UNIT – II

CPU Scheduling - Scheduling Criteria, Scheduling Algorithms, Multiple -Processor Scheduling.
Deadlocks - System Model, Deadlocks Characterization, Methods for Handling Deadlocks, Deadlock Prevention, Deadlock Avoidance, Deadlock Detection, and Recovery from Deadlock

PROCESS SCHEDULING

CPU is always busy in **Multiprogramming**. Because CPU switches from one job to another job.
But in **Simple computers** CPU sit idle until the I/O request granted.
Scheduling is an important OS function. All resources are scheduled before use (CPU, memory, devices…..) Process scheduling is an essential part of Multiprogramming operating systems.
Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Scheduling Objectives
Maximize throughput.
Maximize number of users receiving acceptable response times. Be predictable.

Balance resource use.
Avoid indefinite postponement. Enforce Priorities.
Give preference to processes holding key resources

SCHEDULING QUEUES: people live in rooms. Process are present in rooms knows as queues. There are 3types

1. **Job queue** : when processes enter the system, they are put into a **job queue** , which consists all processes in the system. Processes in the job queue reside on mass storage and await the allocation of main memory.
2. **Ready queue** : if a process is present in main memory and is ready to be allocated to cpu for execution, is kept in **ready queue.**
3. **device queue** : if a process is present in waiting state (or) waiting for an i/o event to complete is said to be in device queue.(or)The processes waiting for a particular I/O device is called device queue.

Schedulers: There are 3 schedulers
1. Long term scheduler.
2. Medium term scheduler
3. Short term scheduler.

Scheduler duties:
Maintains the queue.
Select the process from queues assign to CPU.
processes holds one of these CDRW drives. If each process now requests another

Types of schedulers

1. **Long term scheduler:**
   Select the jobs from the job pool and loaded these jobs into main memory (ready queue).Long term scheduler is also called job scheduler.
2. **Short term scheduler:**
   Select the process from ready queue, and allocates it to the CPU. If a process requires an I/O device, which is not present available then process enters device queue. short term scheduler maintains ready queue, device queue.
   Also called as cpu scheduler.
3. **Medium term scheduler** : if process request an I/O device in the middle of the execution, then the process removed from the main memory and loaded into the waiting queue. When the I/O operation completed, then the job moved from waiting queue to ready queue. These two operations performed by medium term scheduler.

Context Switch : Assume, main memory contains more than one process. If cpu is executing a process, if time expires or if a high priority process enters into main memory, then the scheduler saves information about current process in the PCB and switches to execute the another process. The concept of moving CPU by scheduler from one process to other process is known as context switch.

Non-Preemptive Scheduling : **CPU** is assigned to one process, CPU do not release until the competition of that process. The CPU will assign to some other process only after the previous process has finished.

Preemptive scheduling: here CPU can release the processes even in the middle of the execution. CPU received a signal from process p2. OS compares the priorities of p1, p2. If p1>p2, CPU continues the execution of p1. If p1<p2 CPU preempt p1 and assigned to p2.

Dispatcher: The main job of dispatcher is switching the cpu from one process to another process. Dispatcher connects the cpu to the process selected by the short term scheduler.

Dispatcher latency : The time it takes by the dispatcher to stoPone process and start another process is known as dispatcher latency. If the dispatcher latency is increasing, then the degree of multiprogramming decreases.

SCHEDULING CRITERIA:

1. **Throughput** : how many jobs are completed by the cpu with in atime period.
2. **Turn around time:**The time interval between the submission of the process and time of the completion is turn around time.

TAT = Waiting time in ready queue + executing time + waiting time in waiting queue for I/O.

3. **Waiting time** : The time spent by the process to wait for cpu to be allocated.
4. **Response time** : Time duration between the submission and first response.
5. **Cpu Utilization** : CPU is costly device, it must be kept as busy as possible. Eg: CPU efficiency is

90% means it is busy for 90 units, 10 units idle.

CPU SCHEDULINGALGORITHMS:

First come First served scheduling: (FCFS):

The process that request the CPU first is holds the cpu first.

If a process request the cpu then it is loaded into the ready queue, connect CPU to that process.

Consider the following set of processes that arrive at time 0, the length of the cpu burst time processes holds one of these CDRW drives. If each process now requests another given in milli seconds.

burst time is the time, required the cpu to execute that job, it is in milli seconds.

Process	Burst time(milliseconds)
P1	5
P2	24
P3	16
P4	10
P5	3

Average turnaround time:

Turnaround time= waiting time + burst time

Turn around time for p1= 0+5=5. Turnaround time for 2=5+24=29 Turnaround time for p3=29+16=45 Turnaround time for p4=45+10=55 Turnaround time for p5= 55+3=58

Average turnaround time= (5+29++45+55+58/5) = 187/5 =37.5 milliseconds

Average waiting time:

Waiting time= starting time- arrival time

Waiting time for p1=0

Waiting time for p2=5-0=5 Waiting time for p3=29-0=29 Waiting time for p4=45-0=45 Waiting time for p5=55-0=55

Average waiting time= 0+5+29+45+55/5 = 125/5 = 25 ms.

Average Response Time :

Formula: First Response - Arrival Time Response Time for P1 =0 Response Time for P2 => 5-0 = 5 Response Time for P3 => 29-0 = 29 Response Time for P4 => 45-0 = 45 Response Time for P5 => 55-0 = 55

processes holds one of these CDRW drives. If each process now requests another

Average Response Time => (0+5+29+45+55)/5 =>25ms

1)

2) **First Come First Serve:**

It is Non Primitive Scheduling Algorithm.

PROCESS	BURST TIME	ARRIVAL TIME
P1	3	0
P2	6	2
P3	4	4
P4	5	6
P5	2	8

Process arrived in the order P1, P2, P3, P4, P5. P1 arrived at 0 ms.

P2 arrived at 2 ms. P3 arrived at 4 ms. P4 arrived at 6 ms. P5 arrived at 8 ms.

Average Turn Around Time

Formula : Turnaround Time = waiting time + burst time Turn Around Time for P1 => 0+3= 3

Turn Around Time for P2 => 1+6 = 7 Turn Around Time for P3 => 5+4 = 9 Turn Around Time for P4 => 7+ 5= 12 Turn Around Time for P5 => 2+ 10=12

Average Turn Around Time => (3+7+9+12+12)/5 =>43/5 = 8.50 ms.

Average Response Time :

Formula : Response Time = First Response - Arrival Time Response Time of P1 = 0

Response Time of P2 => 3-2 = 1

Response Time of P3 => 9-4 = 5 Response Time of P4 => 13-6 = 7

processes holds one of these CDRW drives. If each process now requests another

Response Time of P5 => 18-8 =10

Average Response Time => (0+1+5+7+10)/5 => 23/5 = 4.6 ms

Advantages: Easy to Implement, Simple.

Disadvantage: Average waiting time is very high.

Shortest Job First Scheduling (SJF):

Which process having the smallest CPU burst time, CPU is assigned to that process . If two process having the same CPU burst time, FCFS is used.

PROCESS	CPU BURST TIME
P1	5
P2	24
P3	16
P4	10
P5	3

P5 having the least CPU burst time (3ms). CPU assigned to that (P5). After completion of P5 short term scheduler search for nest (P1).....

Average Waiting Time :

Formula = Staring Time - Arrival Time waiting Time for P1 => 3-0 = 3 waiting Time for P2 => 34-0 = 34 waiting Time for P3 => 18-0 = 18 waiting Time for P4 =>8-0=8 waiting time for P5=0

Average waiting time => (3+34+18+8+0)/5 => 63/5 =12.6 ms

Average Turn Around Time : Formula = waiting Time + burst Time

processes holds one of these CDRW drives. If each process now requests another

Turn Around Time for P1 => 3+5 =8 Turn Around for P2 => 34+24 =58 Turn Around for P3 => 18+16 = 34

Turn Around Time for P4 => 8+10 =18 Turn Around Time for P5 => 0+3 = 3

Average Turnaround time => (8+58+34+18+3)/5 => 121/5 = 24.2 ms

Average Response Time :

Formula: First Response - Arrival Time First Response time for P1 =>3-0 = 3

First Response time for P2 => 34-0 = 34 First Response time for P3 => 18-0 = 18 First Response time for P4 => 8-0 = 8

First Response time for P5 = 0

Average Response Time => (3+34+18+8+0)/5 => 63/5 = 12.6 ms

SJF is Non primitive scheduling algorithm

Advantage : **Least average waiting time**

Least average turnaround time Least average response time

Average waiting time (FCFS) = 25 ms
Average waiting time (SJF) = 12.6 ms 50% time saved in SJF.

Disadvantages:

Knowing the length of the next CPU burst time is difficult.
Aging (Big Jobs are waiting for long time for CPU)

3). Shortest Remaining Time First (SRTF);

This is primitive scheduling algorithm.
Short term scheduler always chooses the process that has term shortest remaining time.

When a new process joins the ready queue, short term scheduler compare the remaining time of executing process and new process. processes holds one of these CDRW drives. If each process now requests another

If the new process has the least CPU burst time, the schedulers selects that job and connect to CPU. Otherwise continue the old process.

PROCESS	BURST TIME	ARRIVAL TIME
P1	3	0
P2	6	2
P3	4	4
P4	5	6
P5	2	8

P1 arrives at time 0, P1 executing First , P2 arrives at time 2. Compare P1 remaining time and P2 (3-2 = 1) and 6. So, continue P1 after P1, executing P2, at time 4, P3 arrives, compare P2 remaining time (6-1=5

) and 4 (4<5) .So, executing P3 at time 6, P4 arrives. Compare P3 remaining time and P4 (4-2=2) and 5 (2<5). So, continue P3 , after P3, ready queue consisting P5 is the least out of three. So execute P5, next P2, P4.

FORMULA : Finish time - Arrival Time Finish Time for P1 => 3-0 = 3 Finish Time for P2 => 15-2 = 13 Finish Time for P3 => 8-4 =4 Finish Time for P4 => 20-6 = 14 Finish Time for P5 => 10-8 = 2

Average Turn around time => 36/5 = 7.2 ms.

4)ROUND ROBIN SCHEDULING ALGORITHM :

It is designed especially for time sharing systems. Here CPU switches between the processes. When the time quantum expired, the CPU switched to anotheRjob. A small unit of time, called a time quantum or time slice. A time quantum is generally from 10 to 100 ms. The time quantum is

generally depending on OS. Here ready queue is a circular queue. CPU scheduler picks the first process from ready queue, sets timer to interrupt after one time quantum and dispatches the process.

PROCESS	BURST TIME
P1	30
P2	6
P3	8

AVERAGE WAITING TIME :

Waiting time for P1 => 0+(15-5)+(24-20) => 0+10+4 = 14
Waiting time for P2 => 5+(20-10) => 5+10 = 15
processes holds one of these CDRW drives. If each process now requests another
Waiting time for P3 => 10+(21-15) => 10+6 = 16 Average waiting time => (14+15+16)/3 = 15 ms.

AVERAGE TURN AROUND TIME :

FORMULA : Turn around time = waiting time + burst Time Turn around time for P1 => 14+30 =44
Turn around time for P2 => 15+6 = 21 Turn around time for P3 => 16+8 = 24
Average turn around time => (44+21+24)/3 = 29.66 ms

5) PRIORITY SCHEDULING :

PROCESS	BURST TIME	PRIORITY
P1	6	2
P2	12	4
P3	1	5
P4	3	1
P5	4	3

P4 has the highest priority. Allocate the CPU to process P4 first next P1, P5, P2, P3.

AVERAGE WAITING TIME :

Waiting time for P1 => 3-0 =3 Waiting time for P2 => 13-0 = 13 Waiting time for P3 => 25-0 = 25
Waiting time for P4 => 0
Waiting time for P5 => 9-0 =9
Average waiting time => (3+13+25+0+9)/5 = 10 ms

AVERAGE TURN AROUND TIME :

Turn around time for P1 =>3+6 = 9

Turn around time for P2 => 13+12= 25

Turn around time for P3 => 25+1 = 26

processes holds one of these CDRW drives. If each process now requests another

Turnaround time for P4 => 0+3= 3

Turnaround time for P5 => 9+4 = 13

Average Turnaround time => (9+25+26+3+13)/5 = 15.2 ms

Disadvantage: Starvation

Starvation means only high priority process are executing, but low priorityprocess are waiting for the CPU for the longest period of the time.

Multiple – processor scheduling:

When multiple processes are available, then the scheduling gets more complicated, because there is more than one CPU which must be kept busy and in effective use at all times.

Load sharing resolves around balancing the load between multiple processors. Multi processor systems may be heterogeneous (It contains different kinds of CPU's) (or) Homogeneous(all the same kind of CPU).

Approaches to multiple-processor schedulinga)Asymmetric multiprocessing

One processor is the master, controlling all activities and running all kernel code, while the other runs only user code.

b)Symmetric multiprocessing:

Each processor schedules its own job. Each processor may have its own private queue of readyprocesses .

Processor Affinity

Successive memory accesses by the process are often satisfied in cache memory. what happens if the process migrates to another processor. the contents of cache memory must be invalidated for the first processor, cache for the second processor must be repopulated. Most Symmetric multi processor systems try to avoid migration of processes from one processor to another processor, keePa process running on the same processor. This is called processor affinity.

Soft affinity:

Soft affinity occurs when the system attempts to keePprocesses on the same processor but makes no guarantees.

a) Hard affinity:

Process specifies that it is not to be moved between processors.

2)Load balancing:

One processor wont be sitting idle while another is overloaded. Balancing can be achived through

push migration or pull migration.

Push migration:

Push migration involves a separate process that runs periodically(e.g every 200 ms) and moves processes from heavily loaded processors onto less loaded processors.

Pull migration:

processes holds one of these CDRW drives. If each process now requests another

Pull migration involves idle processors taking processes from the ready queues of the other processors.

Deadlocks- System Model, Deadlocks Characterization, Methods for Handling Deadlocks, Deadlock Prevention, Deadlock Avoidance, Deadlock Detection, and Recovery from Deadlock

DEADLOCKS

System model:

A system consists of a finite number of resources to be distributed among a number of competing processes. The resources are partitioned into several types, each consisting of some number of identical instances. Memory space, CPU cycles, files, I/O devices are examples of resource types. If a system has 2 CPUs, then the resource type CPU has 2 instances.

A process must request a resource before using it and must release the resource after using it. A process may request as many resources as it requires to carry out its task. The number of resources as it requires to carry out its task. The number of resources requested may not exceed the total number of resources available in the system. A process cannot request 3 printers if the system has only two.

A process may utilize a resource in the following sequence:

- REQUEST: The process requests the resource. If the request cannot be granted immediately (if the resource is being used by another process), then there questing process must wait until it can acquire there source.
- USE: The process can operate on the resource .if the resource is a printer, the process can print on the printer.
- RELEASE: The process release there source.

For each use of a kernel managed by a process the operating system checks that the process has requested and has been allocated the resource. A system table records whether each resource is free (or) allocated. For each resource that is allocated, the table also records the process to which it is allocated. If a process requests a resource that is currently allocated to another process, it can be added to a queue of processes waiting for this resource.

To illustrate a deadlocked state, consider a system with 3 CDRW drives. Each of 3

processes holds one of these CDRW drives. If each process now requests another drive, the 3 processes will be in a deadlocked state. Each is waiting for the event "CDRW is

released" which can be caused only by one of the other waiting processes. This example illustrates a deadlock involving the same resource type.

Deadlocks may also involve different resource types. Consider a system with one printer and one DVD drive. The process Pi is holding the DVD and process Pj is holding the printer. If Pi requests the printer and Pj requests the DVD drive, a deadlock occurs.

DEADLOCK CHARACTERIZATION:

In a deadlock, processes never finish executing, and system resources are tied up, preventing otheRjobs from starting.

NECESSARY CONDITIONS:

A deadlock situation can arise if the following 4 conditions hold simultaneously in a system:

1. MUTUAL EXCLUSION: Only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until there source has been released.
2. HOLD AND WAIT: A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
3. NO PREEMPTION: Resources cannot be preempted. A resource can be released only voluntarily by the process holding it, after that process has completed its task.
4. CIRCULAR WAIT: A set {P0,P1,…..Pn} of waiting processes must exist such that P0 is waiting for resource held by P1, P1 is waiting for a resource held by P2,……,Pn-1 is waiting for a resource held by Pn and Pn is waiting for a resource held byP0.

RESOURCE ALLOCATION GRAPH

Deadlocks can be described more precisely in terms of a directed graph called a system resource allocation graph. This graph consists of a set of vertices V and a set of edges E. the set of verticesV is partitioned into 2 different types of nodes:

P= {P1 , P2….Pn}, the set consisting of all the active processes in the system. R= {R1, R2….R m}, the set consisting of all resource types in the system.

A directed edge from process Pi to resource type Rj is denoted by Pi ->Rj. It signifies that process Pi has requested an instance of resource type Rj and is currently waiting for that resource.

A directed edge from resource type Rj to process Pi is denoted by Rj ->Pi, it signifies that an instance of resource type Rj has been allocated to process Pi.

A directed edge Pi ->Rj is called a requested edge. A directed edge Rj->Pi is called an assignmentedge.

We represent each process Pi as a circle, each resource type Rj as a rectangle. Since resource

type Rj may have more than one instance. We represent each such instance as a dot within the rectangle. A request edge points to only the rectangle Rj. Assignment edge must alsodesignate one of the dots in the rectangle.

When process Pi requests an instance of resource type R j, a request edge is inserted in the resource allocation graph. When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge. When the process no longer needs access to the resource, it releases the resource, as a result, the assignment edge is deleted. The sets P, R, E:

P= {P1 , P2, P3}
R= {R1, R 2, R 3, R4}
E= {P1 ->R1, P2 ->R 3, R1 ->P2, R 2 ->P2, R 2 ->P1 , R 3 ->P

One instance of resource type R1
Two instances of resource type R2 One instance of resource type R3 Three instances of resource type R4

PROCESS STATES:

Process P1 is holding an instance of resource type R2 and is waiting for an instance of resourcetype R1.

Process P2 is holding an instance of R1 and an instance of R2 and is waiting for instance of R 3.Process P3 is holding an instance of R3.

If the graph contains no cycles, then no process in the system is deadlocked. Ifthe graph does contain a cycle, then a deadlock may exist.

Suppose that process P3 requests an instance of resource type R2. Since no resource instance iscurrently available, a request edge P3 ->R 2 is added to the graph.

2 cycles:
P1 ->R1 ->P2 ->R 3 ->P3 ->R 2 ->P1 P2 ->R 3 ->P3 ->R 2 ->P2

Processes P1 , P2, P3 are deadlocked. Process P2 is waiting for the resource R3, which is held byprocess P3.process P3 is waiting for either process P1 (or) P2 to release resource R2. In addition,process P1 is waiting for process P2 to release resource R1.

We also have a cycle: P1 ->R1 ->P3 ->R 2 ->P1
However there is no deadlock. Process P4 may release its instance of resource type R2.
That resource can then be allocated to P3, breaking the cycle.

DEADLOCK PREVENTION

For a deadlock to occur, each of the 4 necessary conditions must held. By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.

Mutual Exclusion– not required for sharable resources; must hold for non sharable resources

Hold and Wait– must guarantee that whenever a process requests a resource, it does not hold any other resources

- Require process to request and be allocated all its resources before it begins
- Low resource utilization
- Starvation possible
- execution, or allow process to request resources only when the process has none

o Preemption–

- If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
- Preempted resources are added to the list of resources for which the process is waiting
- Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting

Circular Wait – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

Deadlock Avoidance Requires that the system has some additional *a priori* information available

- Simplest and most useful model requires that each process declare the
- *maximum number* of resources of each type that it may need
- The deadlock-avoidance algorithm dynamically examines the resource- allocation state to ensure that there can never be a circular-wait condition
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes .

Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state
- System is in **safe state**if there exists a sequence < *P1, P2, ..., Pn* > of ALL the processes in the systems such that for each Pi, the resources that Pi can still request can be satisfied
- by currently available resources + resources held by all the *Pj* , with *j <I* That is:
- If Pi resource needs are not immediately available, then *Pi* can wait until all
- *Pj* have finished
  - ✓ When *Pj* is finished, *Pi* can obtain needed resources, execute,return allocated resources, and terminate
  - ✓ When *Pi* terminates, *Pi* +1 can obtain its needed resources, and so on If a system is in safe

state no deadlocks

- ✓ If a system is in unsafe state possibility of deadlock Avoidance ensure that a system will never enter an unsafe state

Avoidance algorithms

Single instance of a resource type

Use a resource-allocation graph Multiple instances of a resource type

Use the banker's algorithm

Resource-Allocation Graph Scheme

Claim edge *Pi ÆRj* indicated that process *Pj* may request resource *Rj* ;represented by a dashed line

Claim edge converts to request edge when a process requests a resource Request edge converted to an assignment edge when the resource is allocatedto the process When a resource is released by a process, assignment edge reconverts to a claim edge Resources

must be claimed *a priori* in the system

Unsafe State In Resource-Allocation Graph

Banker's Algorithm

Multiple instances

Each process must a priori claim maximum use

When a process requests a resource it may have to wait

When a process gets all its resources it must return them in a finite amount of time Let *n* = number of processes, and *m* = number of resources types.

Available *:* Vector of length *m*. If available [*j*] = *k*, there are *k* instances of resource type *Rj* available

Max *: n x m* matrix. If *Max* [*i,j*] = *k*, then process *Pi* may request at most *k* instances of resource type *Rj*

Allocation *: n* x *m* matrix. If Allocation[*i,j*] = *k* then *Pi* is currentlyallocated *k* instances of

Rj

Need : *n* x *m* matrix. If *Need* [*i*,*j*] = *k*, then *Pi* may need *k* more instances of *Rj* to complete its task

Need [*i*,*j*] = *Max* [*i*,*j*] – *Allocation* [*i*,*j*]

Safety Algorithm

Let Work and Finish be vectors of length m and n,respectively.

Initialize: Work = Available

Finish [i] = false fori = 0, 1, …,n- 1

Find an isuch that both:

Finish [i] = false

Needi=Work

If no such iexists, go to steP4

Work=Work+ Allocation i Finish[i] = true go to steP2

If Finish [i] == true for all i, then the system is in a safe state

Resource-Request Algorithm for Process *Pi*

Request = request vector for process *Pi* . If *Requesti* [*j*] = *k* then process *Pi* wants *k* instances of resource type *Rj*

1. If *Requesti* £*Needi* go to steP2. Otherwise, raise error condition, since processhas exceeded its maximum claim
2. If *Requesti* £*Available* , go to steP3. Otherwise *Pi* must wait, since resources are not available

Pretend to allocate requested resources to *Pi* by modifying the state asfollows:

Available=Available –Request;

Allocationi= Allocationi +Requesti ;

Needi =Needi – Requesti;

If safe the resources are allocated toPi

If unsafe Pi must wait, and the old resource-allocation state is restored

Example of Banker's Algorithm(REFER CLASS NOTES)

consider 5 processes *P*0 through *P*4; 3 resource types:

A (10 instances), *B* (5instances), and *C* (7 instances)

Snapshot at time *T*0:

Allocation	*ax*	*Available*
A B C	*A B C*	*A B C*
3 3 2		
	*P*0 0 1 0	7 5 3
	*P*1 2 0 0	3 2 2
	*P*2 3 0 2	9 0 2
	*P*3 2 1 1	2 2 2
	*P*4 0 0 2	4 3 3

Σ The content of the matrix *Need* is defined to be *Max* – *Allocation NeedA B C*
156
satisfies safety criteria
P1 **Request (1,0,2)**
true Check that Request £ Available (that is, (1,0,2) £ (3,3,2)

Allocation	*Need*	*Available*
A B C	*A B C*	*A B C*
*P*0 0 1 0	7 4 3	2 3 0
P1 3 0 2	0 2 0	
*P*2 3 0 2	6 0 0	
*P*3 2 1 1	0 1 1	
*P*4 0 0 2	4 3 1	

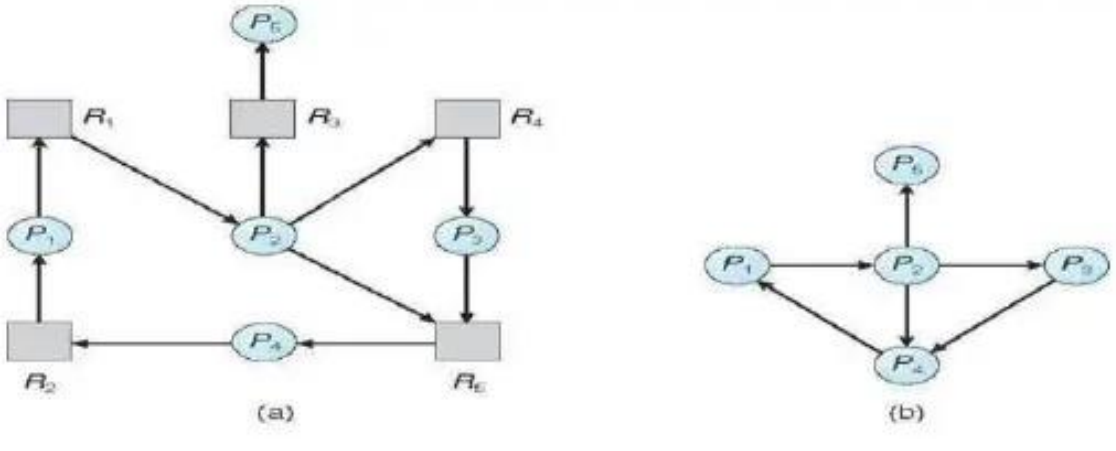
Executing safety algorithm shows that sequence < *P1* , *P*3, *P*4, *P*0, *P*2> satisfies safety requirement

Deadlock Detection
Allow system to enter deadlock state Detection algorithm Recovery scheme

Single Instance of Each Resource Type
Maintain *wait-for* graph Nodes are processes
PiÆPj if *Pi* is waiting for *Pj*
Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle,there exists a deadlock

An algorithm to detect a cycle in a graph requires an order of *n*2 operations,where *n* is the number of vertices in the graph

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph Corresponding wait- for graph

Several Instances of a Resource Type

Allocation *:* An *n* x *m* matrix defines the number of resources of each type currently allocated to each process.

Request *:* An *n* x *m* matrix indicates the current request of each process.
If *Request* [*i*][*j*] = *k*, then process *Pi* is requesting *k* more instances of resource type. *Rj* .

Detection Algorithm

Let Work and Finish be vectors of length m and n, respectively Initialize:
(a)Work = Available
(b)For i = 1,2, …, n, if Allocationiπ 0, then Finish[i] = false; otherwise, Finish[i] = true

2. Find an index isuch that both:
(a)Finish[i] == false
(b)Requesti£Work If no such i exists, go to steP4
3. *Work* = *Work* + *Allocation i Finish* [*i*] = *true*
go to steP2
4. If *Finish* [*i*] == false, for some *i*, 1 £ *i*£*n*, then the system is in deadlock state. Moreover, if *Finish* [*i*] == *false* , then *Pi* is deadlocked

Recovery from Deadlock:

Process Termination

Abort all deadlocked processes
Abort one process at a time until the deadlock cycle is eliminated In which order should we choose to abort?

- Priority of the process
- How long process has computed, and how much longer to completion
- Resources the process has used
- Resources process needs to complete
- How many processes will need to be terminated
- Is process interactive or batch?

Resource Preemption

Selecting a victim – minimize cost
Rollback – return to some safe state, restart process for that state Starvation – same process may always be picked as victim, include number of rollback in cost factor