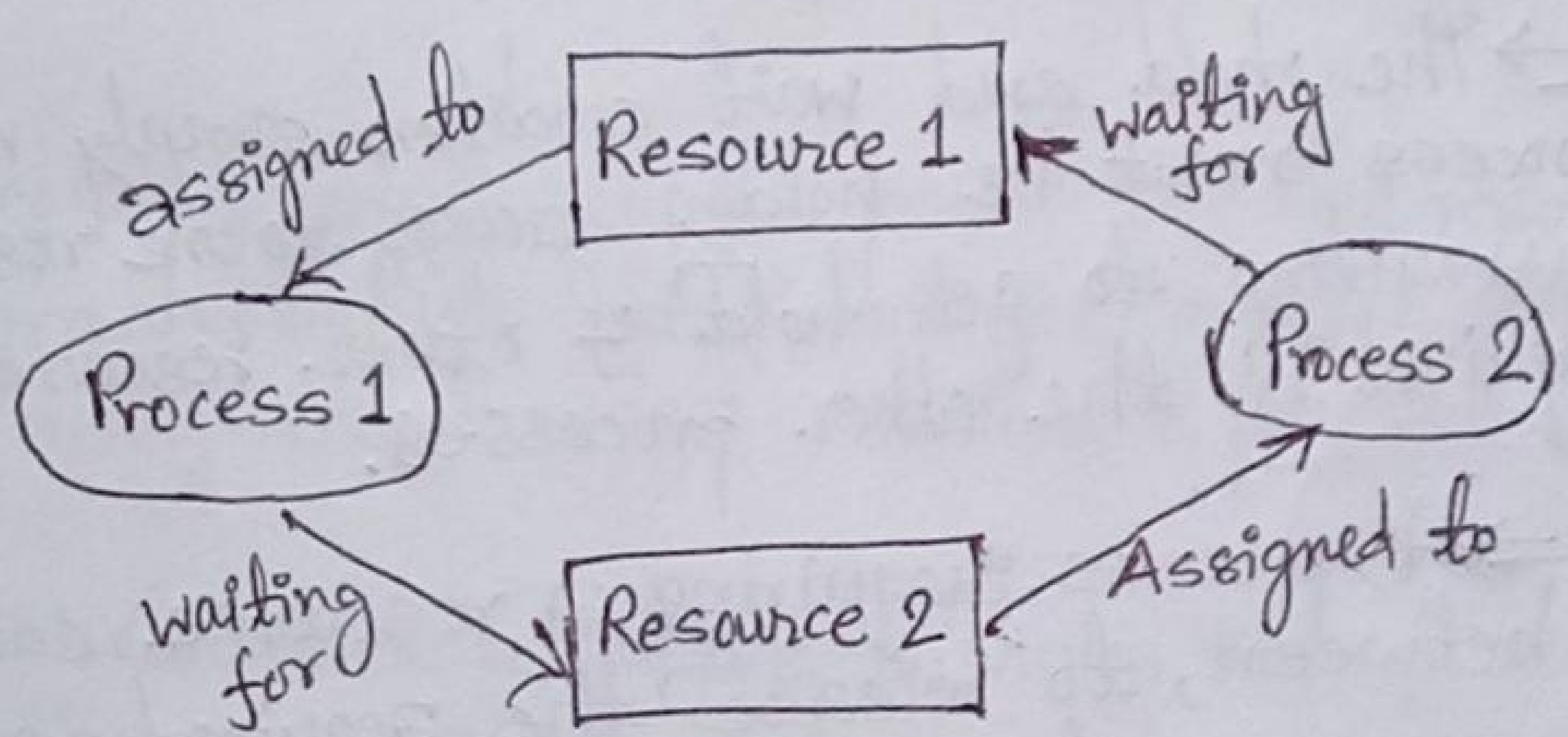
resource → Can be a hardware device or a piece of information. e.g. record in database.

Unit = 3
Process Deadlocks:

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process. In operating system when there are two or more processes that hold some resources and wait for resources held by other(s), then deadlock occurs. For example, in below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by Process 2, and Process 2 is waiting for resource 1.



⊛. Resources: To make the discussion of deadlocks as general as possible, we will refer to the objects granted as resources. A resource can be a hardware device (e.g. a Blu-ray drive) or a piece of information (e.g. a record in database). A computer will have many different resources that a process can acquire. A resource is anything that must be acquired, used and released over the course of time. Resources are of two types: preemptable and non preemptable resources.

Preemptable is a resource that can be taken away from its current owner/place and given back later & non preemptable is one that can not be given back.
For example:- Memory is an example of a preemptable resource. Blu-ray recorders are not preemptable at any arbitary moment. Whether a resource is preemptible depends on the context.

⊛. Deadlock Characterization:

Deadlock characterization describes the distinctive features that are the cause of deadlock occurance. It consists of deadlock prerequisites and System resource allocation graph.

1>. Deadlock Prerequisites (i.e, Necessary conditions for Deadlock):

i) Mutual Exclusion → In a multiprogramming environment, there may be several processes requesting the same resource at a time. The mutual exclusion condition, allow only a single process to access the resource at a time. While the other processes requesting the same resource must wait and delay their execution until it has been released.

ii) Hold and wait → The hold and wait condition simply means that the process must be holding access to one resource and must be waiting to get hold of other resources that have been acquired by the other processes.

iii) No preemption → A process acquiring a resource, can not be preempted in between, to release the acquired resource. Instead, the process must release the resource it has acquired when the task of the process has been completed.

iv) Circular Wait → The processes must be waiting in a circular pattern to acquire the resource. This is similar to hold and wait. The difference is that the processes are waiting in a circular pattern.

2> Resource Allocation Graph:

It is a directed graph that briefs about the deadlock more precisely. Like every graph, it also has a set of vertices and a set of edges. Further, the set of vertices can be classified into two types of nodes P and R. Where P is the set of vertices indicating the set of active processes and R is the set of vertices indicating all types of resources in the system.

⇒ When a process requests for a resource it is denoted by the request edge in the resource-allocation graph. It is directed edge requesting process P_i to requested resource R_j, i.e, $P_i \to R_j$.

⇒ When a resource is alloted to some process then it is denoted by the assignment edge. The assignment edge is the directed edge from the instance of resource R_j to process P_i i.e, $R_j \to P_i$.

⇒ In the graph, resources are denoted by the rectangles and the processes are denoted by the circles. If a resource has multiple instances then it is denoted by dots inside the rectangle.

⇒ When a process request for an instance of the resource it directs a request edge to the resource. If the resource is able to allocate the resource instance to the requesting process then immediately the request edge is converted to assignment edge.

⇒ The request edge always points to the resource rectangle in the graph, not to dots (instance) inside the rectangle. Although the assignment edge nominates the dot (instance) to a process.

Example: Consider we have following set of nodes and edges.

1. There are three active processes $P = \{P_1, P_2, P_3\}$
2. There are four resources $R = \{R_1, R_2, R_3, R_4\}$
3. The set of request edge and assignment edges we have $E = \{P_1 \to R_1, P_2 \to R_3, R_1 \to P_2, R_2 \to P_2, R_2 \to P_1, R_3 \to P_3\}$

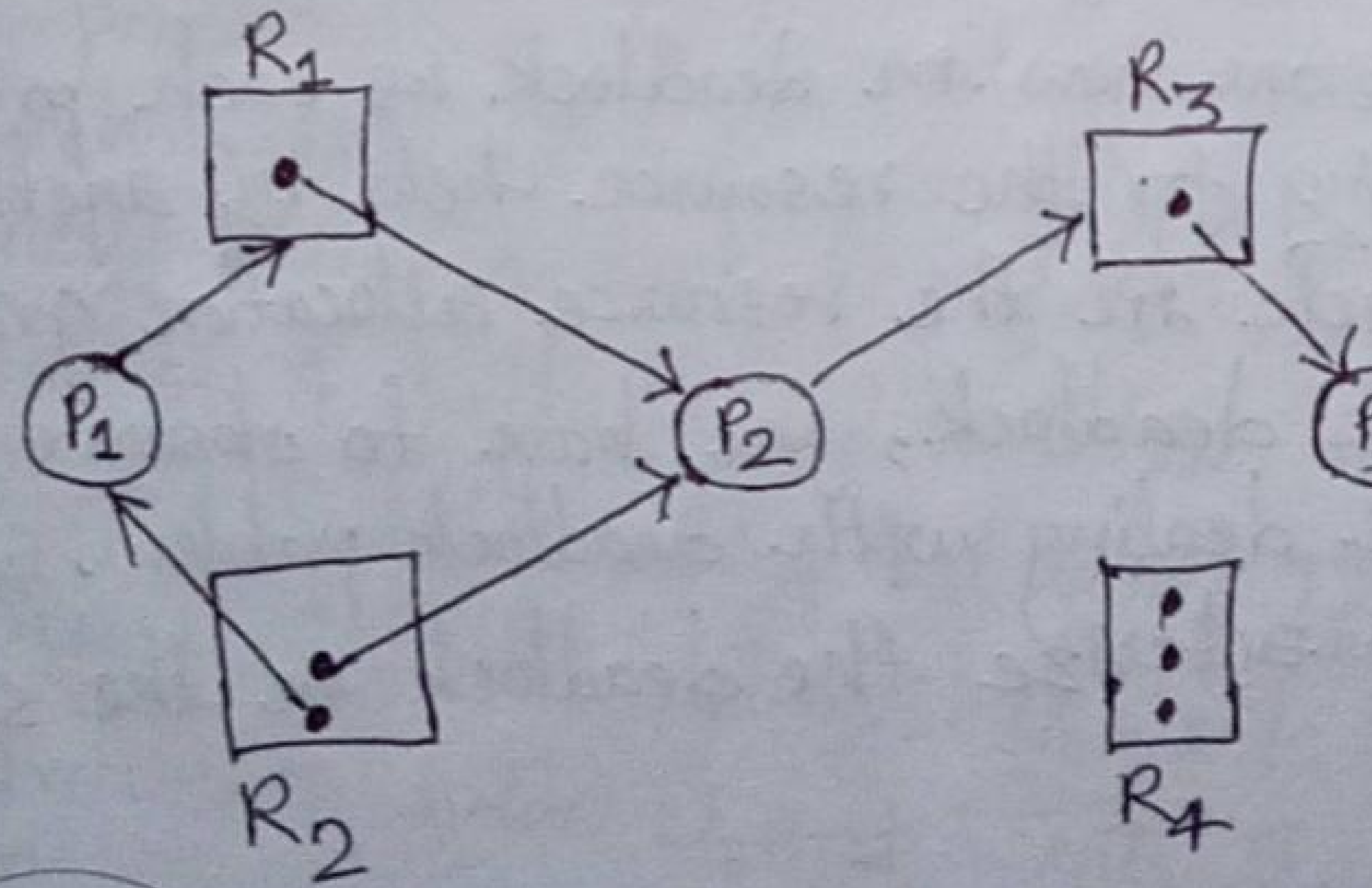


fig. Resource Allocation Graph

The figure shows that the process P_1 has requested for instance of resource R_1 is already holding the instance of resource R_2. The process P_2 has requested for the instance of resource R_3 and is already holding the instances of resource R_1 and R_3. The process P_3 has not requested for any resource instance but is holding the instance for resource R_3.

If the resource allocation graph has a cycle and every resource has a single instance then it implies that a deadlock has occured. In case, the resources has multiple instances then a cycle in the graph need not be indicating the occurance of deadlock.

Consider that the process P_3 is requesting for the instance of resource R_2 which is already held by the process P_1 and P_2. In this case, we will observe that there are two cycles in the resource allocation graph:

- $P_1 \to R_1 \to P_2 \to R_3 \to P_3 \to R_2 \to P_1$
- $P_2 \to R_3 \to P_3 \to R_2 \to P_2$

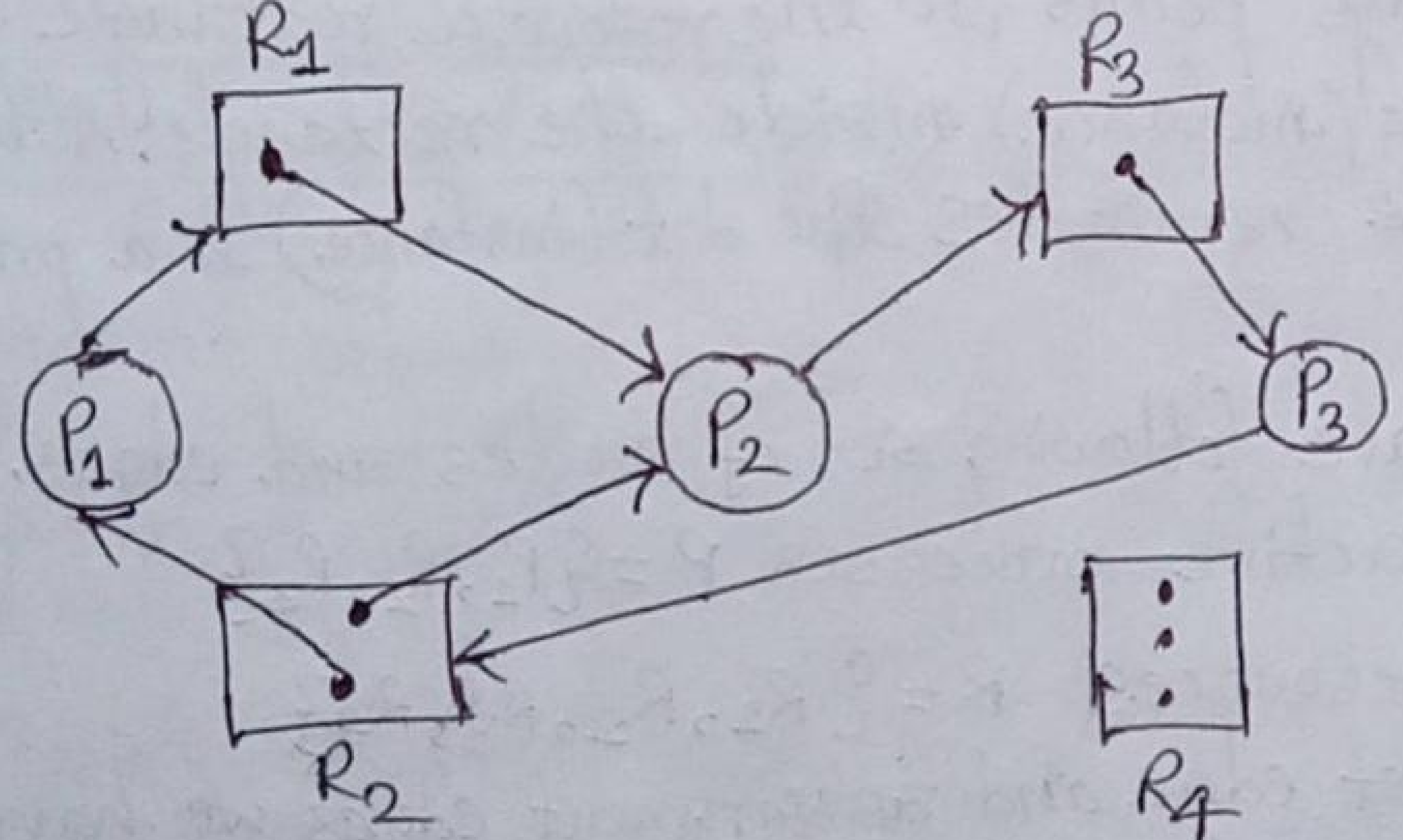


fig. Resource allocation graph with Deadlock

Process P_1, P_2 and P_3 are now in deadlock as each process in the cycle is waiting for the resource held by another process. But every cycle in the resource allocation graph does not indicate the deadlock, you have to observe the cycle carefully while dealing with deadlock problem. So, this is how you can characterize the deadlock in the system.

#Handling Deadlocks:

1) The Ostrich Algorithm: The ostrich algorithm is a strategy of ignoring potential problems on the basis that they may be exceedingly rare. It is named for the ostrich effect which is defined as "to stick one's head in the sand and pretend there is no problem". It is used when it is more cost-effective to allow the problem to occur than to attempt its prevention. This approach may be used in dealing with deadlocks in concurrent programming if they are beleaved to be very rare and the cost of detection or prevention is high. The UNIX and Windows operating system take this approach.

2) Deadlock Detection:

a. Deadlock Detection with one resource of Each Type:

For any system we construct a resource graph. If graph contains one or more cycles, a deadlock exists. Any process that is part of a cycle is deadlocked. If no cycles exist, the system is not deadlocked.

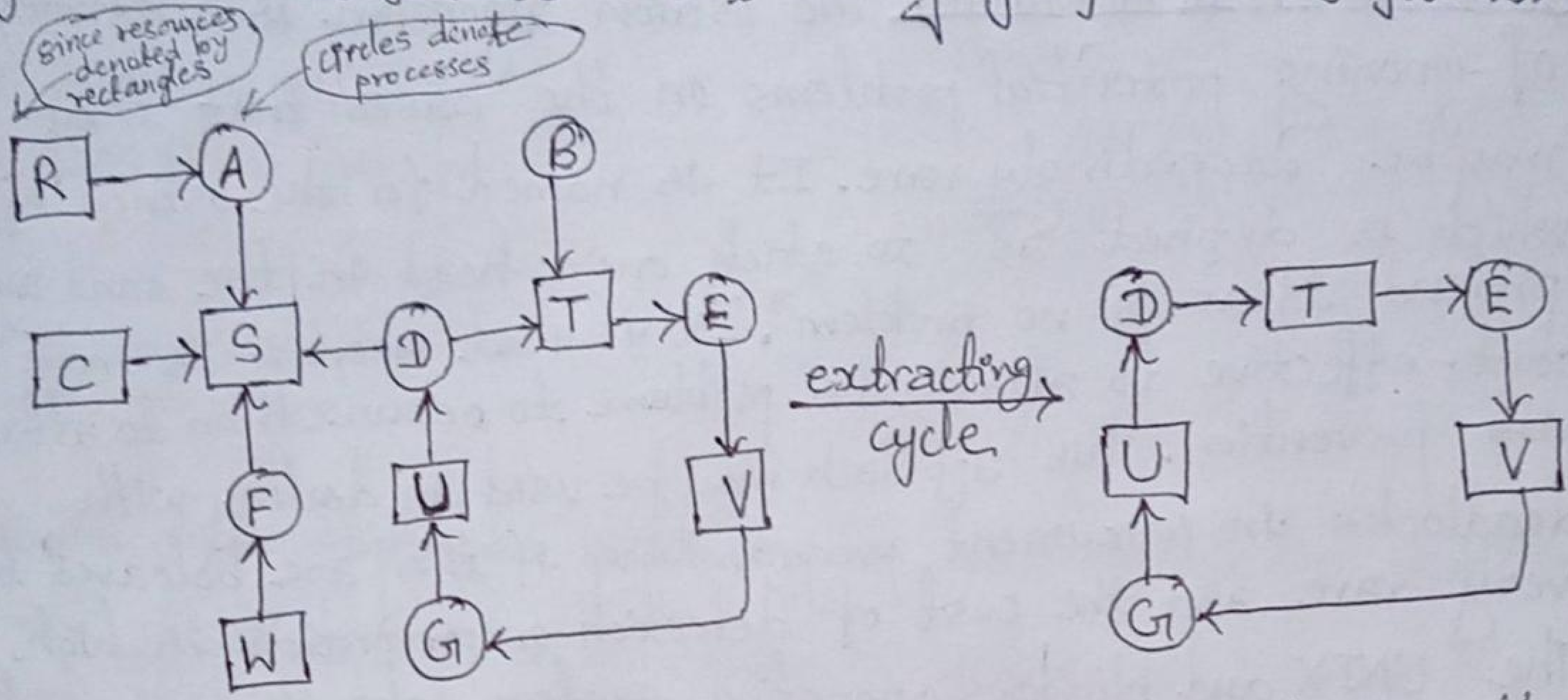
In this method only one resource of each type exists. Such a system might have one scanner, one CD recorder, one plotter etc. but no more than one of each class of resource.

Let us take an example: Consider a system with seven processes, A to G and six resources R to W. The state of which resources are currently owned and which ones are currently being requested as follows:

i) Process A holds R and wants S.
ii) Process B holds nothing but wants T.
iii) Process C holds nothing but wants S.
iv) Process D holds U and wants S and T.
v) Process E holds T and wants V.
vi) Process F holds W and wants S.
vii) Process G holds V and wants U.

The question is: "Is this system deadlocked, and if so, which processes are involved"?

To answer this question first we construct the resource graph of given data ~~as follows~~ and extracting cycle from it as follows:-



We can see that this graph contains one cycle. From this cycle we can see that processes D, E and G are all deadlocked.

(b). Deadlock Detection with multiple resources of each type:

When multiple copies of some of the resources exist, a different approach is needed to detect deadlocks. We will now present a matrix-based algorithm for detecting deadlock among n processes, P_1 to P_n. Let the number of resource classes be m, with E_1 resources of class 1, E_2 resources of class 2, and generally E_i resources of class i ($1 < i < m$). E is the existing resource vector. For example if class 1 is tape drivers, then $E_1 = 2$ means the system has two tape drives.

At any instant, some of the resources are assigned and are not available. Let A be the available resource vector, with A_i giving number of instances of resource i that are currently available (i.e, unassigned). For example if both of our two tape drivers are assigned then A_1 will be 0.

Now we need two arrays, C the current allocation matrix and R the request matrix.

where,

C_{ij} = no. of instances of resource j that are held by process i.

R_{ij} = no. of instances of resource j that P_i wants.

Resources in existence
$(E_1, E_2, E_3, \ldots, E_m)$

Resources available
$(A_1, A_2, A_3, \ldots A_m)$

Current allocation matrix

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$$

Row n is current allocation to process n.

Request matrix.

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Row 2 is what process 2 needs.

Fig. The four structures needed by the deadlock detection algorithm.

This algorithm is based on comparing vectors. Let us define the relation $A \leq B$ on two vectors A and B to mean that each element of A is less than or equal to the corresponding element of B. Each process is initially said to be unmarked. As the algorithm processes, processes will be marked, indicating they are able to complete and are thus not deadlocked. When the algorithm terminates, any unmarked processes are known to be deadlocked.

The deadlock detection algorithm can now be given, as follows:-

i). Look for an unmarked process P_i for which the ith row of R is less than or equal to A.

ii) If such a process is found, add the ith row of C to A, mark the process and go back to step 1. (i.e, step i).

iii). If no such process exists, the algorithm terminates.

The selected process is run until it finishes. If all the processes are ultimately able to run, then none of them are deadlocked. If some of them can never run, they are deadlocked.

Example:

$$E = \begin{array}{cccc} \text{Tape drivers} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ 4 & 2 & 3 & 1 \end{array}$$

$$A = \begin{array}{cccc} \text{Tape drivers} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ 2 & 1 & 0 & 0 \end{array}$$

current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Fig. An example of deadlock detection algorithm.

3) Deadlock Prevention: (Deadlock Avoidance):

Deadlocks can be prevented by preventing at least one of the four required conditions for to happen deadlock which are mutual exclusion, hold & wait, no preemption and circular wait. This requires more information about each process. The state of the system informs that if resources are allocated to different processes then the system undergoes deadlock or not.

⊛ Safe and Unsafe States:

System is in safe state if there exists a safe sequence of all processes. Sequence $< P_1, P_2, \ldots, P_n >$ is safe if for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < i$.

→ If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.

→ When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.

→ When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

Note:

i) If a system is in safe state ⇒ no deadlocks.

ii) If a system is in unsafe state ⇒ possibility of deadlock.

iii) Avoidance ⇒ Ensure that a system will never enter an unsafe state.

fig. safe, unsafe, deadlock state

@ The Banker's Algorithm for a Single Resources:-

The extension of deadlock detection algorithm, ~~as~~ and a scheduling algorithm that can avoid deadlocks is due to Dijkstra (1965); is known as the banker's algorithm.

It is modeled on the way a small-town banker might deal with a group of customers to whom he has granted lies of credit. (Years ago, banks did not lend money unless they knew they could be repaid). What the algorithm does is check to see if granting the request leads to an unsafe state. If so, the request is denied. If granting the request leads to safe state, then it is carried out.

The banker's algorithm considers each request as it occurs, seeing whether granting it leads to a safe state. If it does, the request is granted; otherwise it is postponded until later. To see if a state is safe, the banker checks to see if he has enough resources to satisfy some customer. If so, those loans are assumed to be repaid, and the customer now closest to the limit is checked and so on. If all loans can eventually be repaid, the state is safe and the initial request can be granted.

For Example:- If we have situation as ~~bel~~ figure below then it is safe state because with 10 free units one by one all customers can be served.

Process	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7
Free: 10		

Note:- to understand how this is ~~sa~~ in safe state and to understand each process once refer youtube video by darshan institute of engineering and technology.

⑮ The Banker's Algorithm for Multiple Resources:-

The bankers algorithm can be generalized to handle multiple resources as in figure below:-

Process	Tape drives	Plotters	Printers	Blu-rays
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Printers	Blu-rays
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still assigned.

Fig. The banker's algorithm with multiple resources.

The algorithm for checking to see if a state is safe can now be stated as;

i) Look for a row, R, whose unmet resource needs are all smaller than or equal to A. If no such row exists, the system will eventually deadlock since no process can run to completion.

ii) Assume the process of the chosen row requests all the resources it needs (which is guaranteed to be possible) and finishes. Mark that process as terminated and add all of it's resources to the A vector.

iii) Repeat steps (i) and (ii) until either all processes are marked terminated (in which case the initial state was safe). or no process is left whose resource needs can be met (in which case the system was not safe).

If several processes are eligible to be chosen in step (i), it does not matter which one is selected.

4) Recovery from Deadlock:

When deadlock is detected, then our system stops working, and after the recovery of the deadlock, our system start working again. Therefore, after the detection of deadlock, a method/way must require to recover that deadlock to run the system again. The method/way is called as deadlock recovery. Following are the two main ways of deadlock recovery:-

ⓐ. Deadlock Recovery through Preemption:

It is the ability to take a resource away from a process, have another process ue it, and then give it back without the process noticing. It is highly dependent on the nature of the resource. Deadlock recovery through preemption is too difficult or sometime impossible.

ⓑ. Deadlock Recovery through RollBack:

In this case, whenever a deadlock is detected, it is easy to see which resources are needed. To do the recovery of deadlock, a process that owns a needed resource is rolled back to a point in time before it acquired some other resource just by starting one of its earlier checkpoints.

If my notes really helped you,then you can support me on esewa for my hardwork.

Esewa ID: 9806470952