Operating Systems
Lecture #7

CPU SCHEDULING

1. INTRODUCTION

CPU scheduling is the basis of multiprogrammed operating systems. By switching the cpu between processes, the operating system can make the computer more productive.

2. SCHEDULING CONCEPTS

Scheduling is a fundamental operating system function, since almost all computer resources are scheduled before use. The cpu is one of the primary computer resources. Thus the scheduling is central to operating system design.

1) BASIC COMPONENTS

a) Processes

b) CPU-I/O burst cycle

Process execution is a cycle of cpu execution and I/O wait. Processes alternate back and forth between these two states. Process execution begins with a cpu burst. It is followed by an]O burst, which is followed by another cpu burst, then another I/O hurst, and so on.

c) Process state

d) Process control block

2) SCHEDULING QUEUES

a) Ready queue

The Processes which are ready and waiting to execute are kept on a list called the ready queue. This list is generally a linked list. A ready queue header will contain pointers to the first and last PCBs in the list. Each PCB has a pointer field which points to the next process in the ready queue.

b) Device queue

The list of processes waiting for a particular I/O device is called a device queue.

c) Processing of queues

A process enters the system from the outside world and is put in the ready queue. It waits in the ready queue until it is selected for the cpu. After running on the cpu, it waits for an I/O operation by moving to an I/O queue. Eventually, it is served by the I/O device and returns to the ready queue. A process continues this cpu-I/O cycle until it finishes; then it exits from the system.

3) SCHEDULERS

a) Long-term scheduler

* **Determines which jobs are admitted to the system for processing.**
* **Controls degree of multiprogramming (the number of processes in memory)**
* **Selects a good job mix of I/O-bounded and cpu-bounded jobs.**

b) Short-term scheduler (cpu scheduler)

* **Selects (quite o[ten) from among the jobs in memory which are ready to execute and allocates the cpu to one of them.**

c) Dispatcher

* **The module that actually gives control of the cpu to the process selected by the short-term scheduler.**

3. SCHEDULING ALGORITHMS

1) PERFORMANCE CRITERIA

a) CPU utilization: range from 0 percent to 100 percent.
b) Throughput: number of jobs which are completed per time unit.
c) Turnaround time: the interval from the time of submission to the time of completion.
d) Waiting time: the amount of time that a job spends waiting in the ready queue.
e) Response time: the time from the submission of a request until the first response is produced (the amount of time it take to start responding. not the time that it takes to output that response),

Note: A system with reasonable and predictable response time may be considered better than a system which is faster on the average, but highly variable.

2) FIRST- COME- FIRST- SERVED (FCFs)

The code for FCFS scheduling is simple to write and understand. The Performance of FCFS, however, is often quite poor.

3) SHORTEST- JOB- FIRST

Shortest- Job- First (SJF) is probably optimal, in that it gives the minimum average waiting time for a given set of jobs.

The real difficulty with SJF is knowing the length of the next cpu request. One approach is to try to approximate SJF scheduling.

4) PRIORITY

SJF is a special case of the general priority scheduling algorithm. A SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next cpu burst (tau): p = 1/tau

Priority can be defined either internally or externally.

Major problem with priority scheduling algorithms is indefinite blocking or starvation. Aging for low- priority jobs may be solution to the this problem.

Each job is assigned a priority.
FIFO within each priority level.
Select highest priority job over lower ones.

Priority may based on:

- **Cost to user.**
- **Importance of user**
- **Aging**
- **Percentage of CPU time used in last X hours**

Indefinite blocking or starvation a problem

5) PREEMPTIVE ALGORITHMS

FCFS, SJF, and priority algorithms, as described so far, are non- preemptive scheduling algorithms. (FCFS: Non- preemptive, SJF and Priority: either preemptive or non- preemptive). Preemptive Shortest- Job- First is sometimes called Shortest- Remaining_Time - First.

6) ROUND- ROBIN (RR)

The RR scheduling algorithm (preemptive) is designed especially for time- sharing systems. A small unit of time. called time quantum (generally from 10 to 100

milliseconds) or slice, is defined.

Example)

Time quantum is 4.

Job	Burst time
1	24
2	3
3	3

Average turn around time is (7 + 10 + 30)/3 = 16

In the RR scheduling algorithm, no process is allocated the cpu for more than one time quantum in a row. If its cpu bursts exceeds a titre quantum, it is preempted and put back in the ready queue.

If time quantum is very large (in mite), RR is the same as FCFS. If the time quantum is very small, RR is called processor sharing.

RR has context switch problems. A suggested rule-of-thumb is that 80 percent of the cpu bursts should be shorter than the time quantum.

7) MULTI-LEVEL QUEUES

A multi*queue scheduling algorithm partitions the ready queue into separate queues. Jobs are permanently assigned to one queue, generally based upon some property of the job, such as memory size or job type. Each queue has its own scheduling algorithm. In addition, there must be scheduling between the queues. Jobs do not move between queues.

Processes assigned to one queue permanently
Scheduling between queues: Fixed priority

Example:

- System processes
- Interactive programs
- Interpretive editing
- Batch processes
- Student process

8) MULTI-LEVEL FEEDBACK QUEUES

Allow a job to move between queues. The idea is to separate out jobs with different cpu-burst characteristics. If a job uses too much cpu time, it will be moved to a lower priority queue.

Multi- Level Queue with priorities
Processes move between queue
Each queue represents jobs with similar CPU usage
Jobs in a given queue are executed with a given timeslice

9) REAL- TIME SCHEDULING

periodic scheduler
Demand4riven schedulers
Deadline Scheduler
Aperiodic Scheduler
Sporadic Scheduler

4. ALGORITHM EVALUATION

1) ANALYTIC EVALUATION
- **a) Deterministic modeling**
- **b) Queueing models**

2) SIMULATIONS

3) IMPLEMENTATION

II. DEADLOCK

1. DEFINITIONS

A process is deadlocked if it is waiting for an event that will never occur. Typically, but not necessarily. more than one process will be involved together in a deadlock.

A process is indefinitely postponed if it is delayed repeatedly over a long period of time while the attention of the system is given to other processes i. e., logically the process may proceed but the system never gives in the CPU.

2. RESOURCES
Resource is a commodity needed by a process. Resources work by order of a process or another resource; processes do not.

* Resources can be either:
  - Serially reusable: CPU, memory, disk space. I/O devices, files.
    acquire --> use --> release
  - Consumable: produced by a process, needed by a process; massage, buffers of information. interrupts.
    create ---> acquire --> use

  Resource ceases to exist after it has been used, so it is not released.

* Resources can also be either.
  - Preemptible: CPU, central memory
  - Nonpreemptible: Tape drives

*Resource can be either:
  - Shared among several processes
  - dedicated exclusively to single process

3. CONDITIONS FOR DEADLOCK

The following 4 conditions are both necessary and sufficient for deadlock:

1> Mutual exclusion: Processes claim exclusive control of the resources they require
2> Wait-for condition: Process hold resources already allocated Io them while waiting for additional resources
3> No preemption condition: Resources cannot be removed from the process holding them until used to completion.
4> Circular wait condition: A circular chain of processes exist in which process holds one or more resources that are requested by the next process in the chain.

4. DEADLOCK ISSUES

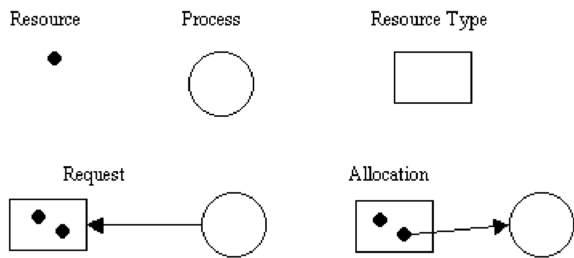
* Prevention: Design a system in such a way that deadlocks cannot occur, at least with respect to serially reusable resources.

* Avoidance: Impose less stringent conditions than for prevention, allowing the possibility of deadlock, but sidestepping it as it approaches.

* Detection: In .a system that allows the possibility of deadlock, determine if deadlock has occurred, and which processes and resources axe involved.

* Recovery: After a deadlock has been detected. clear the problem. allowing the deadlocked processes to complete and the resources to be reused. Usually involves destroying the affected processes and starting them over.

5. RESOURCE ALLOCATION GRAPH



6. DEADLOCK PREVENTION: HARVENDER'S ALGORITHMS

If any one of the condition for deadlock (with reusable resources) is defined, deadlock is impossible.

1> Mutual exclusion: We don't want to deny this; exclusive use of resources is an important feature.

2> Wait- for condition: Force each process to request all required resources at once. It cannot proceed until all resources have been acquired.

3> No preemption condition: If a process holding some reusable resources makes a further request which is denied, and it wishes to wait for the new resources to become available,
it must release all resources currently held and, if necessary, request them again along with the new resources. Thus, resources am removed from a process holding them

4> Circular wait condition: All resource types are numbered. Processes must request resources in numerical order;, if a resource of type N is held, the only resources which can be requested must be of types > N.

7. DEADLOCK AVOIDANCE

* Set of resource
* Set of customers
* Banker
  1> Each customer tells banker the maximum number of resources it needs
  2> Customer borrows resources from banker
  3> Customer returns resources to banker
  4> Customer eventually pay back loan

* Bankers only lends resources if the system will be in a safe state after the loan
* Safe state: there is lending sequence such that all customers can take out a loan
* Unsafe state: there is a possibility of deadlock

8. BANKER'S ALGORITHM

Step 1. If Request* <= Need¡, go to step 2. Otherwise, raise an error condition.
Step 2. If Request, <= available, go to step 3. Otherwise, Process i (P¡) must wait.
Step 3. The system pretends to have allocated the requested resources to process P¡ by modifying the state as follows:

Available := Available - $Request_i$;
Allocation¡ := Allocation + $Request_i$;
Need := Need, - Request;

* Safety Algorithm:

Step 1. Let Work and Finish be vectors of length m and n. respectively. .
Initialize Work := Available and Finish[i] := False for i = 1, 2, ..., n.
Step 2. Find an i such that both
a. Finish[i] = False
b. $Necd_i$ <= Work
If no such i exists, go to step 4.
Step 3. Work := Work + Allocation,
Finish[i] := True
go to step 2.
Step 4. If Finish[i] = True for all i, then the system is in a safe state.

* Example 1.

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

1) Is the system in a safe state?
Answer: Yes, the sequence <P1. P3, P4, P2, P0> satisfies the safety requirement.

2) If a request. from process P1 arrives for (1, 0, 2) can the request be granted immediately?

Answer: Yes. Since

a. (1, 0, 2) <= Available = (3, 3, 2)
b. (1, 0, 2) <= Max, = (3, 2, 2)
c. The new system state after the allocation is made is

	Allocation			Max			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	7	4	3	2	3	0
P1	3	0	2	3	2	2	0	2	0			
P2	3	0	2	9	0	2	6	0	0			
P3	2	1	1	2	2	2	0	1	1			
P4	0	0	2	4	3	3	4	3	1			

d. The sequence <P1, P3, P4, P0, P2> satisfies the safety requirement

<u>* Example 2:</u>

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

1) Is the system in a safe state?

<u>Answer:</u> Yes, the sequence <P0. P2, PI, P3. P4> satisfies the safety requirement.

2) If a request from process P1 arrives for t0, 4, 2, 01 can the request be granted immediately?

<u>Answer:</u> Yes. Since

a. (0, 4, 2, 0) <= Available = (1, 5, 2, 0)
b. (0, 4, 2, 0) <= Max, = (1. 7, 5, 0)
c. The new system state after the allocation is made is

	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	0	0	0	0	1	1	0	0
P1	1	4	2	0	1	7	5	0	0	3	3	0				
P2	1	3	5	4	2	3	5	6	1	0	0	2				
P3	0	6	3	2	0	6	5	2	0	0	2	0				
P4	0	0	1	4	0	6	5	6	0	6	4	2				

d. The sequence <P0, PI, P2, P3, P4> satisfies the safety requirement

* **Banker Solution Issues**

  - **Process may not terminate**
  - **Process may request more than claim**
  - **A process may suffer indefinite postponement**
    a) **Solution is to check for aged process**
    b) **Select an allocation sequence that includes aged process**
    c) **Only select requests that follow that sequence until aged process executes**

9. DEADLOCK DETECTION

* **Check to see if a deadlock has occurred**
* **Multiple resources per type**
  - **Run variant of previous algorithm to see if processes can finish**
  - **Order(n^2) operations, n nodes in graph**
* **How often should the detection algorithm be run?**
* **How many processes will be affected by deadlock?**

10. RECOVERY FROM DEADLOCK

* **Kill deadlocked processes and release resources**
* **Kill one deadlocked process at a time and release its resources**
* **Rollback all or one of the processes Io a checkpoint that occurred before they requested any resources**

II. REAL- TIME SCHEDULING

1. WHAT IS REAL- TIME SYSTEMS

1) DEFINITION OF REAL_TIME COMPUTING

The correctness of the system depends not only on the logical result of the but also on the time at which the results are produced.

2) APPLICATION AREAS

a) Nuclear power plant control
b) Factory automation
c) Air traffic control
d) Telecommunication systems
e) Military defense systems

3) CHARACTERISTICS OF REAL- TIME SYSTEMS

a) Largeness and complexity
b) Reliability and safety
c) Concurrent Control
d) Real- Time control

2. ISSUES IN REAL- TIME SYSTEMS

l) REAL- TIME ISSUES

a) How to represent time
b) How to control time constraints

2) DESIGN AND ANALYSIS ISSUES

a) Finite State Machine
b) Floid Logic
c) Stochastic Petri Net (SPN)
d) Temporal Logic
e) Statecharts

3) LANGUAGE ISSUES

4) VERIFICATION ISSUES

3. REAL-TIME OPERATING SYSTEMS

1) CONVENTIONAL REAL-TIME OPERATING SYSTEM KERNEL

a) Fast context switching
b) Small size
c) Fast interrupt latency
d) Fast response to external interrupts
e) No virtual memory
f) Maintain real-time clock

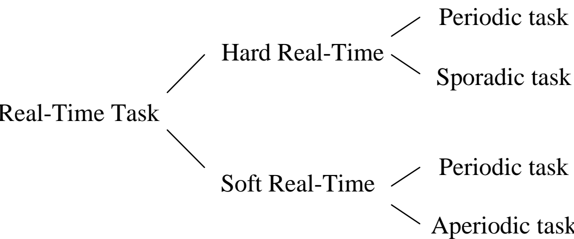
2) ISSUES IN DISTRIBUTED REAL-TIME SYSTEMS

a) Inflexible Real-Time Scheduler (Cyclic Executive Model)
b) Lack of Real-Time Communication Protocols
c) Priority inversion

4. REAL-TIME SCHEDULING

1) REAL-TIME COMPUTATION MODEL

a) Classification of task



b) Classification Categories

* **Number of processor**
* **Task duration (equal/unequal)**
* **Precedence relationship (independent/dependent)**
* **Task interpretability (preemptive/nonpreemptive)**
* **Job periodicity (periodic/aperiodic)**
* **Presence/absence of deadline (hard/soft-real time)**
* **Resource-limited/not limited**
* **Homogeneous/Heterogeneous processors**

3) MEASURE OF MERIT IN REAL-TIME SYSTEMS

a) Predictability
b) High degree of schedulability
c) Stability under transient overload

4) SCHEDULING DIFFERENCES BETWEEN OR AND REAL-TIME OS

a) Operation Research

* Scheduling performed statically
* Off-Line
* Assume all the jobs are scheduled and available at t--0
* No new job appeal- during processing

b) Real-Time Operating Systems

* Scheduling decisions is made dynamically
* On-line
* Irregular arrival/departure of processes
* Stochastic processing time

5. SCHEDULING PERIODIC TASKS

1) REQUIREMENTS: Continuously meeting periodic and sporadic timing constraints in real-time

2) TRADITIONAL REAL-TIME SCHEDULING

Cyclical Executive:
- Ad hoc manner
- Hard to modification
- Time consuming (fitting code into time slot)

3) RATE MONOTONIC SCHEDULING ALGORITHM

a) Idea: Rate Monotonic priority assignment (assign priority tasks according to their request rates (1/Ti where Ti is period of task i)

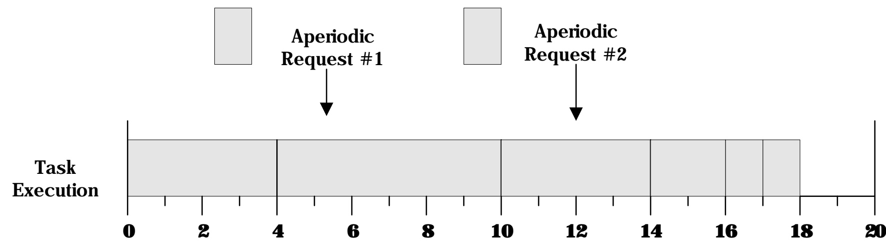
b) Theorem: A set of n independent periodic tasks scheduled by the rate monotonic algorithm will always meet its deadlines if;

$$\sum CTi \leqq n(2^{1/\partial} - 1)$$

where Ci and Ti are execution time: and period of task i respectively

6. SCHEDULING APERIODIC TASKS

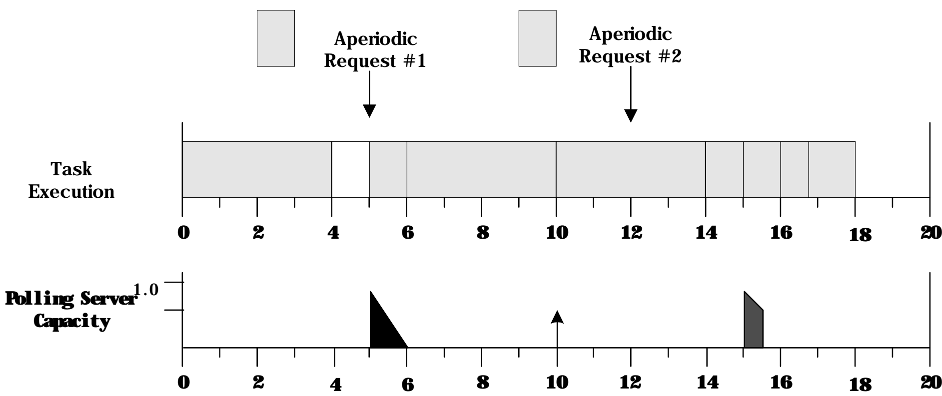
1) BACKGROUND APERIODIC SERVICE



Periodic Tasks:

	Execution Time	Period	Priority
Task A	4	10	High
Task B	8	20	Low

2) POLLING SERVICE



Polling Server: Executio Time = 1, Period = 5

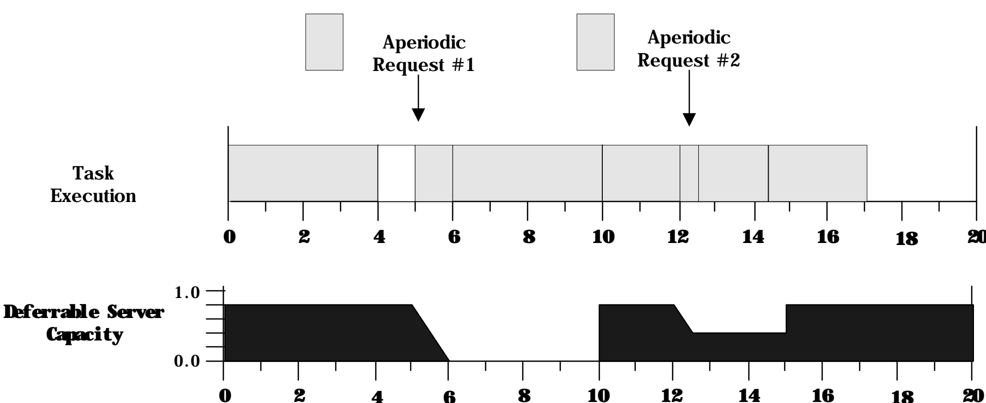
Periodic Tasks:

	Execution Time	Period	Priority
Task A	4	10	High
Task B	8	20	Low

3) DEFERRABLE SERVER

a) Idea: Provide a mechanism for preserving the resource bandwith allocated for aperiodic service.

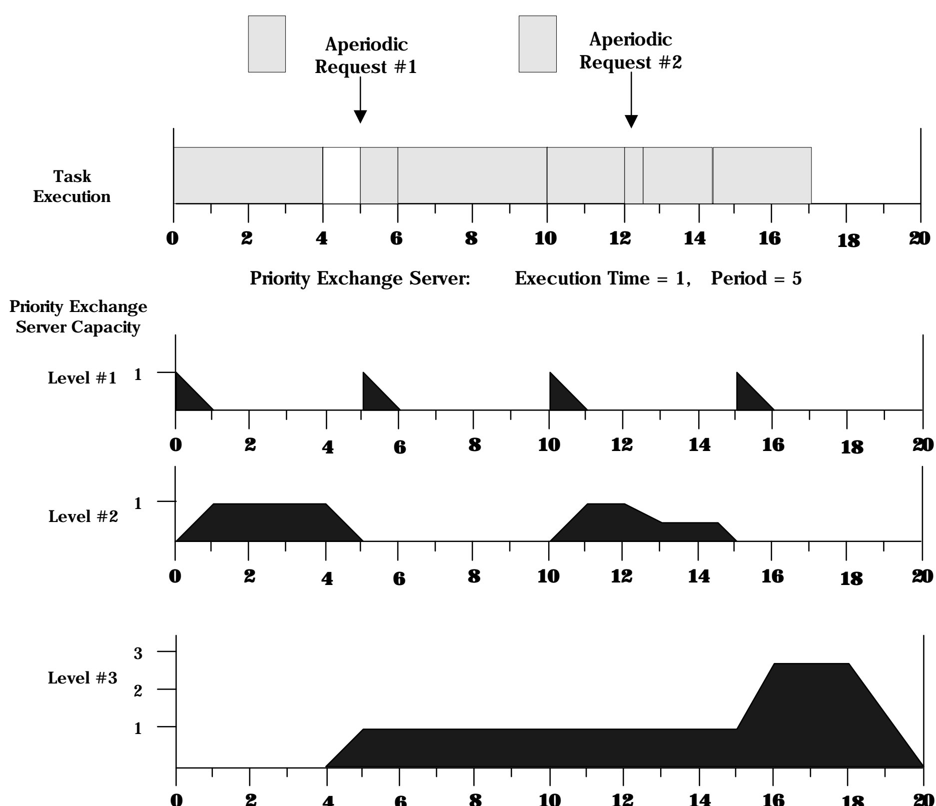
b) Example



4) PRIORITY EXCHANGE

a) Idea: Preserves its high priority execution time by exchanging it for the execution of lower priority periodic task

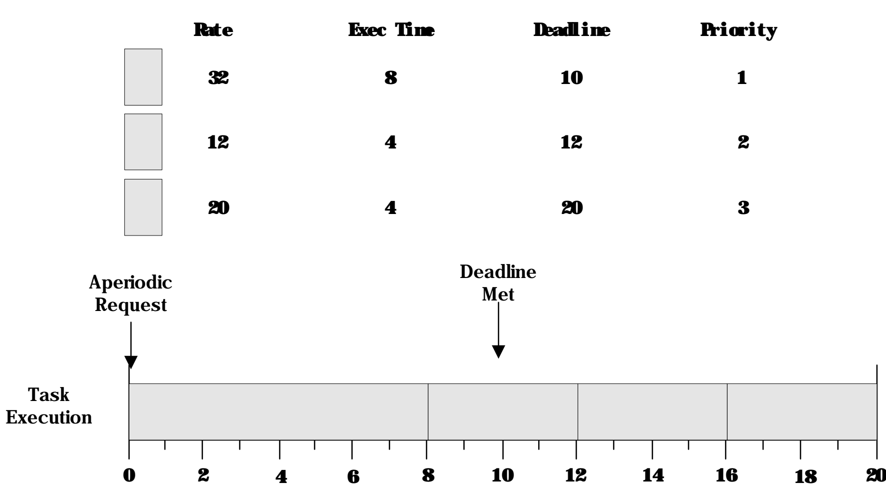
b) Example



6. SCHEDULING SPORADIC TASKS

a) Idea: Deadline monotonic priority assignment (instead of rate monotonic assignment).

b) Example



Deadline Monotonic Priority Assignment:

Rate	Exec Time	Deadline	Priority
32	8	10	1
12	4	12	2
20	4	20	3