

Deadlocks

Contents

3.1. Introduction, Deadlock Characterization, Preemptable and Non-preemptable Resources, Resource – Allocation Graph, Conditions for Deadlock .

3.2. Handling Deadlocks: Ostrich Algorithm, Deadlock prevention, Deadlock Avoidance, Deadlock Detection (For Single and Multiple Resource Instances), Recovery From Deadlock (Through Preemption and Rollback)

- A process may also need exclusive access to multiple resources, *e.g.*:
 - Two processes each want to record a scanned document on a Blu-ray disc
 - Process A requests permission to use the scanner and is granted it;
 - Process B requests permission to use the Blu-ray driver and is granted it;
 - Process A then asks for the Blu-ray recorder:
 - But request is suspended until B releases it;
 - Unfortunately instead of releasing Blu-ray recorder:
 - Process B asks for the scanner;
 - At this point both processes are blocked and will remain so forever.
- This situation is called a **deadlock**.

- Deadlocks can also occur in a variety of other situations.. In a database system, for example,

Database system, where program may have to lock several registers *e.g.*:

- Process A locks records R1;
- Process B locks record R2;
- If each process tries to lock each other one's record, we also have a deadlock.

Conclusion:

- Deadlocks can occur on hardware resources or on software resources.
- Deadlocks happen throughout computer science!

Resources

- Examples of computer resources
 - printers
 - tape drives
 - Tables
- Resource can be:
 - Hardware device;
 - Piece of information;
- Processes need access to resources in reasonable order
- Suppose a process holds resource A and requests resource B
 - at same time another process holds B and requests A
 - both are blocked and remain so

Preemptable Resource:

- Can be taken away from the process owning it with no ill effects.

Memory is a preemptable resource, consider a system with: for example,

- A system with 1 GB of user memory, one printer, and two 1-GB processes that each want to print something.
- Process A requests and gets the printer, then starts to compute the values to print. Before it has finished the computation, it exceeds its time quantum and is swapped out to disk.

Example 2

- Process A requests and gets the printer, then starts to compute the values to print. Before it has finished the computation, it exceeds its time quantum and is swapped out to disk.

- Process B now runs and tries, unsuccessfully as it turns out, to acquire the printer. Potentially, we now have a deadlock situation, because A has the printer and B has the memory, and neither one can proceed without the resource held by the other.

Nonpreemptable resource:

- Cannot be taken away from its current owner without potentially causing failure

Example: If a process has begun to burn a Blu-ray

- Cannot simply give the Blu-ray drive to another process;
- This would simply result in a garbled Blu-ray;
- Blu-ray recorders are not preemptable at an arbitrary moment.

So, how can we deal with no preemptable resources?

Abstract sequence of events required to use a resource:

- Request the resource.
- Use the resource.
- Release the resource.

- If the resource is not available when it is requested, the requesting process is forced to wait.
- In some operating systems, the process is automatically blocked when a resource request fails, and awakened when it becomes available.
- In other systems, the request fails with an error code, and it is up to the calling process to wait a little while and try again.

Introduction to Deadlocks

- Formal definition :

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause

- Usually the event is release of a currently held resource
- None of the processes can ...
 - run
 - release resources
 - be awakened

Four Conditions for Deadlock

- 1. Mutual exclusion condition.** Each resource is either currently assigned to exactly one process or is available.
- 2. Hold-and-wait condition.** Processes currently holding resources that were granted earlier can request new resources.
- 3. No-preemption condition.** Resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them.
- 4. Circular wait condition.** There must be a circular list of two or more processes, each of which is waiting for a resource held by the next member of the chain.

Deadlock Modeling

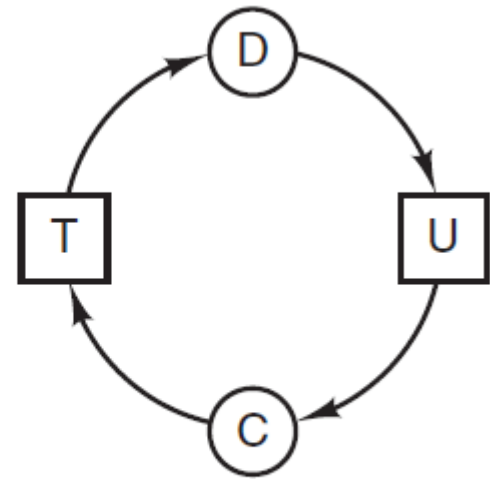
Modeled with directed graphs



(a)



(b)



(c)

Figure 3-1. Resource allocation graphs. (a) Holding a resource. (b) Requesting a resource. (c) Deadlock.

- resource R assigned to process A
- process B is requesting/waiting for resource S
- process C and D are in deadlock over resources T and U

A
Request R
Request S
Release R
Release S

(a)

B
Request S
Request T
Release S
Release T

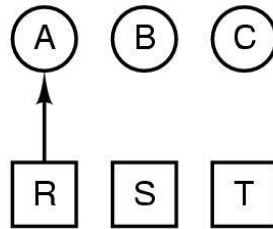
(b)

C
Request T
Request R
Release T
Release R

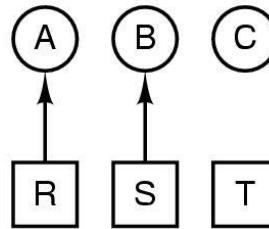
(c)

1. A requests R
 2. B requests S
 3. C requests T
 4. A requests S
 5. B requests T
 6. C requests R
- deadlock

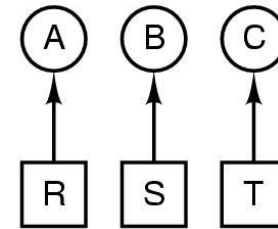
(d)



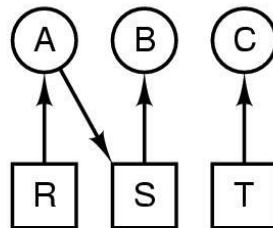
(e)



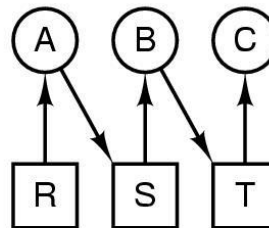
(f)



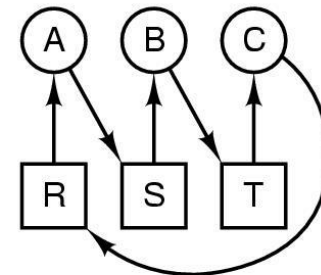
(g)



(h)



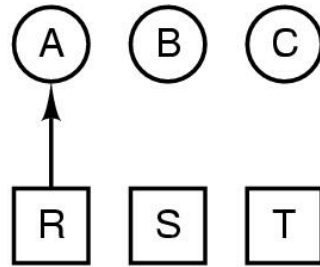
(i)



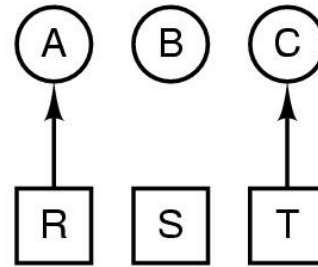
(j)

1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S
no deadlock

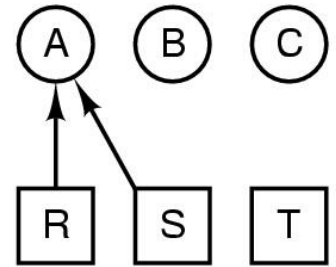
(k)



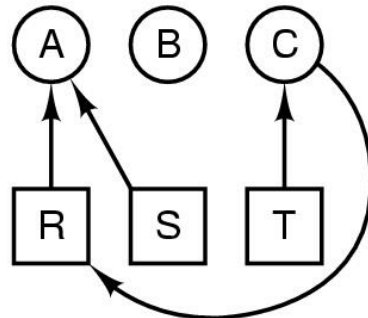
(l)



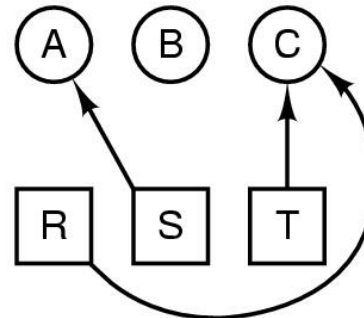
(m)



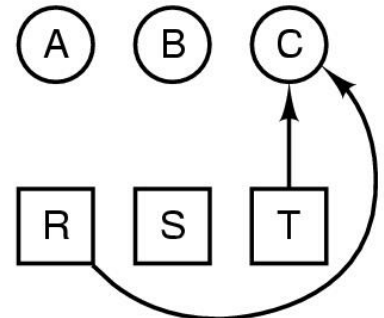
(n)



(o)



(p)



(q)

Figure 3-2. An example of how deadlock occurs and how it can be avoided.

Methods for Handling Deadlocks

Deadlock Prevention

- Ensure that *at least one* of four necessary conditions cannot hold

Deadlock Avoidance

- Do not allow a resource request → Potential to lead to a deadlock
- Requires advance info of all requests

Deadlock Detection

- Always allow resource requests
- Periodically check for deadlocks
- If a deadlock exists → Recover from it

Ignore

- Makes sense if the likelihood is very low, say once per year
- Cheaper than *prevention*, *avoidance* or *detection*
- Used by most common OS

Deadlock Ignorance(The ostrich algorithm)

Ostrich algorithm:

- Stick your head in the sand and pretend there is no problem;
- People react to this strategy in different ways:
- Mathematicians: find it unacceptable and say that deadlocks must be prevented at all costs.
- Engineers: Maybe not worth to deal with deadlocks:
- How often is the problem expected?
- How often the system crashes for other reasons?
- How serious a deadlock is?

Deadlock Detection and Recovery

1. Deadlock Detection with One Resource of Each Type

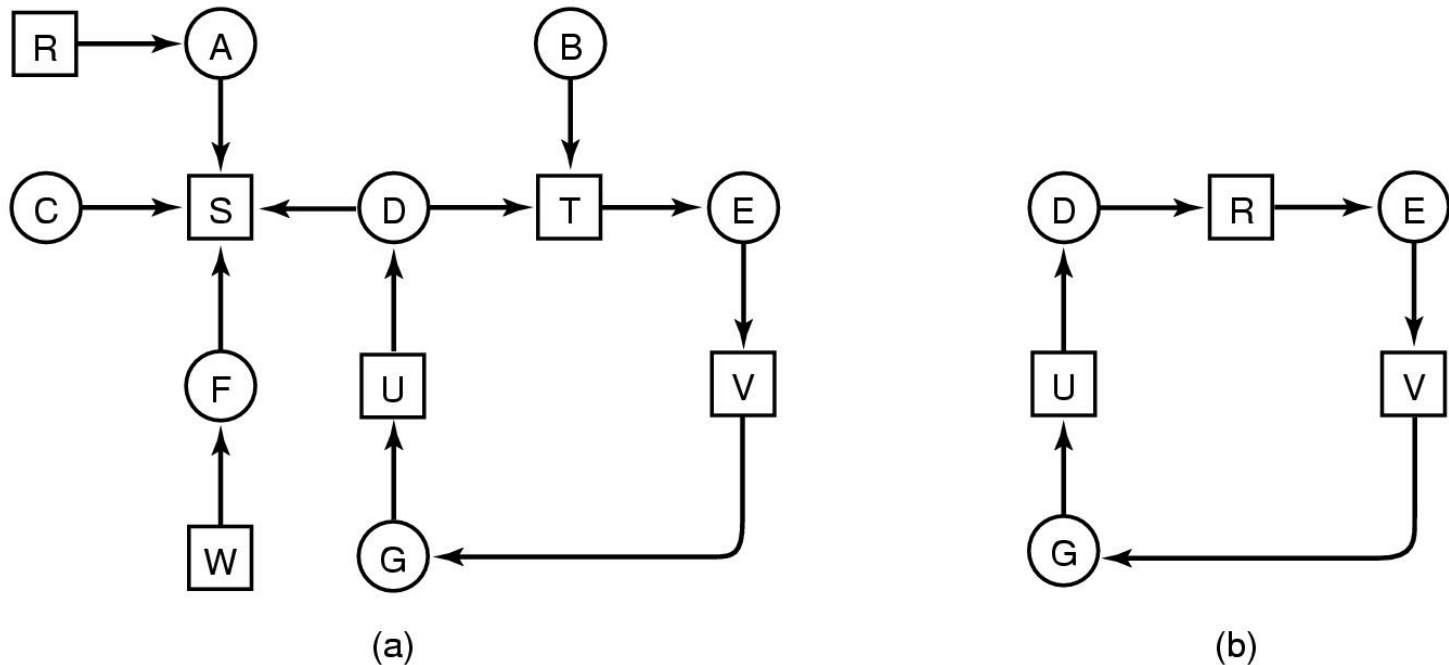


Figure 3-3. (a) A resource graph. (b) A cycle extracted from (a).

- The state of which resources are currently owned and which ones are currently being requested is as follows:
 - Process A holds R and wants S .
 - Process B holds nothing but wants T .
 - Process C holds nothing but wants S .
 - Process D holds U and wants S and T .
 - Process E holds T and wants V .
 - Process F holds W and wants S .
 - Process G holds V and wants U .
- The cycle is shown in Fig. 3(b). From this cycle, we can see that processes D , E , and G are all deadlocked. Processes A , C , and F are not deadlocked because S can be allocated to any one of them, which then finishes and returns it.
- A cycle can be found within the graph, denoting deadlock

2. 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 through P_n .
- Sum of current resource allocation + resources available = resources that exist

$$\sum_{i=1}^n C_{ij} + A_j = E_j$$

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

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


Current allocation matrix



C_{11}	C_{12}	C_{13}	\dots	C_{1m}
C_{21}	C_{22}	C_{23}	\dots	C_{2m}
\vdots	\vdots	\vdots		\vdots
C_{n1}	C_{n2}	C_{n3}	\dots	C_{nm}

Row n is current allocation
to process n

Request matrix



R_{11}	R_{12}	R_{13}	\dots	R_{1m}
R_{21}	R_{22}	R_{23}	\dots	R_{2m}
\vdots	\vdots	\vdots		\vdots
R_{n1}	R_{n2}	R_{n3}	\dots	R_{nm}

Row 2 is what process 2 needs

Figure 3-4. The four data structures needed by the deadlock detection algorithm.

$$E = \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix}$$

Tape drives
Plotters
Scanners
Blu-rays

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix}$$

Tape drives
Plotters
Scanners
Blu-rays

Current allocation matrix

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

Request matrix

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

Figure 3-5. An example for the deadlock detection algorithm.

- The first one cannot be satisfied because there is no Bluray drive available. The second cannot be satisfied either, because there is no scanner free.
- Fortunately, the third one can be satisfied, so process 3 runs and eventually returns all its resources, giving $A = (2 \ 2 \ 2 \ 0)$

Recovery from Deadlock

Recovery

- Abort one process at a time until the deadlock cycle is eliminated.
- A simpler way (used in large main frame computers):
- Do not maintain a resource graph. Only periodically check to see if there are any processes that have been blocked for a certain amount of time, say, 1 hour. Then kill such processes.
- To recover the killed processes, need to restore any modified files. Keep different versions of the file.

1. Recovery from Deadlock

- Recovery through preemption
 - take a resource from some other process
 - depends on nature of the resource
- For example, to take a laser printer away from its owner, the operator can collect all the sheets already printed and put them in a pile. Then the process can be suspended (marked as not runnable). At this point the printer can be assigned to other process.
- When that process finishes, the pile of printed sheets can be put back in the printer's output tray and the original process restarted.

2. Recovery through rollback

- checkpoint a process periodically
- Checkpointing a process means that its state is written to a file so that it can be restarted later.
- use this saved state
- restart the process if it is found deadlocked

3. Recovery through killing processes

- The crudest but simplest way to break a deadlock is to kill one or more processes.
- One possibility is to kill a process in the cycle.
- the other processes get its resources
- choose process that can be rerun from the beginning

- For example, a compilation can always be rerun because all it does is read a source file and produce an object file. If it is killed partway through, the first run has no influence on the second run.

Deadlock Prevention

- Resource allocation rules prevent deadlock by prevent one of the four conditions required for deadlock from occurring
 - Mutual exclusion
 - Hold and wait
 - No preemption
 - Circular Wait

1. Attacking the Mutual Exclusion Condition

- Allow everybody to use the resources immediately they require!
- By spooling printer output, several processes can generate output at the same time.
- In this model, the only process that actually requests the physical printer is the printer daemon. Since the daemon never requests any other resources, we can eliminate deadlock for the printer.

2. Attacking the Hold and Wait Condition

- Require processes to request resources before starting
 - a process never has to wait for what it needs

Problems

- may not know required resources at start of run
- also ties up resources other processes could be using

Variation:

- process must give up all resources
- then request all immediately needed

3. No 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*
- *Not realistic* for many types of resources, such as *printers*

4. Circular Wait

- Impose a total ordering of all resource types
- *Each process requests resources in an increasing order of enumeration*

Deadlock avoidance

- Instead of detecting deadlock, can we simply avoid it?
 - YES, but only if enough information is available in advance.
 - Maximum number of each resource required

Resource Trajectories

- The main algorithms for deadlock avoidance are based on the concept of safe states.
- Graphical approach does not translate directly into a usable algorithm, it gives a good intuitive feel for the nature of the problem.

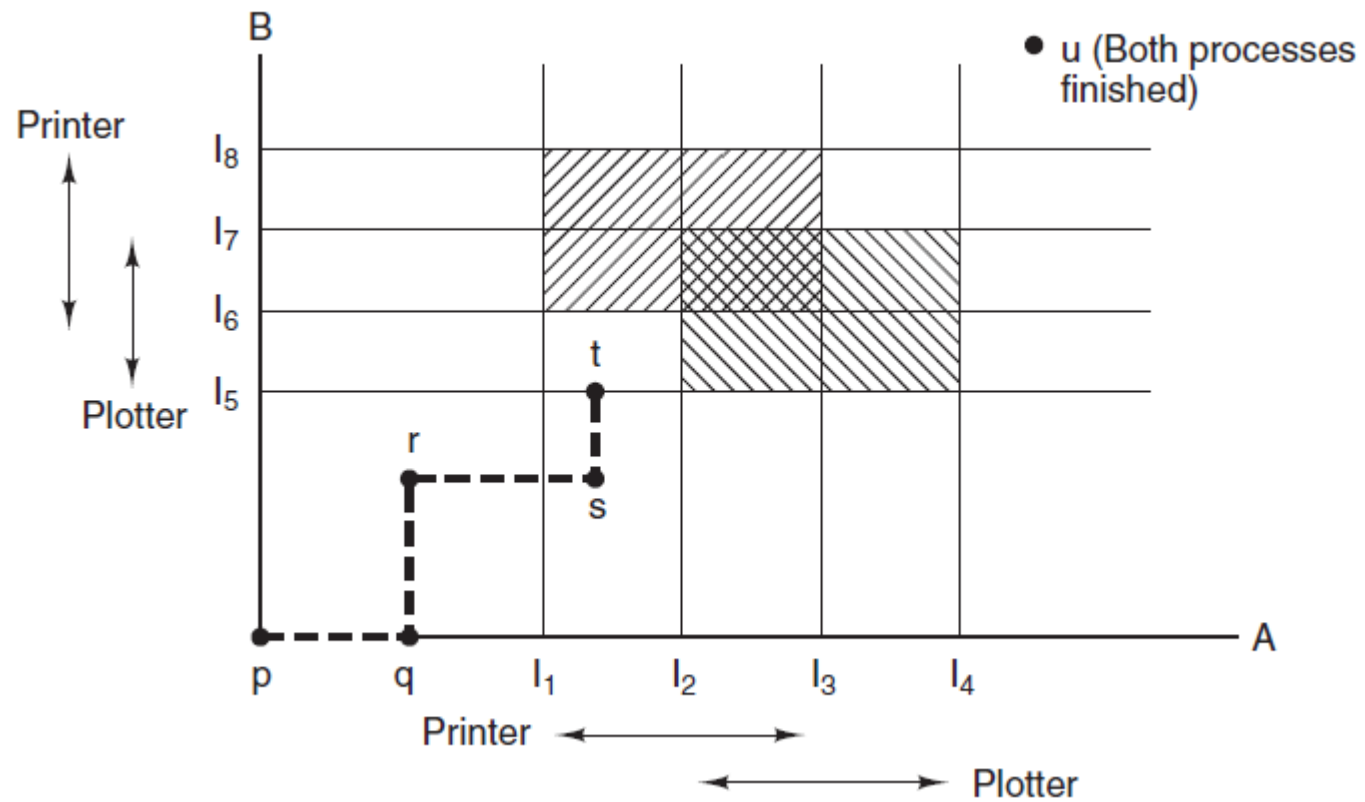


Figure 3-6. Two process resource trajectories.

Safe and Unsafe States

- A state is said to be **safe** if there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources immediately.
- In Fig. (a) we have a state in which *A* has three instances of the resource but may need as many as nine eventually. *B* currently has two and may need four altogether, later. Similarly, *C* also has two but may need an additional five. A total of 10 instances of the resource exist, so with seven resources already allocated, three there are still free.

Has Max			Has Max			Has Max			Has Max			Has Max		
A	3	9	A	3	9	A	3	9	A	3	9	A	3	9
B	2	4	B	4	4	B	0	–	B	0	–	B	0	–
C	2	7	C	2	7	C	2	7	C	7	7	C	0	–
Free: 3			Free: 1			Free: 5			Free: 0			Free: 7		
(a)			(b)			(c)			(d)			(e)		

Figure 3-7 . Demonstration that the state in (a) is safe.

- The state of Fig.(a) is safe because there exists a sequence of allocations that allows all processes to complete. Namely, the scheduler can simply run *B* exclusively, until it asks for and gets two more instances of the resource, leading to the state of Fig.(b). When *B* completes, we get the state of Fig.(c). Then the scheduler can run *C*, leading eventually to Fig.(d). When *C* completes, we get Fig.(e). Now *A* can get the six instances of the resource it needs and also complete. Thus, the state of Fig.(a) is safe because the system, by careful scheduling, can avoid deadlock.
- Safe states guarantee we will eventually complete all processes.

Has Max		
A	3	9
B	2	4
C	2	7
Free: 3		
(a)		

Has Max		
A	4	9
B	2	4
C	2	7
Free: 2		
(b)		

Has Max		
A	4	9
B	4	4
C	2	7
Free: 0		
(c)		

Has Max		
A	4	9
B	—	—
C	2	7
Free: 4		
(d)		

Figure 3-8. Demonstration that the state in (b) is not safe.

- There is no sequence that guarantees completion. Thus, the allocation decision that moved the system from Fig.(a) to Fig.(b) went from a safe to an unsafe state. Running *A* or *C* next starting at Fig.(b) does not work either.

The Banker's Algorithm for a Single Resource

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

(a)

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

(b)

	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

(c)

Figure 3-9. Three resource allocation states: (a) Safe. (b) Safe. (c) Unsafe.

- The banker's algorithm was first published by Dijkstra in 1965.
- **Banker's Algorithm** is used majorly in the banking system to avoid deadlock. It helps you to identify whether a loan will be given or not.
- In Fig. 6-11(a) we see four customers, *A*, *B*, *C*, and *D*, each of whom has been granted a certain number of credit units.
- The banker knows that not all customers will need their maximum credit immediately, so he has reserved only 10 units rather than 22 to service them. (In this analogy, customers are processes, units are, say, tape drives, and the banker is the operating system.)

The Banker's Algorithm for Multiple Resources

	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					

E = (6342)
P = (5322)
A = (1020)

Figure 3-10. The banker's algorithm with multiple resources.

- The three vectors at the right of the figure show the existing resources, E , the possessed resources, P , and the available resources, A , respectively. From E we see that the system has six tape drives, three plotters, four printers, and two Blu-ray drives.
- Five tape drives, three plotters, two printers, and two Blu-ray drives are currently assigned.

Data Structures for the Banker's Algorithm

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 R_j available
- **Max:** $n \times m$ matrix. If $Max [i,j] = k$, then process P_i may request at most k instances of resource type R_j
- **Allocation:** $n \times m$ matrix. If $Allocation[i,j] = k$ then P_i is currently allocated k instances of R_j
- **Need:** $n \times m$ matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task

$$Need [i,j] = Max[i,j] - Allocation [i,j]$$

Safety Algorithm

1. Let *Work* and *Finish* be vectors of length m and n , respectively. Initialize:

Work = *Available*

Finish [i] = *false* for $i = 0, 1, \dots, n-1$

2. Find an i such that both:

(a) *Finish* [i] = *false*

(b) $Need_i \leq Work$

If no such i exists, go to step 4

3. *Work* = *Work* + *Allocation* _{i}

Finish [i] = *true*

go to step 2

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

Resource-Request Algorithm for Process P_i

$\mathbf{Request}_i$ = request vector for process P_i . If $\mathbf{Request}_i[j] = k$ then process P_i wants k instances of resource type R_j

1. If $\mathbf{Request}_i \leq \mathbf{Need}_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If $\mathbf{Request}_i \leq \mathbf{Available}$, go to step 3. Otherwise P_i must wait, since resources are not available
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$$\mathbf{Available} = \mathbf{Available} - \mathbf{Request}_i;$$

$$\mathbf{Allocation}_i = \mathbf{Allocation}_i + \mathbf{Request}_i;$$

$$\mathbf{Need}_i = \mathbf{Need}_i - \mathbf{Request}_i;$$

- If safe \Rightarrow the resources are allocated to P_i
- If unsafe $\Rightarrow P_i$ must wait, and the old resource-allocation state is restored

Example :1

- 5 processes P_0 through P_4 ;
3 resource types: A (10 instances), B (5 instances), and C (7 instances)
- Snapshot at time T_0 :

	Allocation	Max	Available
	<i>A B C</i>	<i>A B C</i>	<i>A B C</i>
P_0	0 1 0	7 5 3	3 3 2
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	

- The content of the matrix *Need* is defined to be *Max – Allocation*

	Need		
	<i>A</i>	<i>B</i>	<i>C</i>
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

- The system is in a safe state since the sequence $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ satisfies safety criteria

Example: P_1 Request (1,0,2)

- Check that Request \leq Available (that is, $(1,0,2) \leq (3,3,2) \Rightarrow$ true

	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	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 $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ satisfies safety requirement
- Can request for (3,3,0) by P_4 be granted?
- Can request for (0,2,0) by P_0 be granted?

Example :2 Consider a system that contains five processes P1, P2, P3, P4, P5 and the three resource types A, B and C. Following are the resources types: A has 10, B has 5 and the resource type C has 7 instances.

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2
P2	2	0	0	3	2	2			
P3	3	0	2	9	0	2			
P4	2	1	1	2	2	2			
P5	0	0	2	4	3	3			

Answer the following questions using the banker's algorithm:

- 1.What is the reference of the need matrix?
- 2.Determine if the system is safe or not.
- 3.What will happen if the resource request (1, 0, 0) for process P1 can the system accept this request immediately?

Ans. 1: Context of the need matrix is as follows:

Need [i] = Max [i] – Allocation/Has [i]

Need for P1: (7, 5, 3) - (0, 1, 0) = 7, 4, 3

Need for P2: (3, 2, 2) - (2, 0, 0) = 1, 2, 2

Need for P3: (9, 0, 2) - (3, 0, 2) = 6, 0, 0

Need for P4: (2, 2, 2) - (2, 1, 1) = 0, 1, 1

Need for P5: (4, 3, 3) - (0, 0, 2) = 4, 3, 1

Process	Need		
	A	B	C
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	1

Ans. 2: Apply the Banker's Algorithm: Available Resources of A, B and C are 3, 3, and 2. Now we check if each type of resource request is available for each process.

Step 1: For Process P1:

Need \leq Available

7, 4, 3 \leq 3, 3, 2 condition is **false**.

So, we examine another process, P2.

Step 2: For Process P2:

Need \leq Available

1, 2, 2 \leq 3, 3, 2 condition **true**

New available = available + Allocation

(3, 3, 2) + (2, 0, 0) \Rightarrow 5, 3, 2

Similarly, we examine another process P3.

Step 3: For Process P3:

$P3 \text{ Need} \leq \text{Available}$

$6, 0, 0 \leq 5, 3, 2$ condition is **false**.

Similarly, we examine another process, P4.

Step 4: For Process P4:

$P4 \text{ Need} \leq \text{Available}$

$0, 1, 1 \leq 5, 3, 2$ condition is **true**

$\text{New Available resource} = \text{Available} + \text{Allocation}$

$5, 3, 2 + 2, 1, 1 \Rightarrow 7, 4, 3$

Similarly, we examine another process P5.

Step 5: For Process P5:

$P5 \text{ Need} \leq \text{Available}$

$4, 3, 1 \leq 7, 4, 3$ condition is **true**

$\text{New available resource} = \text{Available} + \text{Allocation}$

$7, 4, 3 + 0, 0, 2 \Rightarrow 7, 4, 5$

Now, we again examine each type of resource request for processes P1 and P3.

Step 6: For Process P1:

P1 Need \leq Available

7, 4, 3 \leq 7, 4, 5 condition is **true**

New Available Resource = Available + Allocation

7, 4, 5 + 0, 1, 0 \Rightarrow 7, 5, 5

So, we examine another process P2.

Step 7: For Process P3:

P3 Need \leq Available

6, 0, 0 \leq 7, 5, 5 condition is true

New Available Resource = Available + Allocation

7, 5, 5 + 3, 0, 2 \Rightarrow 10, 5, 7

Hence, we execute the banker's algorithm to find the safe state and the safe sequence : P2, P4, P5, P1 and P3.

Ans. 3: For granting the Request (1, 0, 2), first we have to check that **Request** \leq **Available**, that is (1, 0, 2) \leq (3, 3, 2), since the condition is true. So the process P1 gets the request immediately.

Example :3

Consider the following snapshot of a system:

Processes	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P ₀	1	1	2	4	3	3	2	1	0
P ₁	2	1	2	3	2	2			
P ₂	4	0	1	9	0	2			
P ₃	0	2	0	7	5	3			
P ₄	1	1	2	1	1	2			

1. Calculate the content of the need matrix?
2. Is the system in a safe state?
3. Determine the total amount of resources of each type?

Ans. 1. Content of the need matrix can be calculated by using the below formula

$$\text{Need} = \text{Max} - \text{Allocation}$$

Process	Need		
	A	B	C
P ₀	3	2	1
P ₁	1	1	0
P ₂	5	0	1
P ₃	7	3	3
P ₄	0	0	0

2. Now, we check for a safe state

Safe sequence:

1. For process P_0 , Need = (3, 2, 1) and

$$\text{Available} = (2, 1, 0)$$

$$\text{Need} \leq \text{Available} = \text{False}$$

So, the system will move for the next process.

2. For Process P_1 , Need = (1, 1, 0)

$$\text{Available} = (2, 1, 0)$$

$$\text{Need} \leq \text{Available} = \text{True}$$

Request of P_1 is granted.

$$\therefore \text{Available} = \text{Available} + \text{Allocation}$$

$$= (2, 1, 0) + (2, 1, 2)$$

$$= (4, 2, 2) \text{ (New Available)}$$

3. For Process P_2 , Need = (5, 0, 1)

$$\text{Available} = (4, 2, 2)$$

$$\text{Need} \leq \text{Available} = \text{False}$$

So, the system will move to the next process.

4. For Process P_3 , Need = (7, 3, 3)

$$\text{Available} = (4, 2, 2)$$

$$\text{Need} \leq \text{Available} = \text{False}$$

So, the system will move to the next process.

5. For Process P_4 , Need = (0, 0, 0)

$$\text{Available} = (4, 2, 2)$$

$$\text{Need} \leq \text{Available} = \text{True}$$

Request of P_4 is granted.

$$\therefore \text{Available} = \text{Available} + \text{Allocation}$$

$$= (4, 2, 2) + (1, 1, 2)$$

$$= (5, 3, 4) \text{ now, (New Available)}$$

6. Now again check for Process P_2 , Need = (5, 0, 1)

$$\text{Available} = (5, 3, 4)$$

$$\text{Need} \leq \text{Available} = \text{True}$$

Request of P_2 is granted.

$$\therefore \text{Available} = \text{Available} + \text{Allocation}$$

$$= (5, 3, 4) + (4, 0, 1)$$

$$= (9, 3, 5) \text{ now, (New Available)}$$

7. Now again check for Process P_3 , Need = (7, 3, 3)

$$\text{Available} = (9, 3, 5)$$

$$\text{Need} \leq \text{Available} = \text{True}$$

Request of P_3 is granted.

$$\therefore \text{Available} = \text{Available} + \text{Allocation}$$

$$= (9, 3, 5) + (0, 2, 0) = (9, 5, 5)$$

8. Now again check for Process P_0 , = Need (3, 2, 1)

$$= \text{Available } (9, 5, 5)$$

$$\text{Need} \leq \text{Available} = \text{True}$$

So, the request will be granted to P_0 . Safe sequence: $\langle P_1, P_4, P_2, P_3, P_0 \rangle$

The system allocates all the needed resources to each process. So, we can say that system is in a safe state.

3. The total amount of resources = sum of columns of allocation + Available

$$= [8 \ 5 \ 7] + [2 \ 1 \ 0] = [10 \ 6 \ 7]$$