CSE 4/521 Introduction to Operating Systems

Lecture 9 – CPU Scheduling II

(Scheduling Algorithms, Thread Scheduling, Real-time CPU Scheduling)

Summer 2018

Overview

Objective:

1. To describe priority scheduling and round-robin scheduling algorithm
2. To study options provided by Pthreads thread scheduling library.
3. To examine how real-time scheduling works

- Scheduling Algorithms
- Thread Scheduling
- Real-Time CPU Scheduling

Recap

- Basic Concepts
  - CPU burst and I/O burst, CPU Scheduler
- Scheduling Criteria
  - 5 criteria – CPU utilization, Throughput, Turnaround time, Waiting time, Response time
- Scheduling Algorithms
  - First-come-first-serve, Shortest-Job-First

Questions

1. What does each of the 6 scheduling criteria mean? Which needs to be maximized and minimized? (Easy)

2. Since the time period of the future is not known, how to guess the CPU burst time in SJF? (Medium)

3. Which of 2 scheduling algorithm lead to starvation? Or deadlock? *[Difference between deadlock and starvation]* (Hard)

Overview

- Scheduling Algorithms
- Thread Scheduling
- Real-Time CPU Scheduling

Overview

- Scheduling Algorithms
- Thread Scheduling
- Real-Time CPU Scheduling

Scheduling Algorithms – Priority Scheduling

- A priority number (integer) is associated with each process

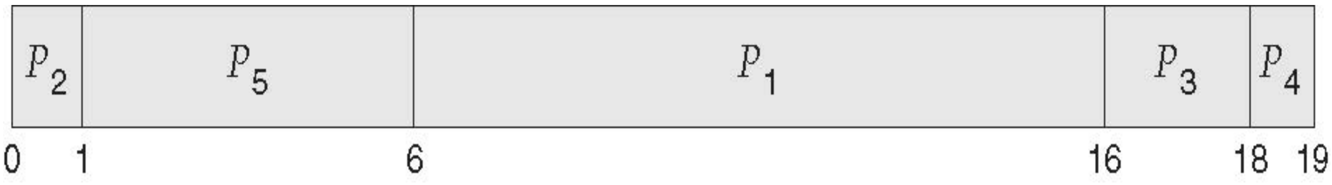
- The CPU is allocated to the process with the highest priority (smallest integer = highest priority)
  - Preemptive
  - Nonpreemptive

- Problem = Starvation – low priority processes may never execute
- Solution = Aging – as time progresses increase the priority of the process

Scheduling Algorithms – Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

Scheduling Algorithms – Round-Robin

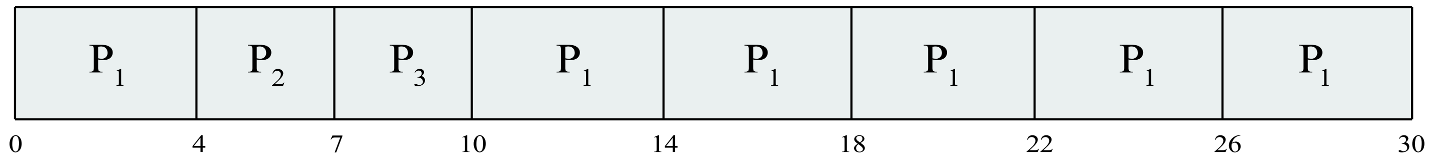
- Each process gets a small unit of CPU time (time quantum q). After time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units.
- Performance:
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Scheduling Algorithms – Example of Round-Robin

Process	Burst Time
P_1	24
P_2	3
P_3	3

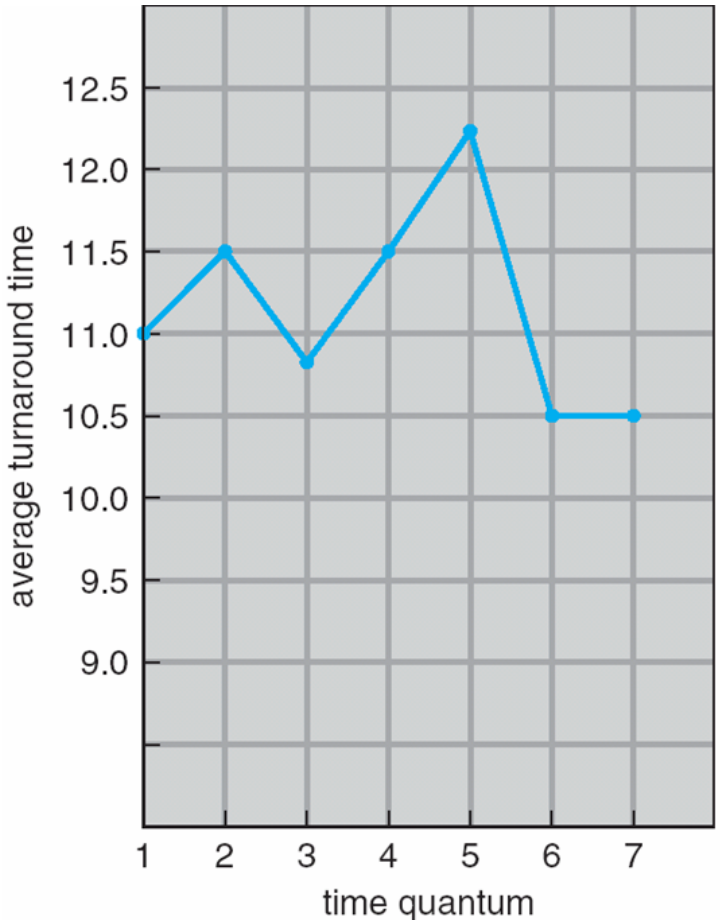
q = 4

- The Gantt chart is:



- 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

Scheduling Algorithms – Round-Robin Concerns



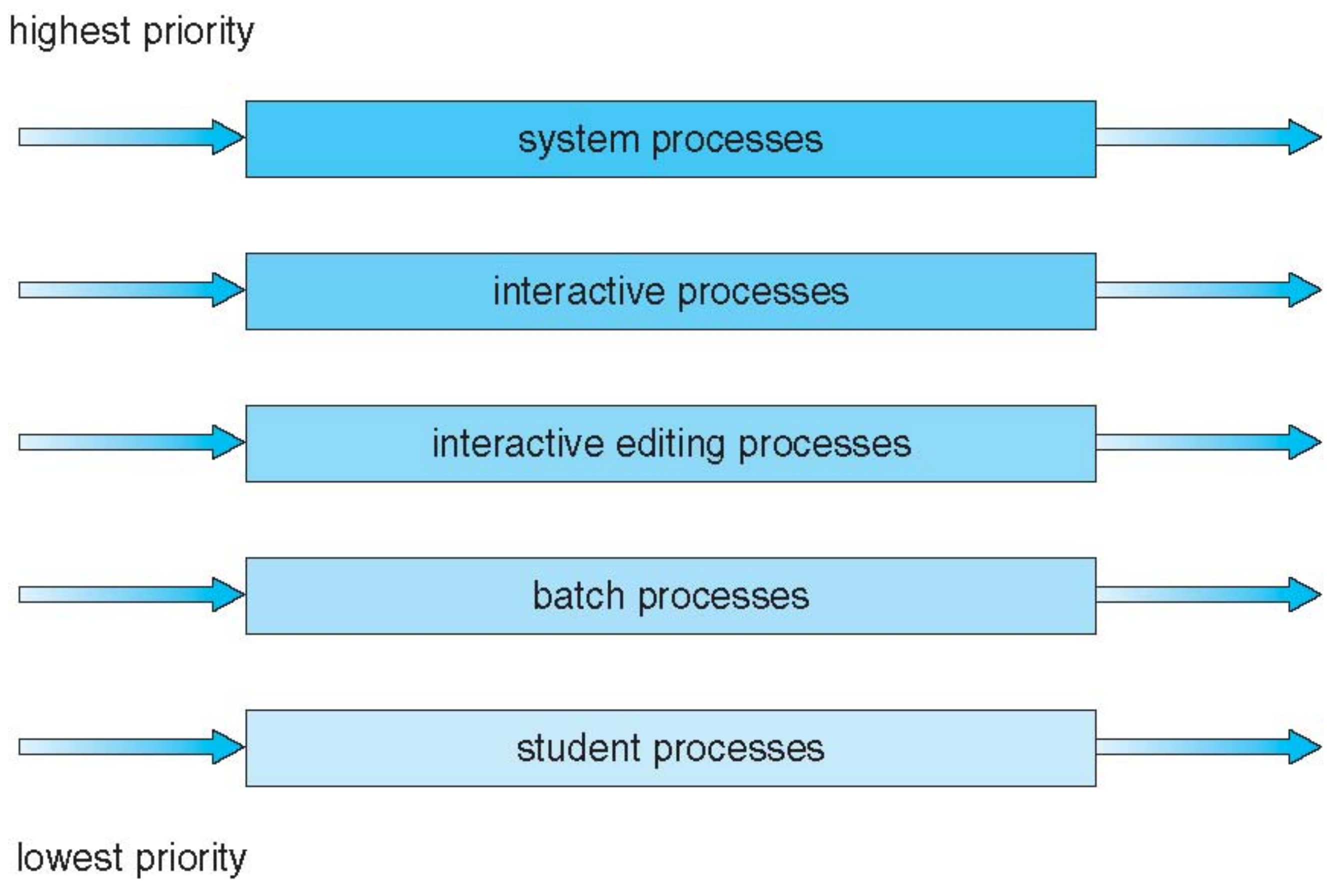
process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts
should be shorter than q

Scheduling Algorithms – Multilevel Queue

- Ready queue is partitioned into separate queues, eg:
  - foreground (interactive)
  - background (batch)
- Process permanently put into a queue
- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues:
  - Time slice – each queue gets a certain amount of CPU time, i.e., 80% to foreground in RR , 20% to background in FCFS

Scheduling Algorithms – Multilevel Queue

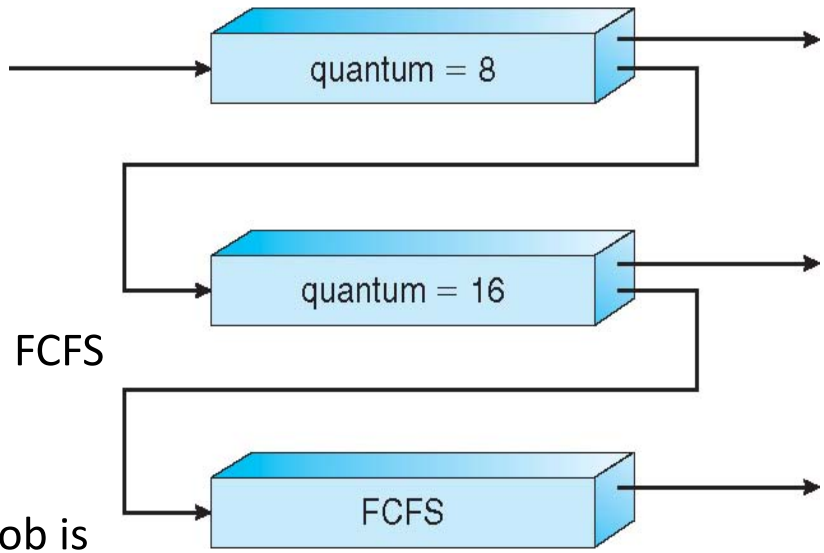


Scheduling Algorithms – Multilevel Feedback Queue

- A process can move between the 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

Scheduling Algorithms – Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS



- Scheduling:
  - A new job enters queue Q_0 which is served FCFS
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q_1
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2

Overview

- Scheduling Algorithms
- Thread Scheduling
- Real-Time CPU Scheduling

Thread Scheduling

- Many-to-one and many-to-many models, thread library schedules user-level threads to run in kernel:
- Two methods:
  - Process-contention scope (PCS) : Scheduling competition between threads of same process
  - System-contention scope (SCS) – Scheduling competition among all threads in system

Thread Scheduling - Pthreads

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
int main(int argc, char *argv[]) {
   int i, scope;
   pthread_t tid[NUM THREADS];
   pthread_attr_t attr;
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* first inquire on the current scope */
   if (pthread_attr_getscope(&attr, &scope) != 0)
      fprintf(stderr, "Unable to get scheduling
scope\n");
   else {
      if (scope == PTHREAD_SCOPE_PROCESS)
         printf("PTHREAD_SCOPE_PROCESS");
      else if (scope == PTHREAD_SCOPE_SYSTEM)
         printf("PTHREAD_SCOPE_SYSTEM");
      else
         fprintf(stderr, "Illegal scope
value.\n");
   }
```

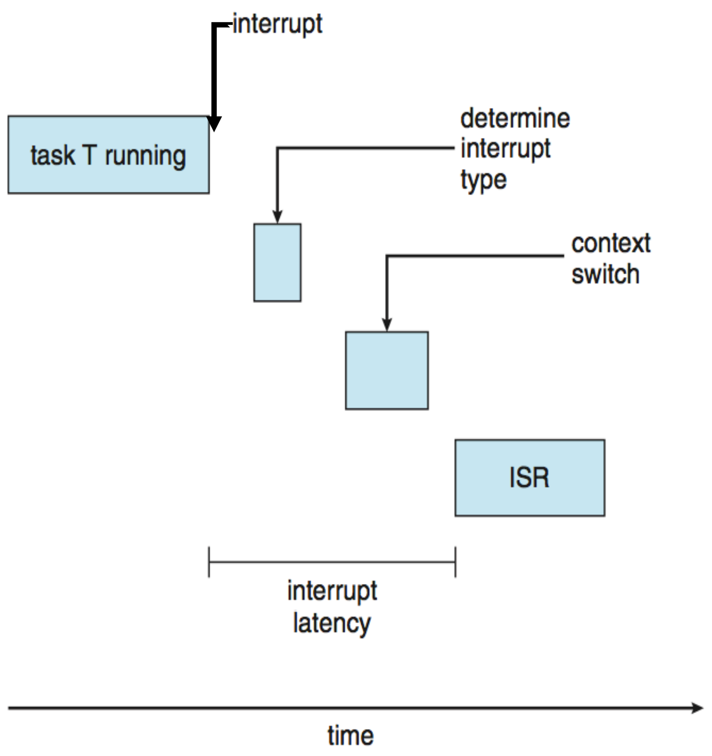
```
 /* set the scheduling algorithm to PCS or SCS */
pthread_attr_setscope(&attr,PTHREAD_SCOPE_SYSTEM);
   /* create the threads */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_create(&tid[i],&attr,runner,NULL);
   /* now join on each thread */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_join(tid[i], NULL);
}
/* Each thread will begin control in this function
*/
void *runner(void *param)
{
   /* do some work ... */
   pthread_exit(0);
}
```

Overview

- Scheduling Algorithms
- Thread Scheduling
- Real-Time CPU Scheduling

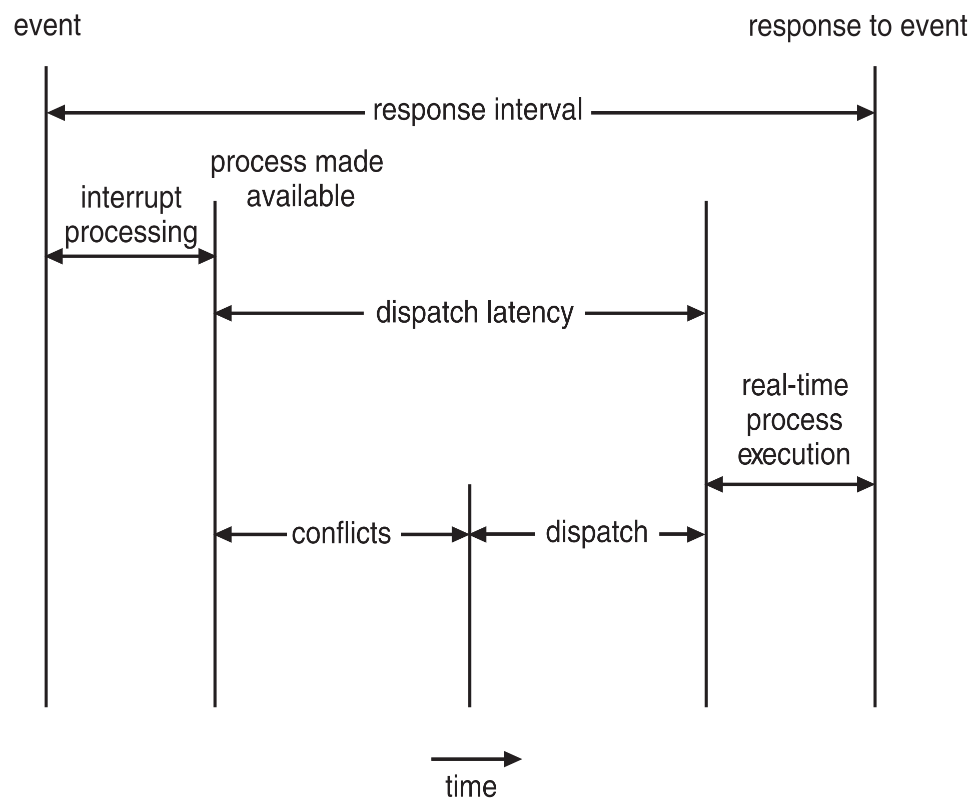
Real-Time CPU Scheduling

- Can present obvious challenges
- **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
  1. Interrupt latency – time from arrival of interrupt to start of routine that services interrupt
  2. Dispatch latency – time for scheduler to take current process off CPU and switch to another



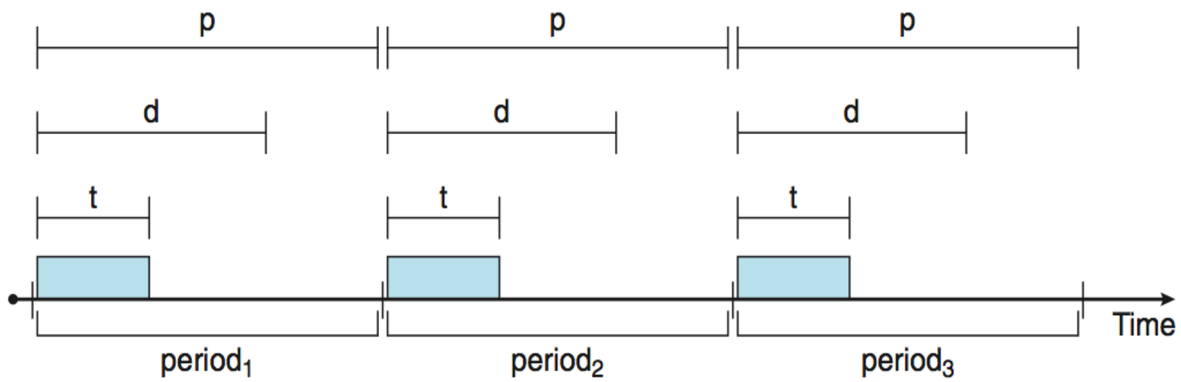
Real-Time CPU Scheduling

- Conflict phase of dispatch latency:
  1. Preemption of any process running in kernel mode
  2. Release by low-priority process of resources needed by high-priority processes



Real-Time CPU Scheduling : Priority-based Scheduling

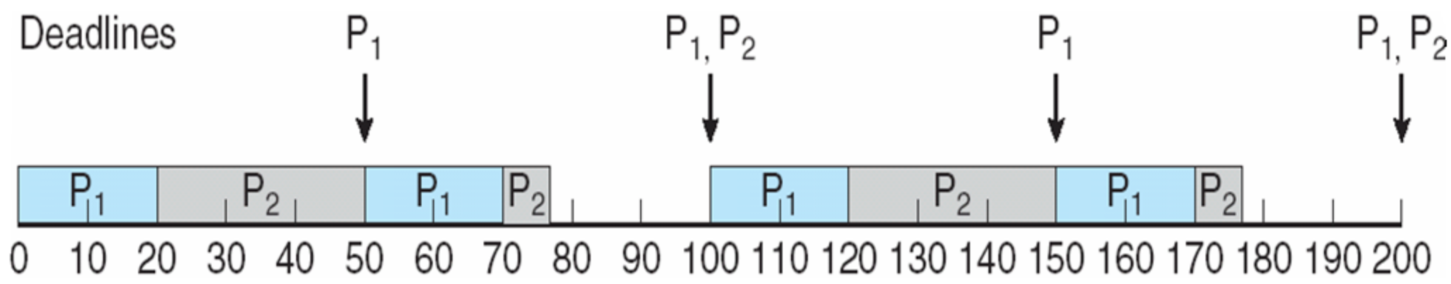
- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
  - But only guarantees soft real-time
- For hard real-time must also provide ability to meet deadlines
- Processes have new characteristics: periodic ones require CPU at constant intervals
  - Has processing time *t*, deadline *d,* period *p*
  - $0 \leq t \leq d \leq p$
  - Rate of periodic task is $1/p$



Real-Time CPU Scheduling : Rate Monotonic Scheduling

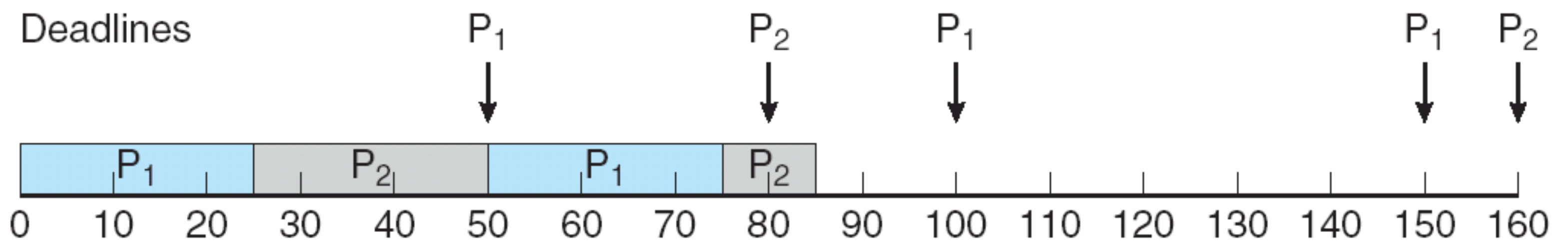
- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- P_1 is assigned a higher priority than P_2.

P1 = 50
P2 = 100
t1 = 20
t2 = 35
Deadline is to complete before next period



Real-Time CPU Scheduling :
Missed Deadlines with Rate Monotonic Scheduling

P1 = 50
P2 = 80
t1 = 25
t2 = 35
Deadline is to complete before next period

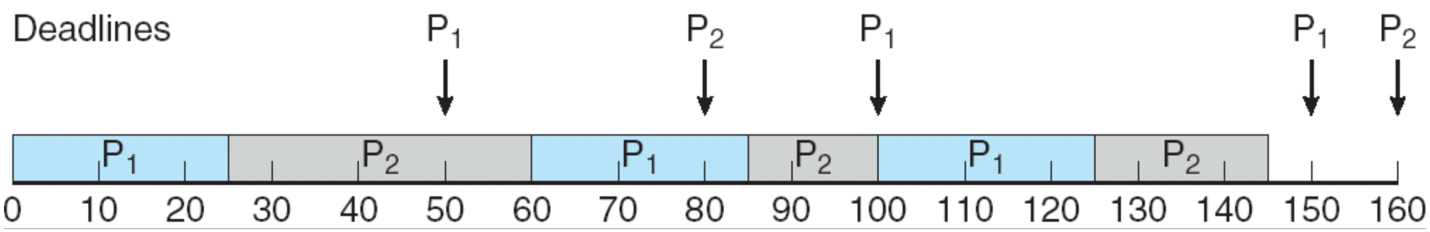


There is no pre-emption in Rate Monotonic Scheduling

Real-Time CPU Scheduling: Earliest Deadline First Scheduling (EDF)

- Priorities are assigned according to deadlines:
  - The earlier the deadline, the higher the priority
  - The later the deadline, the lower the priority

P1 = 50
P2 = 80
t1 = 25
t2 = 35
Deadline is to complete before next period



Credits for slides

Silberschatz, Galvin and Gagne