Operating Systems

CPU Scheduling

Giorgio Grisetti
grisetti@diag.uniroma1.it

Department of Computer Control and Management Engineering
Sapienza University of Rome

CPU Scheduler

Selects which processes to grant the cpu among those in the ready queue

Invoked when

1. a process switches from running to waiting
2. terminates
3. switched from running to ready
4. switches from waiting to ready

preemptive

non-preemptive

CPU Scheduling

In a multiprogrammed system, the CPU scheduler is always invoked upon an I/O request by a process

- during I/O the CPU will be useless for the process requesting it.
- wisdom suggests to assign the CPU to another process (if any) in ready.

Non Preemptive (batch) schedulers

a running process is never evicted from CPU if it does not request an I/O or terminates.

Preemptive

a running process can be evicted and put in the ready queue(s) before it requests an I/O, since it has consumed its CPU time quantum

Dispatcher

Module that gives the CPU to the process selected by the short term scheduler.

Actions:

- switch to kernel mode and save current process state (see preamble)
- switch to user mode while restoring new process state (see postamble)

Switching costs time:

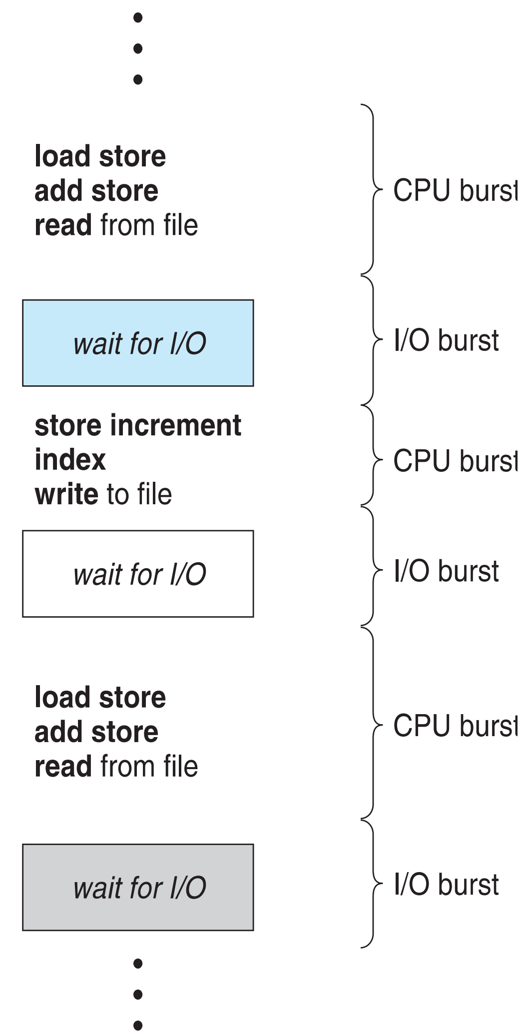
- Saving/Restoriing CPU state
- **Cache after a task switch is invalid**

CPU/IO Bursts

User programs are characterized by an alternation of

- CPU bursts
  - time interval where the CPU is used
  - CPU is the bottleneck
  - I/O rests
- I/O bursts
  - time interval where the I/O is used
  - I/O is the bottleneck
  - CPU rests

CPU burst distribution is of main concern



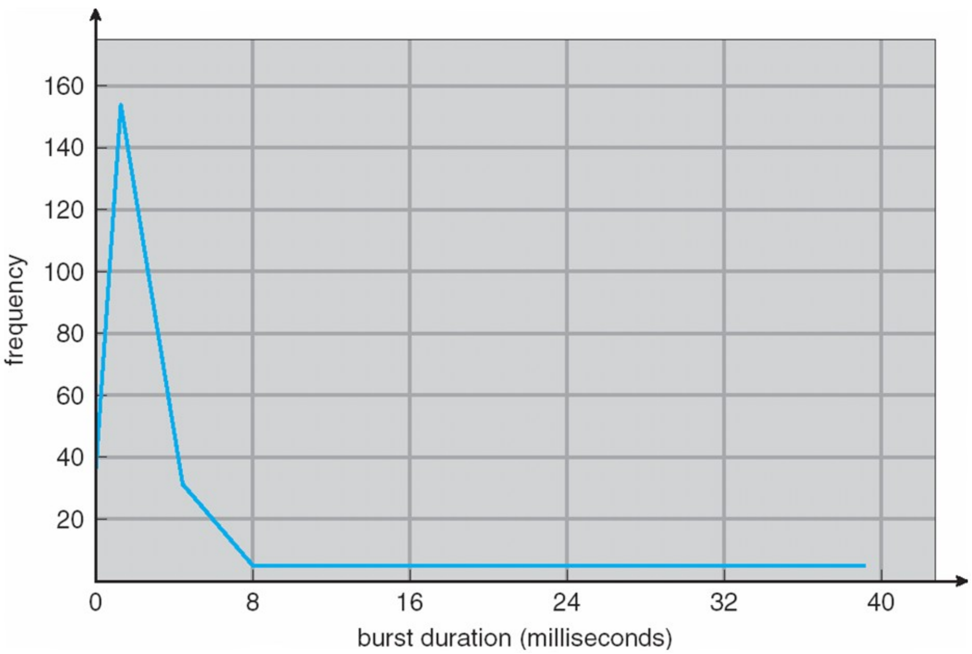
CPU Burst Distribution

Is a distribution over the length of cpu bursts.

Characterizes the CPU behavior of a program.

Can be computed by calculating an histogram while running the program

- on the x: duration of cpu burst (interval)
- on the y: # of times an interval of duration x occurs



CPU Scheduling: Metrics

The choice of which process to give the CPU next affects the behavior of the entire system.

The behavior can be monitored by the following indicators:

- **CPU utilization**: fraction of the time the CPU is used by the processes.
- **Turnaround time**: time to complete a process.
- **Throughput**: number of processes that complete in a time unit.
- **Waiting time**: how long a process has been waiting in the ready queue.
- **Response time:** how long does it take for a process that receives a command to start providing the answer

CPU Scheduling: Metrics

The choice of which process to give the CPU next affects the behavior of the entire system.

The behavior can be monitored by the following indicators:

MAXIMIZE

- **CPU utilization**: fraction of the time the CPU is used by the processes.
- **Throughput:** number of processes that complete in a time unit.

MINIMIZE

- **Turnaround time**: time to complete a process.
- **Waiting time**: how long a process has been waiting in the ready queue.
- **Response time**: how long does it take for a process that receives a command to start providing the answer

Describing a Process

To the extent of the CPU scheduler, process can be summarized by

- its arrival time
- a list of "actions", each action can be
  - a CPU burst of a given duration
  - an I/O burst on a device, of a given duration

For non interactive processes asynchronous I/O is not considered

Process P1:

- arrival, T=100
- CPU, D=5
- DISK1_IO, D=5000
- CPU, D=1
- DISK2_IO, D=5000
- CPU, D=1
- ....

Process P2:

- arrival, T=10
- CPU, D=50
- DISK1_IO, D=100
- CPU, D=1000
- DISK2_IO, D=100
- CPU, D=1000
- ...

Illustrating a Scheduler

The behavior of a scheduler executing a set of processes can be illustrated through a diagram

- Time on the X axis
- One row per I/O resource*
- One row per CPU core*

The rows of a resource are filled with a color/id corresponding to the process using that resource in that time interval.

The dispatch latencies are usually neglected.

To "paint" these diagrams it is usually helpful to use N rows, one per process.

* for simplicity we will consider 1 core and 1 I/O

First Come First Served (FCFS)

Non Preemptive

Key idea:

- pick the first process that is in ready
- when an I/O terminates, a process moves from waiting to ready, and is but in the back of the queue

Example:

Process	Burst Time	Arrival Time
P_1	24	0
P_2	3	1
P_3	3	2

- The Gantt Chart for the schedule is:

- Waiting time for P_1 = 0; P_2 = 23; P_3 = 25
- Average waiting time: (0 + 23 + 25)/3 = 16

FCFS

Suppose that the processes arrive in the order:

P_2 , P_3 , P_1

- **GANTT**

  - Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
  - Average waiting time: (6 + 0 + 3)/3 = 3

Much better than previous case

Convoy effect - short process behind long process

Consider one CPU - bound and many I/O-bound processes

Shortest Job First (SJF)

Non Preemptive

Idea:

- Select the job in ready whose **next** CPU burst is the shortest

  (keep the ready list sorted by next cpu burst)

Pros:

- Achieves the optimal minimum average waiting time for a given set of processes
- Maximizes throughput

Cons:

- Requires to know the behavior of a process (CPU and I/O bursts) in advance (often not possible in practice)

Implications:

To be used at exams when dealing with multiple exercises, provided you have a good estimate of how long each exercise will take

Approximating SJF

We can still use the SJF schema, if we have a way to "predict" the next cpu burst of the process.

The same process typically has a "cyclic" behavior

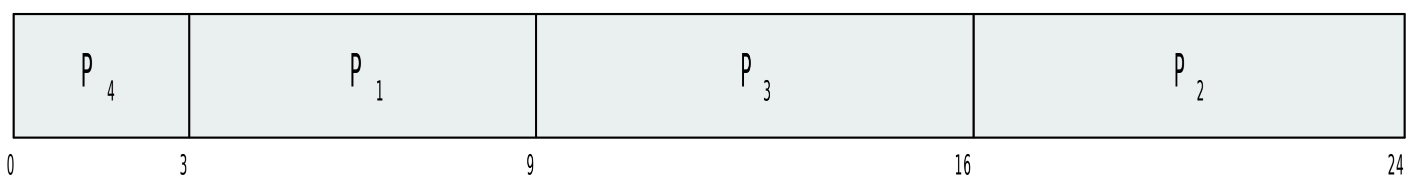
- CPU bursts of the same length, followed by IO burst of similar length
- The behavior might change during the execution of the program, but stays relatively steady for relatively long periods
- e.g. A programming IDE
  - editing: short CPU bursts, long I/O bursts (keyboard is slow)
  - compiling: long burst, medium I/Os (disk is fast)

SJF example

Assume all processes arrive at time 0

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

SJF: Predicting Next Bursts

Rule of thumb:

- the next burst will be as long as the current one

Issue:

- sudden "spikes" in the bursts might degrade overall quality of prediction

Solution

- use a discrete low pass filter to smooth the spike (aka exponential mean)

$$\underbrace{\hat{b}_{t+1}}_{\text{prediction at t+1}} = \overbrace{\alpha}^{\text{decay coefficient}} \underbrace{b_t}_{\text{measure at t}} + (1-\overbrace{\alpha})\underbrace{\hat{b}_t}_{\text{prediction at t}}$$

Priority Scheduler

Non-Preemptive

Processes are assigned a priority (int p)

- traditionally if p1<p2, p1 has highest priority

Processes in ready with highest priority are executed first

- SJF is a priority scheduler where the priority is the inverse of the next cpu burst

Issues:

- Starvation: low priority processes might never be executed

Solution:

- Aging: increase the priority as a process spends time in the ready queue

Priority Example

Process	Burst Time	Priority	Arrival Time
P_1	11	3	0
P_2	5	1	1
P_3	2	4	2
P_4	1	5	3

- Priority scheduling Gantt Chart

- Average waiting time = 8.2 msec

Preemptive Schedulers

All schemas discussed above can be extended to preemptive schedulers.

The core of a preemptive scheduler is a routine that can put in ready a running process that has not yet requested an I/O.

Each process gets a small unit of CPU time (the cpu quantum q, usually 10/100 ms). If after this time the process is still using the CPU, it is preempted and put in the ready queue.

- **If evicted process is put at the end -> Round Robin Scheme (RR)**

Scheduler is invoked upon

- I/O requests
- timer interrupt

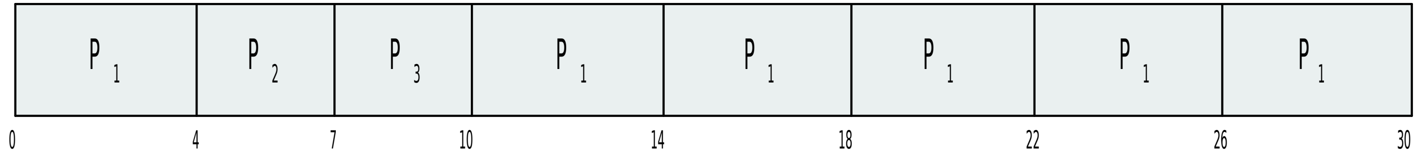
With N processes in ready and a quantum of q, no process waits more than $(N-1)*q$

RR example

Assuming q=4

Process	Burst Time
P_1	24
P_2	3
P_3	3

The Gantt chart is:

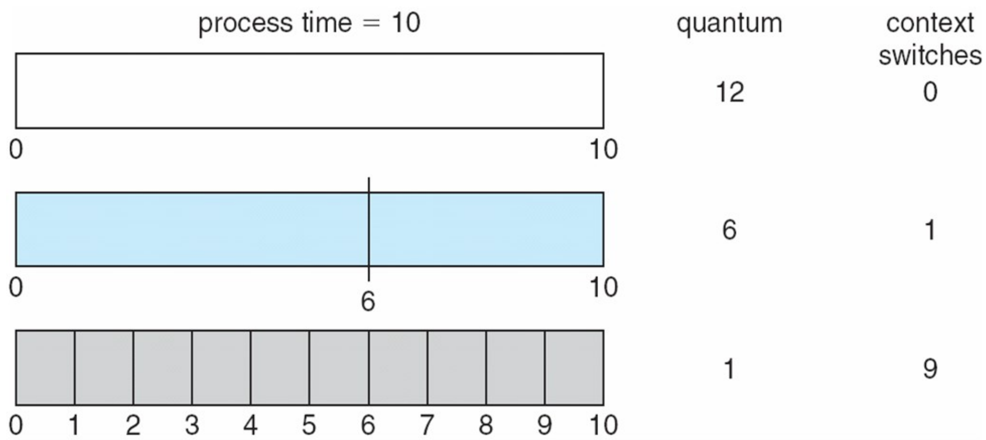


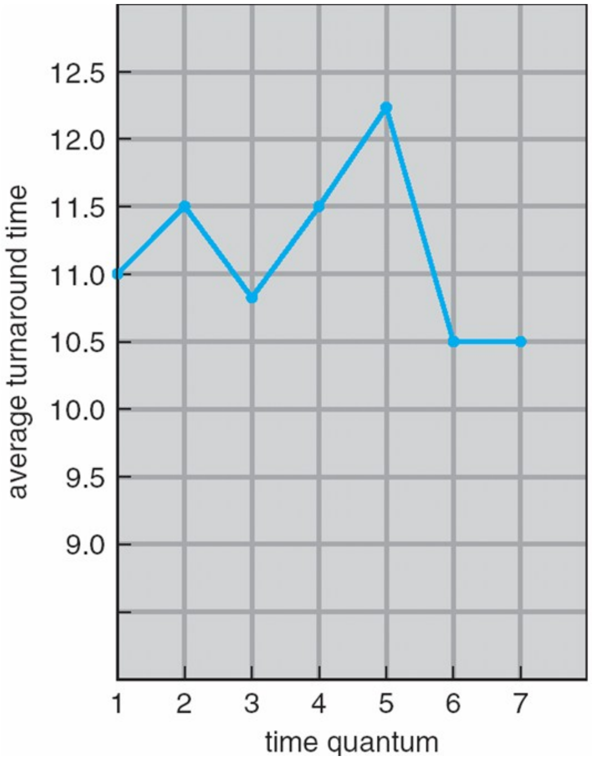
Considerations:

- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

q and Context Switch Time

- The smaller the quantum, the more the context switches
- Too many context switches might waste CPU
- Usually q chosen so that 80% of cpu bursts are shorter than q





process	time
P_1	6
P_2	3
P_3	1
P_4	7

Matching Application's Needs

The scheduling metric to be optimized depends on the application

- Interactive processes (e.g. editors): response time
- Batch Processes: (e.g. building an application) throughput

The scheduler is responsible of optimizing the parameters for each application.

Idea:

- Threat applications differently

Multilevel Queue

Ready queue is partitioned into separate queues, eg:

- **foreground** (interactive)
- **background** (batch)

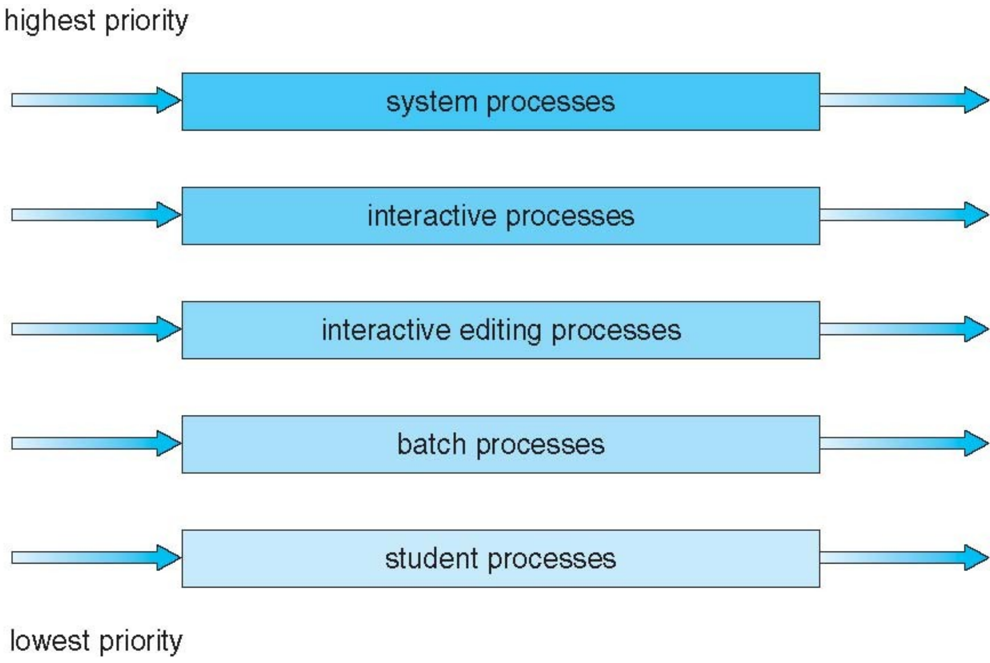
Process permanently in a given queue

Each queue has its own scheduling algorithm:

- foreground – RR
- background – FCFS

Scheduling must be done between the queues:

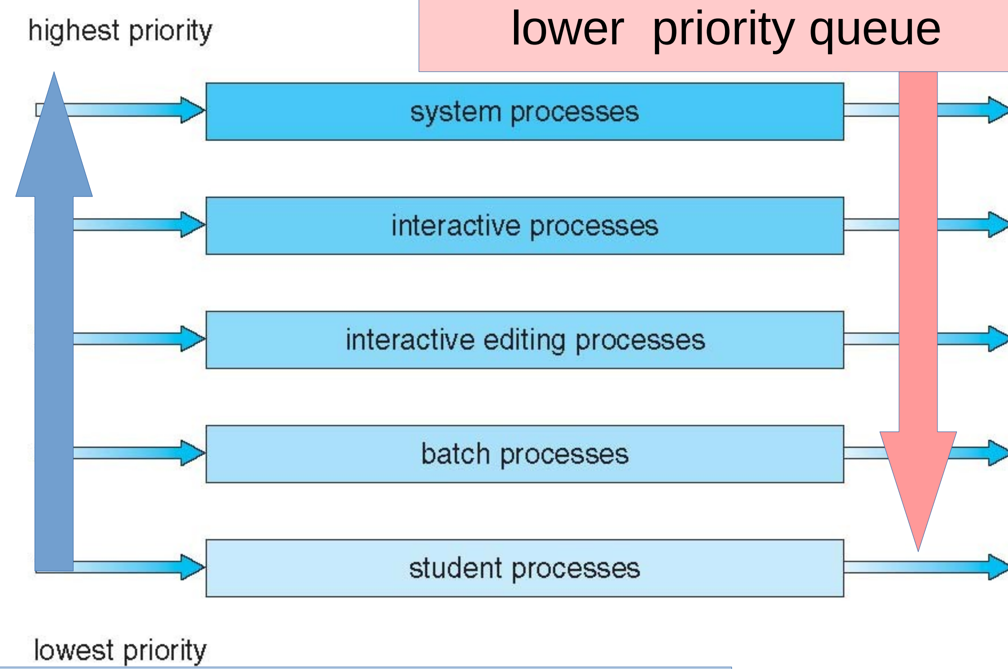
- Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
- Time slice – each queue gets a certain amount of CPU time which is distributed to its processes; i.e., 80% to foreground in RR
- 20% to background in FCFS



Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

when a process consumes steadily all its quantum, is demoted to the lower priority queue



when a process uses steadily less than its quantum, is demoted to the lower priority queue

Multiple Processor Scheduling

CPU scheduling more complex when multiple CPUs are available

Homogeneous processors within a multiprocessor

Asymmetric multiprocessing – only one processor accesses the system data structures, alleviating the need for data sharing

Symmetric multiprocessing **(SMP)** – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes

- Currently, most common

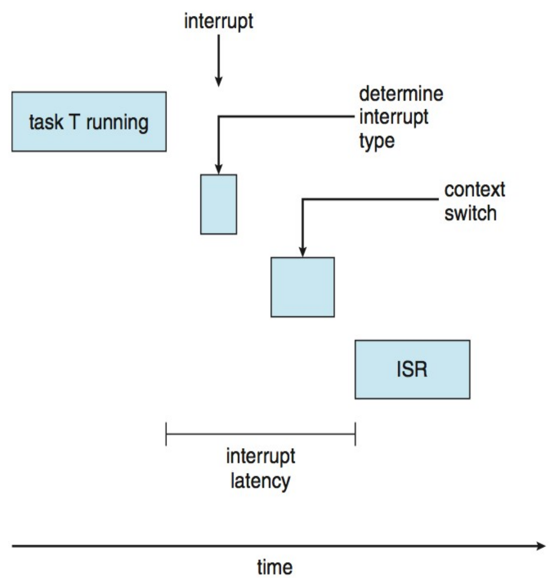
Processor affinity – process has affinity for processor on which it is currently running

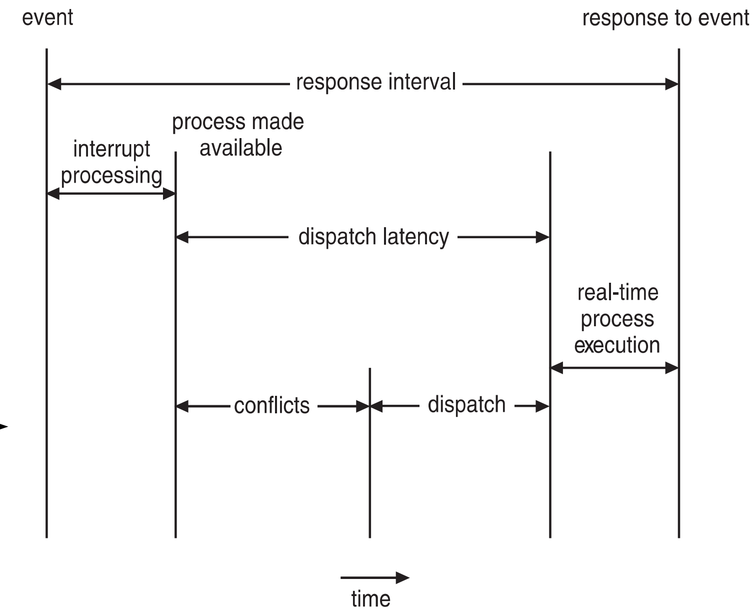
- **soft affinity**
- **hard affinity**
- Variations including **processor sets**

Real Time CPU Scheduling

Soft real-time systems no guarantee as to when critical real-time process will be scheduled

- **Hard real-time systems** task must be serviced by its deadline
- Two types of latencies affect performance
  - Interrupt latency – time from arrival of interrupt to start of routine that services interrupt
  - Dispatch latency – time for scheduler to take current process off CPU and switch to another





Real Time Priority Scheduling

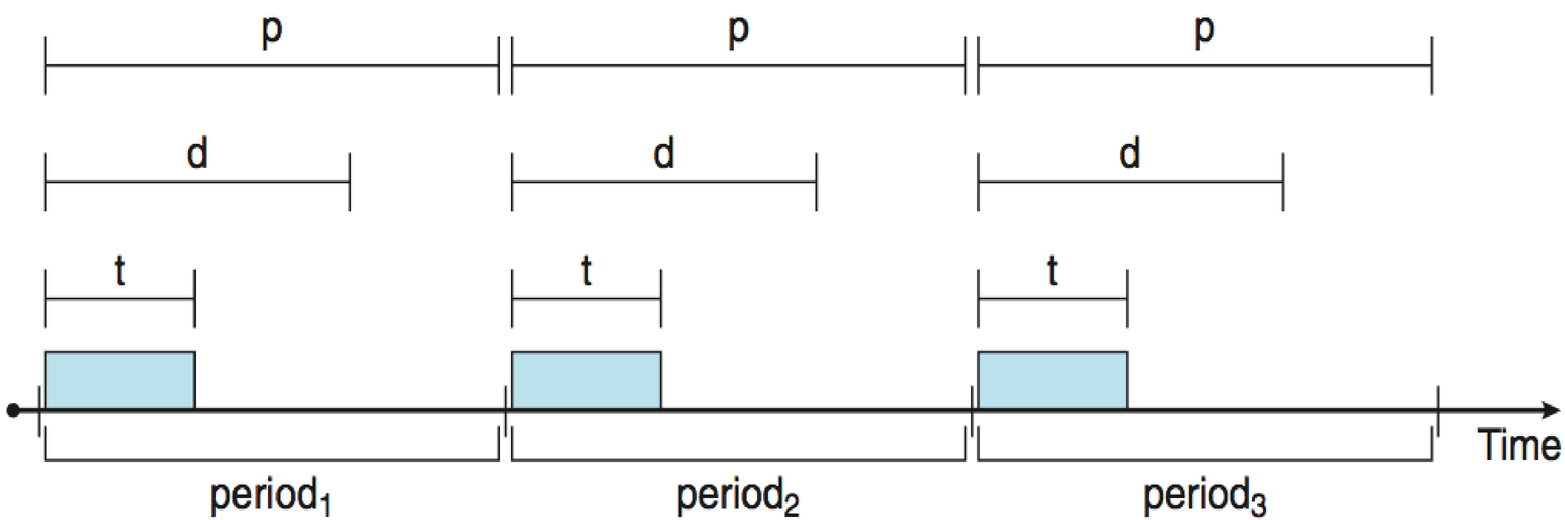
For **soft real-time** scheduling, scheduler must support preemptive, priority-based scheduling

For **hard real-time** the scheduler must also provide ability to meet deadlines

- Processes have new characteristics: **periodic** ones require CPU at constant intervals
- Has processing time t, deadline d, period p
- $0 \leq t \leq d \leq p$
- **Rate** of periodic task is $1/p$

$$U = \sum_i \frac{t_i}{d_i} < 1$$

this tells if I can schedule a set of processes in EDF



EDF: pick a process with earliest deadline VS fixed priorities

check paper

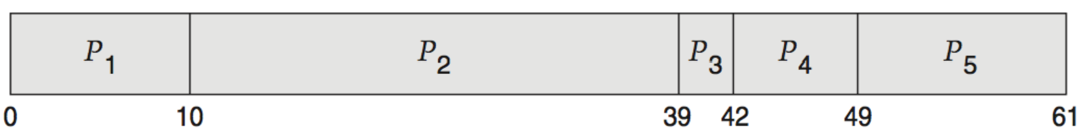
Evaluating a Scheduler

- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- **Deterministic modeling**
  - Type of **analytic evaluation**
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving at time 0:

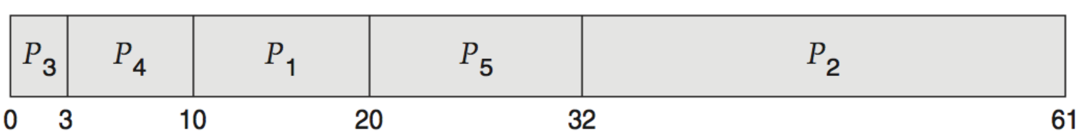
Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

Deterministic Evaluation

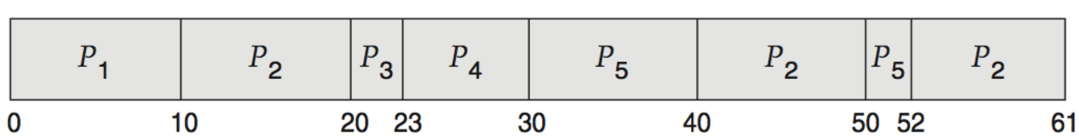
- For each algorithm, calculate minimum average waiting time
- Simple and fast, but requires exact numbers for input, applies only to those inputs
  - FCS is 28ms:



  - Non-preemptive SFJ is 13ms:



  - RR is 23ms:



Queuing Models

Describes the arrival of processes, and CPU and I/O bursts probabilistically

- Commonly exponential, and described by mean
- Computes average throughput, utilization, waiting time, etc

Computer system described as network of servers, each with queue of waiting processes

- Knowing arrival rates and service rates
- Computes utilization, average queue length, average wait time, etc

LITTLE's Formula

- Little's law – in steady state, processes leaving queue must equal processes arriving, thus:
  $n = \lambda \times W$
  - Valid for any scheduling algorithm and arrival distribution
- n = average queue length
- W = average waiting time in queue
- λ = average arrival rate into queue
- For example, if on average 7 processes arrive per second, and normally 14 processes in queue, then average waiting time per process = 2 seconds

Simulations

Queueing models limited

Simulations more accurate

- Programmed model of computer system
- Clock is a variable
- Gather statistics indicating algorithm performance
- Data to drive simulation gathered via
  - Random number generator according to probabilities
  - Distributions defined mathematically or empirically
  - Trace tapes record sequences of real events in real systems