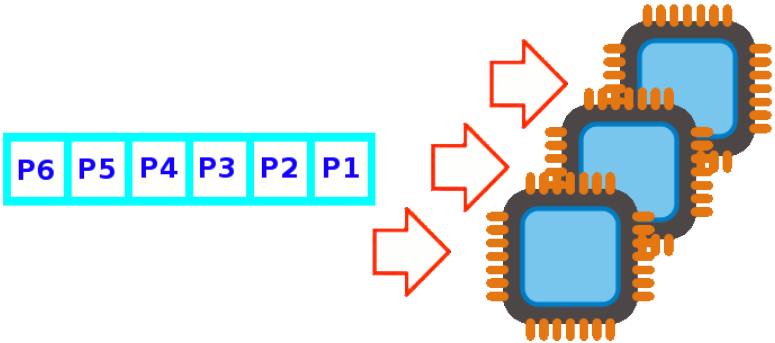
Improving the Performance of Multi-Processor Scheduling



Two methods:

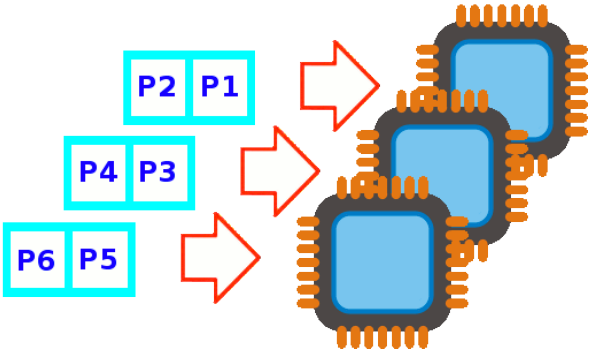
1. Load balancing
2. Processor affinity

(1) Load Balancing



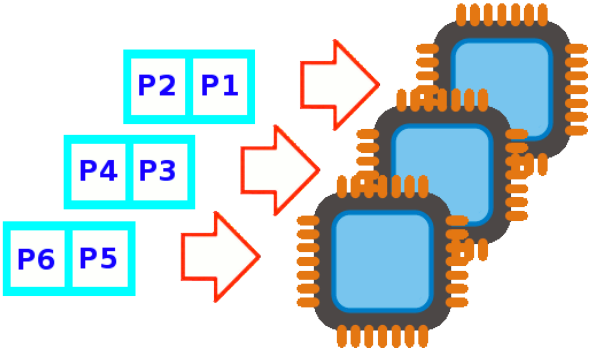
- Goal: balance processes evenly among processors
- Load balancing is automatically enforced in common ready queues

Cold/Warm Caches



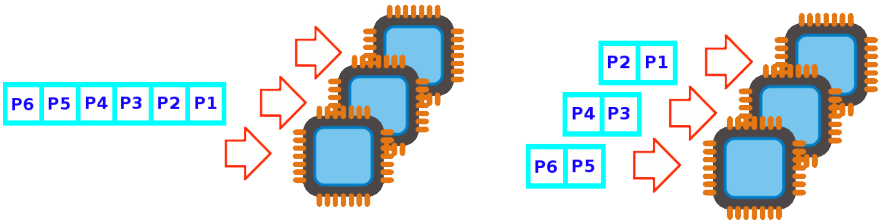
- When a process is running on a processor its associated data will populate the cache memory of that processor
- If we move the process to a new processor, the cache will no longer be warm

(2) Processor Affinity



- Goal: Keep a process running on the same processor
- Soft affinity: Only move processes if there is a good reason
- Private queues automatically enforce processor affinity

Interaction Effects

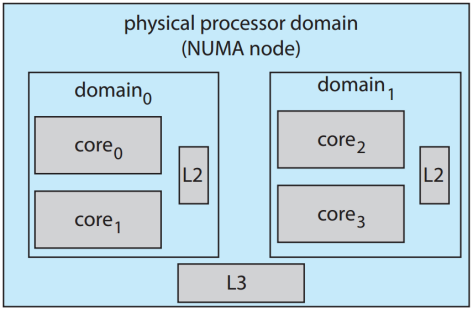


Load balancing counteracts processor affinity and vice versa.

Example

- Each core has its own L1-cache
- A pair of cores share an L2-cache
- Four cores share an L3-cache

How implement load balancing?



The Linux Scheduler

- Scheduling domain: a set of CPU cores that can be balanced against another domain
- Perform load balancing within each domain, beginning at the lowest level of the hierarchy and working upwards
- Only perform load balancing if there are severe imbalances (processor affinity)