Deadlocks

CS 342 – Operating Systems

Ibrahim Korpeoglu
Bilkent University
Department of Computer Engineering

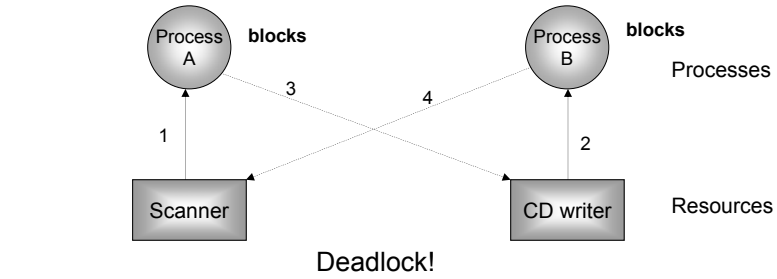
Deadlocks

- There are lots of resource that can be used by one process at a time:
  - Scanner, printer, tape drives, slots in process table, …
- OSs have the ability to grant (temporarily) a resource to a single process
  - Exclusive access.
- A process may need exclusive access to more than one resource.

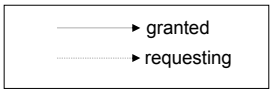
Example

A is programmed to request Scanner first and then CD Writer

B is programmed to request CD Writer first and then Scanner



Deadlock!



Different kind of Resources

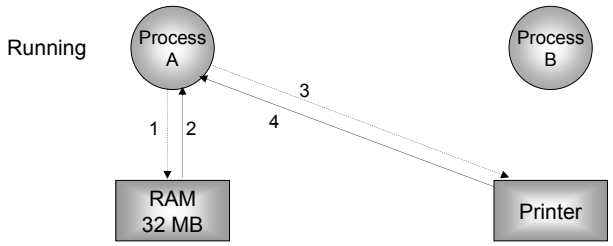
- I/O Devices
  - Scanner, plotter, printer, tape drive, …
  - These are examples of hardware resources
- Files
  - A file may need to be written by multiple processes
- Database records
  - Many processes can lock DB records and use them
  - These are examples of software resources.
- OS system tables
  - Process table, i-node table, …

Different kind of Resources

- A computer may have multiple instances of a resource type.
  - Multiple printers, tape drives, …
- We will abstract the *resources* as *objects* that may be requested and granted if available.

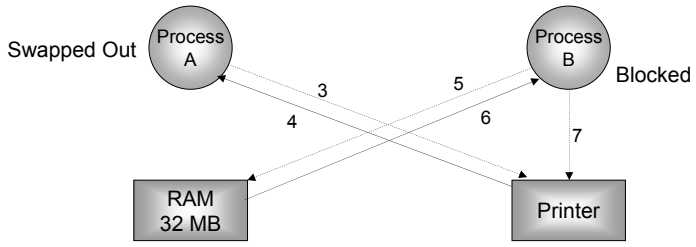
Preemptable resource - deadlock

- A preemptive resource is one that can be taken away from the process owning it with no ill affects.
  - Memory for example



Preemptable resource - deadlock

A gets swapped out and B is run now. A releases Memory and B uses it.

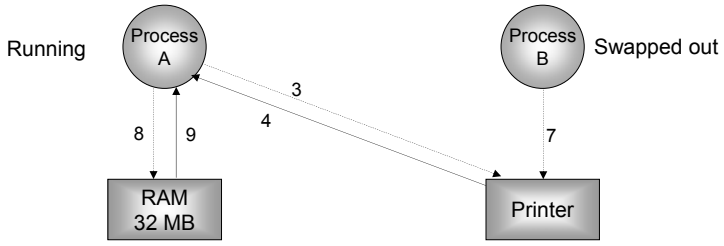


B requests printer and blocks.

DEADLOCK! A is swapped out, can not run!. B is blocked, can not run!

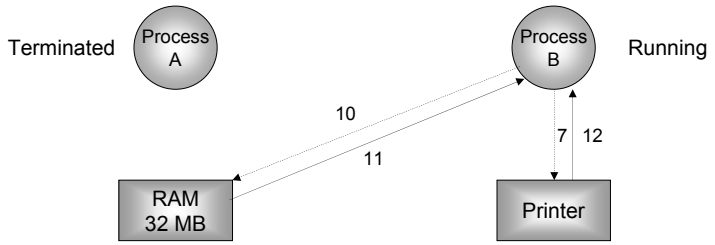
Preemptable resource - solution

Take memory from B, swap B out.
Run A until completion so that it can release printer.



Preemptable resource - solution

Take memory from B, swap B out.
Run A until completion so that it can release printer.



Nonpreemptable resources

- A nonpreemptable resource is one that can not be taken away from its current owner without causing the computation to fail.
  - A CD writer for example: while a process is using it, we can not give it some other process.
- We will be concerned most of the time with nonpreemptable resources.

How a resource is requested and allocated

- The sequence of events requires to use a resource is given below:
  - Request the resource.
  - Use the resource
  - Release the resource
- If request can not be granted:
  - The requesting process can block
  - The requesting process can be returned an error code and it can then try requesting the resource again.

How a resource is requested and allocated

- A resource could be request by one of the following ways:
  - Call a system call *request*
    - *Grants the request of resource is available*
    - *Blocks the process if resource is not available.*
  - Open a *special file* corresponding to the resource.
    - *Only one process can succeed opening the file.*
    - *Other process will but to sleep.*

Management of Resources

- Some resources are managed by OS
  - Like kernel tables, I/O devices, etc.
- Some resources may be managed by the user level processes themselves by use of semaphores.
  - Like database records.

Management of Resources at the Application level.

- A process calls *down*() system call on a semaphore before using a resource.
- If it success *down*() operation, it uses the resource.
- It calls *up*() operation after it has used the resource.
- Only one process can succeed the down() operation on a shared semaphore which is initialized to 1 (or mutex). The other will block until an up operation is called on the semaphore.
- A process can lock and use more than one resource.

Use of application level resources

```
semaphore resource1;

void processA (void)
{
        down(&resource1);
        use_resource1();
        up(&resource1);
}
```

```
semaphore resource1, resource2;

void processA (void)
{
        down(&resource1);
        down(&resource2);
        use_resource_1_and_2();
        up(&resource2);
        up(&resource1);
}
```


Use of application level resources

- Order of requesting and using the resource does matter!!!
- For example, in the figure 1 of next slide, we will not have any deadlock.
- But in the figure 2 of next slide, we may have a deadlock!

Use of application level resources

```
semaphore resource1;
Semaphore resource2;

void processA (void)
{
        down(&resource1);
        down(&resource2);
        use_resource_1_and_2();
        up(&resource2);
        up(&resource1);
}

void processB (void)
{
        down(&resource1);
        down(&resource2);
        use_resource_1_and_2();
        up(&resource2);
        up(&resource1);
}
```

Figure 1

```
semaphore resource1;
Semaphore resource2;

void processA (void)
{
        down(&resource1);
        down(&resource2);
        use_resource_1_and_2();
        up(&resource2);
        up(&resource1);
}

void processB (void)
{
        down(&resource2);
        down(&resource1);
        use_resource_1_and_2();
        up(&resource1);
        up(&resource2);
}
```

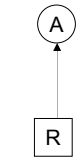
Figure 2

Formal Definition of Deadlock

- *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.*
- Conditions for Deadlock
  - 4 conditions must hold for there to be deadlock:
  1. *Mutual Exclusion*: A resource could be assigned to exactly one process at any given time.
  2. *Hold and Wait*: processing holding resources can request new resources and wait for them.
  3. *No preemption*: a resource can not be taken forcibly from a holding process
  4. *Circular wait*: there must be chain of processes, each of which is waiting for a resource held by the next member of the chain.

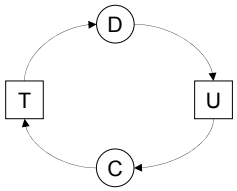
Deadlock Modeling

- Deadlock can be modeled using *directed graphs*.
  - Processes: circles
  - Resources: rectangles (or squares)
  - Arcs: *holding* or *requesting* relationships.



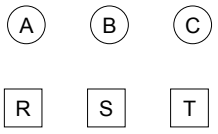
A holding resource R

B is requesting resource S



A deadlock situation
Cycle: C-T-D-U-C

Example-1



Assume, we have 3 processes and 3 different resources.

A	B	C
Request R	Request S	Request T
Request S	Request T	Request R
Release R	Release S	Release T
Release S	Release T	Release R

Sequence of operations that each process performs.

Assume

A executes fist.
B executes after A finishes.
C executes after B finishes.

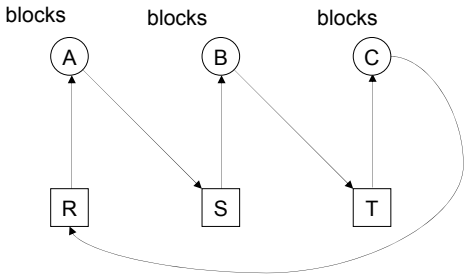
This execution sequence does not cause any deadlock!

Example-2

Lets execute the processes and their operation
in a different order as shown below:

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

Deadlock!



Use of resource graphs for deadlocks detection

- We should execute requests and releases step by step
- Each step we check if the resource graph has a cycle.
  - If it has, deadlock occurs.
  - Otherwise, no deadlock so far.

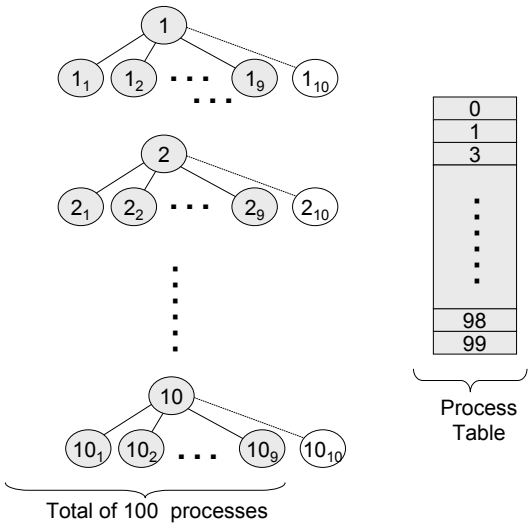
Strategies to deal with deadlocks

- Four strategies are used to deal with deadlocks:
  - Ignore the problem
    - Useful for rare deadlock cases.
  - Detection and recovery
    - Let deadlocks occur, detect them, and take action.
  - Dynamic avoidance
    - By careful resource allocation.
  - Prevention
    - By negating one of the four conditions to have deadlocks.

The Ostrich Algorithm

- Stick your head in the sand and pretend that there is no problem at all.
- This is useful in cases where occurrence of the deadlock situation is very rare.
- Example:
  - Creating new processes using fork() using a finite slot process table.
  - Fork() fails if there is no empty slot in the process table for the child.

- Example continues:
  - Assume we have a process table of size 100.
  - 10 programs are running, each needing to create 12 children.
  - After each has created 9 children, total number of process is 10 + 10*9 = 100 (table filled).
  - Now, assume each process is trying creating a new child using fork(), but fork() fails and each process tries again some time later in a loop.
    - ➔DEADLOCK



Each of the 10 parents is trying repeatedly to create the 10th child, but can not be successful.

Deadlock Detection and Recovery

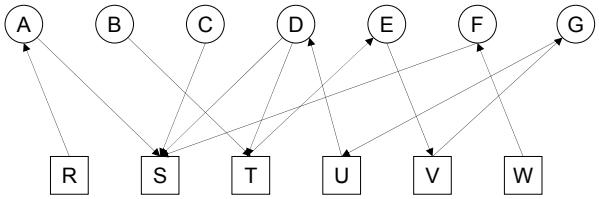
- System lets the deadlocks occur.
- Tries to detect the deadlocks when they occur
- Takes action to recover from deadlock.
- We will see different algorithms of deadlocks detection for:
  - Only one resource of each type exists.
  - Multiple copies of each type of resource may exist

Deadlock Detection: One Resource of Each Type

- Construct the resource graph.
- If the graph contains one or more cycles then a deadlock exists:
  - All the processes on the cycle are deadlocked.
- If no cycles exist, the system is not deadlocked.

Example

1. Process A holds R and wants S
2. Process B holds nothing but wants S.
3. Process C holds nothing but wants S
4. Process D holds U and wants S and T
5. Process E hold T and wanst V
6. Process F holds W and wants S
7. Process G holds V and wants U.



Cycle: T – E – V – G – U – D - T

Detecting Cycles in Directed Graphs

1. For each node N in the graph, perform the following 5 steps with N as the starting node.
2. Initialize L to be empty list and designate all the arcs as unmarked
3. Add the current node to the end of L and check to see if the node now appears in L twp times. If it does, the graph contains a cycle and the algorithm terminates.
4. From the given node, see if there are any unmarked outgoing arcs. If so, go to step 5; if not, go to step 6.
5. Pick an unmarked outgoing arc at random and mark it. Then follow it to the new current node and go to step 3.
6. We have now reached a dead end. Remove it and go back to the previous node, that is, the one that was current just before this one, make that one the current node, and go to step 4. If this node is the initial node, the raph does not contain any cycles and the algorithm terminates.

Detecting Cycles in Directed Graphs

- We keep a list L and update it - insert and remove nodes from resource graph.
- Whe check cycles in list L.
- Initially the list is empty.
- Initially all edges of resource graph is *unmarked*.
- Algorithm *terminates*:
  - If we detect a cycle
  - If no cycles is detected and all edges are *marked*.

Example run of algorithm - 1

Resource Graph

Node	Next Node(s)
A	S
B	T
C	S
D	S, T
E	V
F	S
G	U
R	A
S	
T	E
U	D
V	G
W	F

Alg Step	**process**	**İnit node**	**List L**
1		A	
2	Make L empty	A	empty
3	Add current node to the end of list	A	**A**
4	Is there unmarked edge – Y	A	
5	Pick unmarked and mark it, follow next node and make it current	A	
3	Add the current node to end of list	A	A **S**
4	Is there unmarked edge – N	A	
6	Remove current. Reached inıt node and all marked? Y – stop	A	**A**
		A	
		A	
		A	
		A	

Example run of algorithm - 2

Resource Graph

Node	Next Node(s)
A	S
B	T
C	S
D	S, T
E	V
F	S
G	U
R	A
S	
T	E
U	D
V	G
W	F

Alg Step	process	İnit node	List L
1		B	
2	Make L empty	B	empty
3	Add current node to the end of list	B	**B**
4	Is there unmarked edge – Y	B	
5	Pick unmarked and mark it, follow next node and make it current	B	B
3	Add the current node to end of list	B	B **T**
4	Is there unmarked edge – Y	B	
5	Pick unmarked and mark it, follow next node and make it current	B	B T
3	Add current node to the end of list	B	B T **E**
4	Is there unmarked edge – Y	B	
5	Pick unmarked and mark it, follow next node and make it current	B	
3	Add the current node to end of list	B	B T E **V**

Example run of algorithm - 3

Resource Graph

Node	Next Node(s)
A	S
B	T
C	S
D	S, T
E	V
F	S
G	U
R	A
S	
T	E
U	D
V	G
W	F

Alg Step	process	İnit node	List L
4	Is there unmarked edge – Y	B	
5	Pick unmarked and mark it, follow next node and make it current	B	
3	Add current node to the end of list	B	B T E V **G**
4	Is there unmarked edge – Y	B	
5	Pick unmarked and mark it, follow next node and make it current	B	
3	Add the current node to end of list	B	B T E V G **U**
4	Is there unmarked edge – Y	B	
5	Pick unmarked and mark it, follow next node and make it current	B	
3	Add the current node to end of list	B	B T E V G U **D**

Example run of algorithm - 4

Resource Graph

Node	Next Node(s)
A	S
B	T
C	S
D	S, T
E	V
F	S
G	U
R	A
S	
T	E
U	D
V	G
W	F

Alg Step	process	İnit node	List L
4	Is there unmarked edge – Y	B	
5	Pick unmarked and mark it, follow next node and make it current	B	
3	Add current node to the end of list	B	B T E V G U D **S**
4	Is there unmarked edge – N	B	
6	Remove current. Reached inıt node and all marked? N – go to step 4	B	B T E V G U **D**
4	Is there unmarked edge – Y	B	
5	Pick unmarked and mark it, follow next node and make it current	B	
3	Add current node to the end of list – **Node T appears two times. Stop- Deadlock detected. End of Algorithm**	B	B T E V G U D **T**

Deadlock Detection with Multiple Resources of Each Type

- We will present a different algorithm.
- We wil use matrices to expresse resource types and their counts, their avaliability and thier allocation

Deadlock Detection with Multiple Resources of Each Type

Assume we have *n* processes, P_1 through P_n.
Assume we have *m* different type of resources.

$E = (E_1, E_2, ..., E_m)$: Resources in existence

$A = (A_1, A_2, ..., A_m)$: Resources available

$$C = \begin{bmatrix} C_{11} & C_{12} & \dots & C_{1m} \\ C_{21} & C_{22} & \dots & C_{2m} \\ \dots & \dots & \dots & \dots \\ C_{n1} & C_{n2} & \dots & C_{nm} \end{bmatrix} \quad R = \begin{bmatrix} R_{11} & R_{12} & \dots & R_{1m} \\ R_{21} & R_{22} & \dots & R_{2m} \\ \dots & \dots & \dots & \dots \\ R_{n1} & R_{n2} & \dots & R_{nm} \end{bmatrix}$$

$$\sum_{i=1}^{n} Cij + Aj = Ej$$

C : Current allocation matrix

R : Request matrix

Deadlock Detection with Multiple Resources of Each Type

- Definition
  - Given two vectors A and B, we say A ≤ B if each element of A is smaller or equal to the corresponding element of B.

$$A \le B \text{ if and only if } A_i \le B_i \text{ for } 1 \le i \le m.$$

Each process is initially unmarked. As the algorithm progresses, Processes will be marked if they will be able to comlete without deadlock. At the end, all the unmarked processes are deadlocked.

Deadlock Detection with Multiple Resources of Each Type

Algorithm

1. Look for an unmarked process P_i, for which the i^{th} row of R is less than or equal to A.
2. If such a process is found, add the i^{th} row of C to A, mark the process and go back to step 1.
3. If no such process exists, the algorithm terminates.

Deadlock Detection with Multiple Resources of Each Type

Example: There are 3 processes,
4 resource types.

$$E = (4,2,3,1)$$

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

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

Deadlock Detection with Multiple Resources of Each Type

Below X_i means the i^{th} row of matrix X.

Compare R_1 with A! R_1 is not smaller or equal to A. So it can not be satisfied.
Compare R_2 with A! R_2 is not smaller or equal to A. So it can not be satisfied.
Compare R_3 with A! R_3 is smaller or equal to A. So it *can be satisfied*.
Release resource of process 3. A = A + C_3, A = (2, 2, 2, 0)
Mark process 3.
Compare R_1 with A! R_1 is not smaller or equal to A. So it can not be satisfied.
Compare R_2 with A! R_2 is smaller or equal to A. So it *can be satisfied*.
Release resource of process 2. A = A + C_2, A = (4, 2, 2, 1)
Mark process 2.
Compare R_1 with A! R_1 is smaller or equal to A. So it *can be satisfied*.
Release resource of process 1. A = A + C_1, A = (4, 2, 3, 1)
Mark process 2.

All processes are marked. There is no deadlock.

Deadlock Recovery

- Recovery through preemption
  - Temporarily take the resource from the current owner and give it to another process.
    - Example: laser printer can be taken away in some cases
- Recover through rollback
  - Processes are check pointed periodically.
    - State of process is written to a file including the resource allocation state.
  - Process can be restarted later starting from that checkpoint.

Deadlock Recovery

- Recovery through killing the process
  - Kill one or more processes
  - A process in the cycle can be chosen as the victim.
  - Restart the process
  - OK for some applications such as compiling
  - Not OK for some other applications: database record updates.

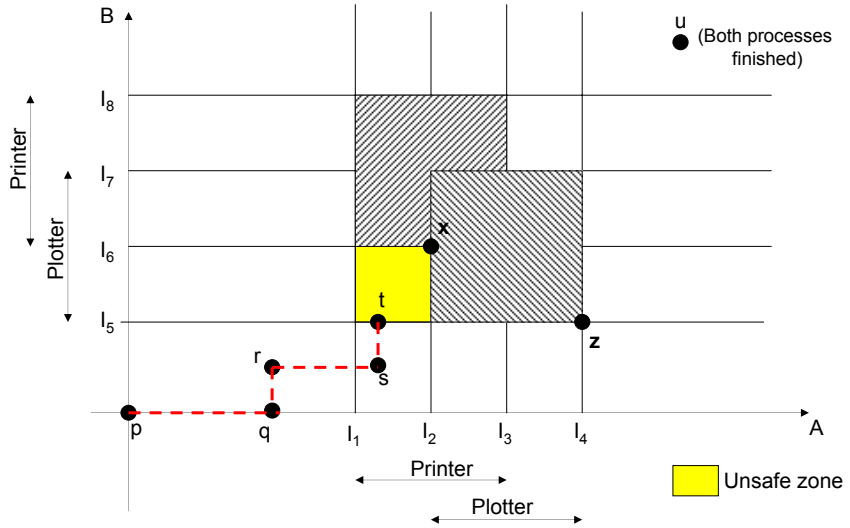
Deadlock Avoidance

- An other method to deal with deadlocks is Avoiding them!
- When a request for a resource is made:
  - Check if granting the resource may lead to deadlocks.
  - If it may lead to deadlock, do not grant the resource.
    - Unsafe state
  - If it is guaranteed that it will not lead to deadlock, give the resource.
    - Safe state

Deadlock Avoidance

- Various algorithms to avoid deadlocks
  - Resource Trajectories
  - Safe and Unsafe States
  - The Banker's Algorithm for a Single Resource
  - The Banker's Algorithm for Multiple Resources

Resource Trajectories



Resource Trajectories

- Shaded zones can not be entered.
- At point t, process B requests plotter.
- If plotter is granted to B at point t, we will enter an unsafe zone (yellow).
- After entering unsafe zone, we will reach finally to point x, where deadlock will occur.
- In order to avoid deadlock:
  - B's request for plotter at point t should not be granted.
  - Instead A should be started running and it should run until point z.

Safe and Unsafe States

- From resource allocation perspective, at any given time, the system state will be consisting of value of matrices E, A, R, C and R:
  - E: existing resource vector
  - A: available resource vector
  - C: current resource allocation matrix
  - R: request matrix

Safe and Unsafe States

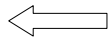
- A state is said to be safe if:
  - It is not deadlocked, and
  - There is some scheduling order for processes that guarantees every process to finish even though they request their maximum number resources suddenly.

Safe and Unsafe States

Assume one resource. We have 10 instances of that resource.

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3



Is this state safe?

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

	Has	Max
A	3	9
B	0	-
C	2	7

Free: 5

	Has	Max
A	3	9
B	0	-
C	7	7

Free: 0

	Has	Max
A	3	9
B	0	-
C	0	-

Free: 7

	Has	Max
A	9	9
B	0	-
C	0	-

Free: 1

	Has	Max
A	0	-
B	0	-
C	0	-

Free: 10

All processes can terminate!.
That state was safe!

Safe and Unsafe States

A	3	9
B	2	4
C	2	7

Free: 3

Current State

A requests an other resource instance.
Should we grant the Resource Instance?

If we would grant one more resource to A,
the state would be as follows:

A	4	9
B	2	4
C	2	7

Free: 2

New State

Check if the new state is safe!

Safe and Unsafe States

A	4	9
B	2	4
C	2	7

Free: 2

Run B

A	4	9
B	4	4
C	2	7

Free: 0

B finished

A	4	9
B	0	-
C	2	7

Free: 4

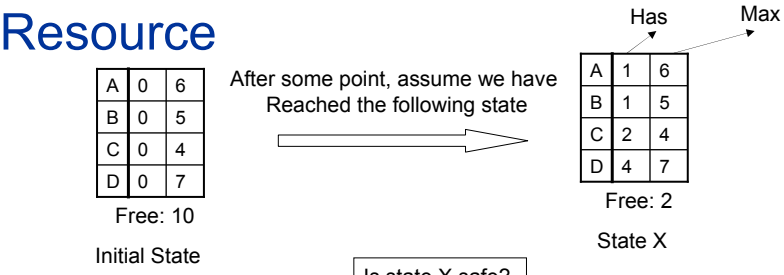
We can not run any other process!
All processes need more than 4 resouce
Instances to reach the maxiumum.

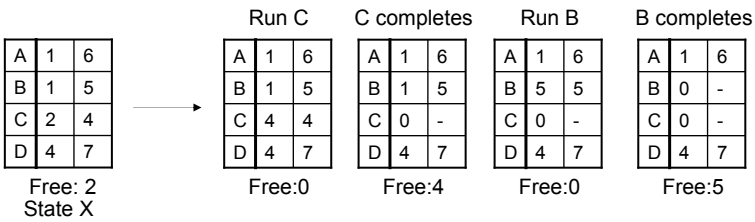
Therefore that state was not safe.
We should not grant one more resource to A!

Banker's Algorithm for a Single Resource

- When a process request a resource
  - Check if granting the resource leads to a safe state.
  - If it does grant the request
  - Otherwise, deny the request.

Banker's Algorithm for a Single Resource





Banker's Algorithm for a Single Resource

Run D			D finishes			Run A			A finishes		
A	1	6	A	1	6	A	6	6	A	0	-
B	0	-	B	0	-	B	0	-	B	0	-
C	0	-	C	0	-	C	0	-	C	0	-
D	7	7	D	0	-	D	0	-	D	0	-
Free:2			Free:9			Free:4			Free:10		

All processes terminate. The state X was safe.

Banker's Algorithm for a Single Resource

Let say we were at state X and B request one more resource.

A	1	6
B	1	5
C	2	4
D	4	7
Free: 2		
State X		

If we grant one more resource to B, We reach state Y.

A	1	6
B	2	5
C	2	4
D	4	7
Free: 1		
State Y		

Is state Y safe?

No, since none of the processes can run until completion if they request their maximum resource usage. We have 1 free resource. It is not enough. Therefore, state Y is not safe, and we should not grant one more resource to B at state X: it can lead to deadlock!

Banker's Algorithm for Multiple Resources

Process	Tape drives	Plotters	Scanner	CD Writes
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	Scanner	CD Writes
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 needed

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

E: Resources in existence
P: Possessed resources
A: Available resources

E = P + A

Algorithm to check if a state is safe

1. Look for a row in matrix R, whose unmet resource needs are all smaller than or equal to A.

   If no such row exists (no process exists), the system may eventually deadlock since no process can run completion.

2. Assume the process of the row chosen requests all the resources it needs and finishes. Mark that process as terminated and add all its resources to vector A.

3. Repeat steps 1 and 2
   - Until all processes are *marked terminated*, in which case the initial state was safe, or
   - until a deadlock occurs, in which case the initial state was not safe

Deadlock Prevention

- Attack the four conditions
  - Make sure that at least one of these conditions is never satisfied.
- Attacking
  - Mutual Exclusion Condition
  - Attacking the hold and wait condition
  - Attacking the no preemption condition.
  - Attacking the circular wait condition.