Chapter 8: Deadlocks

-
-
- **Calculary Constrainers Constrainers Constrainers (COVID)**
 Example 2
 **Examp Carline

1988 - System Model

1998 - Deadlock Characterization

1998 - Methods for Handling Deadlocks

1998 - Deadlock Prevention Cutline

• System Model

• Deadlock Characterization

• Methods for Handling Deadlocks

• Deadlock Prevention

• Deadlock Avoidance Cutline

System Model

- Beadlock Characterization

- Methods for Handling Deadlocks

- Deadlock Prevention

- Deadlock Avoidance

- Deadlock Detection Cutline

•** System Model

• Deadlock Characterization

• Methods for Handling Deadlocks

• Deadlock Avoidance

• Deadlock Detection

• Recovery from Deadlock **Cutline

•** System Model

• Deadlock Characterization

• Methods for Handling Deadlocks

• Deadlock Prevention

• Deadlock Avoidance

• Deadlock Detection

• Recovery from Deadlock **Example 18 System Model

• Deadlock Characterization

• Methods for Handling Deadlocks

• Deadlock Avoidance

• Deadlock Avoidance

• Deadlock Detection

• Recovery from Deadlock**
-
-
-
-

Chapter Objectives

-
- **Chapter Objectives**

 Illustrate how deadlock can occur when mutex locks are used

 Define the four necessary conditions that characterize deadlock

 Identify a deadlock situation in a resource allocation graph **Chapter Objectives**

• Illustrate how deadlock can occur when mutex locks are used

• Define the four necessary conditions that characterize deadlock

• Identify a deadlock situation in a resource allocation graph

• Eval
-
- **Chapter Objectives**

 Illustrate how deadlock can occur when mutex locks are used

 Define the four necessary conditions that characterize deadlock

 Identify a deadlock situation in a resource allocation graph

 Eval **Chapter Objectives**

• Illustrate how deadlock can occur when mutex locks are used

• Define the four necessary conditions that characterize deadlock

• Identify a deadlock situation in a resource allocation graph

• Eval **Chapter Objectives**

• Illustrate how deadlock can occur when mutex locks are used

• Define the four necessary conditions that characterize deadlock

• Identify a deadlock situation in a resource allocation graph

• Eval **Chapter Objectives**

• Illustrate how deadlock can occur when mutex locks are used

• Define the four necessary conditions that characterize deadlock

• Identify a deadlock situation in a resource allocation graph

• Eval **Example 1:**

Illustrate how deadlock can occur when mutex locks are used

• Define the four necessary conditions that characterize deadlock

• Identify a deadlock situation in a resource allocation graph

• Evaluate the f
-
-
- Apply the deadlock detection algorithm

 Evaluate approaches for recovering from deadlock

Deperating System Concepts 10th Edition 8.3 Silberschatz, Galvin and Gagne ©2018

Deadlock

- □ Peadlock
□ Permanent blocking of a set of processes that either compete for
□ system resources or communicate with each other
□ A set of blocked processes each holding a resource and waiting
- **Deadlock**
Permanent blocking of a set of processes that either compete for
system resources or communicate with each other
A set of blocked processes each holding a resource and waiting
to acquire a resource held by anoth □ A set of processes that either compete for

■ Permanent blocking of a set of processes that either compete for

■ A set of blocked processes each holding a resource and waiting

■ to acquire a resource held by another p **Deadlock**
 Permanent blocking of a set of processes that either compete for

system resources or communicate with each other

A set of blocked processes each holding a resource and waiting

to acquire a resource held by **Deadlock**

■ Permanent blocking of a set of processes that eit

system resources or communicate with each othe

■ A set of blocked processes each holding a resource a resource held by another process in

■ No efficient s **Deadlock**

• Permanent blocking of a set of processes that either compete for

system resources or communicate with each other

• A set of blocked processes each holding a resource and waiting

⊙ acquire a resource held **Deadlock**
 Permanent blocking of a set of processes that either compete for

system resources or communicate with each other

A set of blocked processes each holding a resource and waiting

co acquire a resource held b
- -
	-

Deadlock

Figure 6.1 Illustration of Deadlock

Deadlock

System Model **System Model**

• System consists of resources

• Resource types R_1, R_2, \ldots, R_m

• CPU cycles, memory space, I/O devices $\begin{array}{l} \mathbf{System\ Model} \ \end{array}$ ources $_{n \times N_\mathrm{m}}$ ory space, I/O devices
has W_i instances.

-
- Resource types R_1, R_2, \ldots, R_m
- **System Model**

 System consists of resources

 Resource types R_1, R_2, \ldots, R_m

 CPU cycles, memory space, I/O devices

 Each resource type R_i has W_i instances. **• System Model**

vstem consists of resources

esource types R_1, R_2, \ldots, R_m

• CPU cycles, memory space, I/O devices

ach resource type R_i has W_i instances.

ach process utilizes a resource as follows:
- Each resource type R_i has W_i instances.
- **System Model**

 System consists of resources

 Resource types R_1, R_2, \ldots, R_m

 CPU cycles, memory space, I/O devices

 Each resource type R_i has W_i instances.

 Each process utilizes a resource as follows:

	- request
- Operating System Concepts 10th Edition 8.7 Silberschatz, Galvin and Gagne ©2018

Operating System Concepts 10th Edition 8.7 Silberschatz, Galvin and Gagne ©2018 **System Model**

• System consists of resources

• Resource types $R_1, R_2, ..., R_m$

• CPU cycles, memory space, I/O devices

• Each resource type R_i has W_i instances.

• Each process utilizes a resource as follows:

• **re IFFORM STATE THE REPART CONSIST CONSISTED THE REPU Cycles, memory space, I/O devices**
 IFFORM Cycles, memory space, I/O devices
 IFFORM Cycles, memory space, I/O devices
 IFFORM CYCLES
 IFFORM CONSISTED ASSEMS
 consists of resources
 $\sum_{n=1}^{\infty} P_n$
 $\sum_{n=1}^{\infty} P_n$
 $\sum_{n=1}^{\infty} P_n$, $\sum_{n=1}^{\infty} P_n$, $\sum_{n=1}^{\infty} P_n$, $\sum_{n=1}^{\infty} P_n$
 $\sum_{n=1}^{\infty} P_n$

	- use
	- release

Deadlock with Semaphores **Deadlock with Semaphores**

ata:

• A semaphore s1 initialized to 1

• A semaphore s2 initialized to 1

wo processes P1 and P2 **Deadlock with Semaphores**

ata:

• A semaphore s1 initialized to 1

• A semaphore s2 initialized to 1

• A semaphore s2 initialized to 1

• A semaphore s2 initialized to 1

1: <p>• Data:</p>\n\n• A semaphore s1 initialized to 1\n• A semaphore s2 initialized to 1\n• Two processes P1 and P2\n• pi:\n  wait(s1)\n

- Data:
	-
	-
-
- \blacksquare P1:

wait(s1)

wait(s2)

▪ P2:

wait(s2)

wait(s1)

Necessary Conditions for Deadlock

can arise if four conditions hold simultaneously (*Coffman conditions*): **CONDUCT MECESSARY CONDUCTS FOR DEADLOCK**
Deadlock can arise if four conditions hold simultaneously (*Coffman conditions*):
1. Mutual exclusion: only one process at a time can use a resource **1. Mecessary Conditions for Deadlock**

Mecessary Conditions hold simultaneously (Coffman conditions):

1. Mutual exclusion: only one process at a time can use a resource

2. Hold and wait: a process holding at least one r **2. Hold and wait:** a process hold simultaneously (Coffman conditions):
2. Hold and wait: a process holding at least one resource is waiting to
acquire additional resources held by other processes Mecessary Conditions for Deadlock

Headlock can arise if four conditions hold simultaneously (Coffman conditions):

1. Mutual exclusion: only one process at a time can use a resource

2. Hold and wait: a process holding at **Example 2. Necessary Conditions for Deadlock**

1. Mutual exclusion: only one process at a time can use a resource

2. Hold and wait: a process holding at least one resource is waiting to

acquire additional resources held

Process ary Conditions for Deadlock

Metadlock can arise if four conditions hold simultaneously (*Coffman conditions*):

1. Mutual exclusion: only one process at a time can use a resource

2. Hold and wait: a process hol

Hold and wait: a process holding at least o
quire additional resources held by other process holding it, after that process has comp
The first three conditions are necessary but
to exist

process holding it, after that process has completed its task

• The first three conditions are necessary but not sufficient for a deadlock

to exist

• Concepts – 10th Edition

8.9 Silberschatz, Galvin and Gagne ©2018

• 1. Mutual exclusion: only one process at a time can use a resource
2. Hold and wait: a process holding at least one resource is waiting to
acquire additional resources held by other processes
3. No preemption: a resource c

Necessary Conditions for Deadlock
Can arise if four conditions hold simultaneously (*Coffman conditions*): **Deadlock Can arise if four conditions for Deadlock**
Deadlock can arise if four conditions hold simultaneously (*Coffman conditions*):
4. Circular wait: there exists a set $\{P_0, P_1, ..., P_n\}$ of waiting processes such that **1. Circular wait:** there exists a set $\{P_0, P_1, ..., P_n\}$ of waiting processes such that 4. Circular wait: there exists a set $\{P_0, P_1, ..., P_n\}$ of waiting processes such **pind for Deadlock**

simultaneously (*Coffman conditions*):
, …, $P_{\sf n}$ } of waiting processes such } of waiting processes such that **Necessary Conditions for Deadlock**

Exam arise if four conditions hold simultaneously (Coffman conditions):

In vait: there exists a set $\{P_0, P_1, ..., P_n\}$ of waiting processes such

is waiting for a resource that is held **Necessary Conditions for Deadlock**
an arise if four conditions hold simultaneously (*Coffman conditions*):
r wait: there exists a set $\{P_0, P_1, ..., P_n\}$ of waiting processes such
is waiting for a resource that is held by **• Necessary Conditions for Deadlock**

lock can arise if four conditions hold simultaneously (*Coffman conditions*):

rcular wait: there exists a set $\{P_0, P_1, ..., P_n\}$ of waiting processes such

• P_0 is waiting for a rise if four conditions hold simultaneously (*Coffman conditions*):
it: there exists a set $\{P_0, P_1, ..., P_n\}$ of waiting processes such
aiting for a resource that is held by P_1 ,
is waiting for a resource that is held b **4. Circular wait:** there exists a set $\{P_0, P_1, ..., P_n\}$ of waiting processes such that
 4. Circular wait: there exists a set $\{P_0, P_1, ..., P_n\}$ of waiting processes such

that
 9. P_0 is waiting for a resource that

- P_0 is waiting for a resource that is held by P_1 , $,$
- P_1 is waiting for a resource that is held by P_2 , ,
- ,
- and P_n is waiting for a resource that is held by P_0 . .
- ..., P_{n-1} is waiting for a resource that is held by P_n .

 and P_n is waiting for a resource that is held by P_0 .

 The first three conditions are necessary but not sufficient for a deadlock

to exist

 For **EXECUTE THE CONSERVANCE SETTLE CONSERVISTION**
 t
 e P_0 is waiting for a resource that is held I
 e \ldots , P_{n-1} is waiting for a resource that is
 and P_n is waiting for a resource that is F

The first thr \n P₀ is waiting for a resource that is held by P₁,\n P₁ is waiting for a resource that is held by P₂,\n ..., P_{n-1} is waiting for a resource that is held by P_n,\n and P_n is waiting for a resource that is held by P₀.\n The first three conditions are necessary but not sufficient for a deadlock to exist\n For deadlock to actually take place, the fourth condition is required\n
	-

-
- **A directed graph consistsof: A set of vertices V and a set of edges E.**

 V is partitioned into two types:

 $P = \{P_1, P_2, ..., P_n\}$, the set consisting of all the processes in the system • $P = \{P_1, P_2, ..., P_n\}$, the set consisting of all the processes in the system
	- **Resource-Allocation (**
 Resource-Allocation (
 P = { P_1 , P_2 , ..., P_n }, the set consisting of all the system

	 $R = \{R_1, R_2, ..., R_m\}$, the set consisting of all resystem • $R = \{R_1, R_2, ..., R_m\}$, the set consisting of all resource types in the **SOUITCE-Allocation Graph**

	Insists of: A set of vertices V and a set of edges E.

	..., P_n , the set consisting of all the processes in the

	, ..., R_m , the set consisting of all resource types in the system directed graph consists of: A set of vertices V and a set of edges E.

	• V is partitioned into two types:

	• $P = \{P_1, P_2, ..., P_n\}$, the set consisting of all the processes in the system

	• $R = \{R_1, R_2, ..., R_m\}$, the set consi cted graph consists of: A set of vertices V and a set of edges E.

	is partitioned into two types:

	• $P = \{P_1, P_2, ..., P_n\}$, the set consisting of all the processes in the

	system

	• $R = \{R_1, R_2, ..., R_m\}$, the set consisting o
	- -
		-

-
-
-
-
- T_1 holds one instance of R_2 and is waiting for an
- T_2 holds one instance of R_1 , one instance of R_2 , and is waiting for an instance of R_3
- T_3 is holds one instance of R_3

Deadlock?

Resource Allocation Graph

R₁ R₃

Deadlock?

Resource Allocation Graph

Deadlock?

Basic Facts

-
-
- **Basic Facts**
 EXECUSE EXECUS
 EXECUSE If graph contains no cycles \Rightarrow **no deadlock**
 EXECUSE If graph contains a cycle \Rightarrow
 EXECUSE IF graph contains a cycle \Rightarrow
 EXECUSE IF graph contains a cycle \Rightarrow
- **Basic Facts**

 If graph contains no cycles \Rightarrow no deadlock

 If graph contains a cycle \Rightarrow

 if only **one instance** per resource type, then deadlock

 if several instances per resource type, **possibility** of deadlo **Basic Facts**
 Facts
 Sasic Facts
 Sasic Facts
 Sasic Facts
 Sasic Spanned Sasisfy Algermance per resource type, then deadlock
 • if several instances per resource type, **possibility** of deadlock **Basic Facts**
 EXECUTE:
 EXECUTE:
 EXECUTE:

• if only one instance per resource type, then deadlock

• if several instances per resource type, possibility of deadlock

Methods for Handling Deadlocks **• Methods for Handling Deadlocks**
• Ignore the problem and pretend that deadlocks never
• Used by most operating systems, including UNIX, Windows
• Used by most operating systems, including UNIX, Windows **Methods for Handling Dea • Methods for Handling Deadlocks**
• Ignore the problem and pretend that deadlocks never
• Used by most operating systems, including UNIX, Windows
• Used by most operating systems, including UNIX, Windows

- **Ostrich**
	-
	-
- -
- re the problem and pretend that deadlocks **never**
or in the system
d by most operating systems, including UNIX, Windows
at the system will **never** enter a deadlock state:
dlock **prevention**
A set of methods for ensuring th
	- Deadlock avoidance
		- \rightarrow Tries to avoid deadlock by delaying the requests which may result in a deadlock.
- ▶ Requires additional information concerning which resources a process will Operating System Concepts – 10th Edition

Operating S Sure that the system will **never** enter a deadlock state:

→ Deadlock **prevention**

→ A set of methods for ensuring that at least one of the necessary conditions

→ Tries to avoid deadlock by delaying the requests which
	- -

Deadlock Prevention

Ne four necessary conditions for deadlock:

- **INCREDIE IN A SET AND READ TO A SET AND READ TO A SET A SET AND READ TO A SET A SET AND READ TO A SET A SET AND READ THE SOLUT SHOW THE SHOW Deadlock Prevention

Invalidate one of the four necessary conditions for deadlock:**

• Mutual Exclusion – not required for sharable resources (e.g., read-only files); must hold for non-sharable resources

• In general, th **Example 19 Sean Accepts Accepts Accepts Accepts Accepts Accepts**
Files); must hold for non-sharable resources (e.g., read-only
files); must hold for non-sharable resources
• In general, the first of the four listed condit **Prevention**

• Idate one of the four necessary conditions for deadlock:
 utual Exclusion – not required for sharable resources (e.g., read-only

• In general, the first of the four listed conditions cannot be disallowed **Prevention**

• Idate one of the four necessary conditions for deadlock:
 utual Exclusion – not required for sharable resources (e.g., read-only

• In general, the first of the four listed conditions cannot be disallowed **Deadlock Prevention**
te one of the four necessary conditions for deadlock:
 all Exclusion – not required for sharable resources (e.g., read-only

i; must hold for non-sharable resources

In general, the first of the fou <table>\n<tbody>\n<tr>\n<th>■</th>\n<th>■</th>\n<th>■</th>\n<th>■</th>\n<th>■</th>\n</tr>\n<tr>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n</tr>\n<tr>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n</tr>\n<tr>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n<td>1</td>\n<td> idate one of the four necessary conditions for deadlock:
 utual Exclusion – not required for sharable resources (e.g., read-only
 es); must hold for non-sharable resources

• In general, the first of the four listed co
	-
	- it one of the four necessary conditions for deadlock:
 aal Exclusion not required for sharable resources (e.g.

	i; must hold for non-sharable resources

	In general, the first of the four listed conditions cannot b

	If

■ Hold and Wait

■ must guarantee that whenever a process requests a resource, it

does not hold any other resources

Silberschatz, Galvin and Gagne ©2018

Doperating System Concepts – 10th Edition

8.17 Silberschatz, G

Deadlock Prevention

- □ Deadlock Prevention
■ Hold and Wait (Cont.)
■ 1. Require a process request all of its required reso □ **Deadlock Prevention**

→ 1. Require a process request all of its required resources at one

time (e.g., at the beginning) **Deadlock Prevention
Id and Wait (Cont.)**
1. Require a process request all of its required resourc
time (e.g., at the beginning)
2. Allow process to request resources only when the pr **Deadlock Prevention**

■ 1. Require a process request all of its required resources at one

time (e.g., at the beginning)

■ 2. Allow process to request resources only when the process has none

■ i.e., before a process c ■ **Deadlock Prevention**
 Example 3 is added
 Example: Allow process request all of its required resources at one

ime (e.g., at the beginning)
 Example: Allow process to request resources only when the process has n and Wait (Cont.)
Require a process request all of its required resources at one
ne (e.g., at the beginning)
Allow process to request resources only when the process has none
i.e., before a process can request any additiona 1. Require a process request all of its required resoutime (e.g., at the beginning)

2. Allow process to request resources only when the

■ i.e., before a process can request any additional resour-

all the resources that
	-
- Example: The Depinning

Starvation possible: a process to request resources only when the process has none

 i.e., before a process can request any additional resources, it must release

all the resources that it is cur nay to (e.g., at the beginning)
Allow process to request resources only when the protiety.
I.e., before a process can request any additional resources
all the resources that it is currently allocated
sadvantages
Low resour
	- □ Disadvantages
		-
		-

Deadlock Prevention

- □ Deadlock Preemption
■ No Preemption
■ If a process that is holding some resour □ **Deadlock Prevention**

■ If a process that is holding some resources requests another resource

that cannot be immediately allocated to it, then all resources currently

→ hains better an invitable and another **Deadlock Prevention
That cannot be immediately allocated to it, then all resources currently
being held are implicitly released Deadlock Prevention**
 Preemption

If a process that is holding some resources requests

that cannot be immediately allocated to it, then all reprends the implicitly released

Preempted resources are added to the list of <p>Deadlock Prevention</p>\n<p>■ 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 implicitly released</p>\n<p>Preempted resources are added to the list of resources for which the process is waiting</p> **Preemption**
If a process that is holding some resources requerent
that cannot be immediately allocated to it, then a
being held are implicitly released
Preempted resources are added to the list of res
process is waiting
P <p>• Preemption</p>\n<p>• 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 implicitly released</p>\n<p>• Preempted resources are added to the list of resources for which the process is waiting</p>\n<p>• Process will be restarded only when it can regain its old resources, as well as the new ones that it is requesting</p> If a process that is holding some resources requests another resc
that cannot be immediately allocated to it, then all resources curre
being held are implicitly released
Preempted resources are added to the list of resourc
	-
	-

**Deadlock Prevention

Deadlock Prevention

Deadlock Prevention**

- □ Disadvantage
- □ Deadlock Prevention
■ No Preemption (Cont.)
■ Disadvantage
■ It is often applied to resources whose state can be easi **Deadlock Prevention
Preemption (Cont.)**
Disadvantage
- It is often applied to resources whose state can be easily saved and restored
- It cannot generally be applied to such resources as printers and tape drives **Deadlock Prevention

reemption (Cont.)**

sadvantage

It is often applied to resources whose state can be easily say

later, such as CPU registers

It cannot generally be applied to such resources as printers and the Wait **Deadlock Prevention

Preemption (Cont.)**

Disadvantage

- It is often applied to resources whose state can be easily saved and restored

- It cannot generally be applied to such resources as printers and tape drives

cula
	-

□ Deadlock Preventi

■ **No Preemption (Cont.)**

■ Disadvantage

■ It is often applied to resources whose state can later, such as CPU registers

■ It cannot generally be applied to such resource

■ Circular Wait

■ Impo **Deadlock Prevention

Disadvantage

- It is often applied to resources whose state can be easily saved and restored

- It cannot generally be applied to such resources as printers and tape drives

Circular Wait

- Impose Preemption (Cont.)**

Disadvantage

It is often applied to resources whose state can be easily saved and restored

later, such as CPU registers

It cannot generally be applied to such resources as printers and tape drives

Deadlock Prevention

□ Disadvantage: It is up to application developers to write programs that follow the ordering

Deadlock Avoidance

- Deadlock Avoidance

In deadlock prevention, we constrain resource requests which leads to

inefficient use of resources and inefficient execution of processes Deadlock Avoidance
In deadlock prevention, we constrain resource requests which leads to
inefficient use of resources and inefficient execution of processes
With deadlock avoidance, a decision is made dynamically whether t
- Deadlock Avoidance

In deadlock prevention, we constrain resource requests which leads to

inefficient use of resources and inefficient execution of processes

With deadlock avoidance, a decision is made dynamically whethe Deadlock Avoidance
The deadlock prevention, we constrain resource requests which leads to
inefficient use of resources and inefficient execution of processes
With deadlock avoidance, a decision is made dynamically whether □ Deadlock Avoidance

■ In deadlock prevention, we constrain resource requests which leads to

inefficient use of resources and inefficient execution of processes

■ With deadlock avoidance, a decision is made dynamically
- requests

Safe State

- **Safe State**

A state is safe if the system can allocate resources to each process (up to its

maximum) in some order and still avoid a deadlock (i.e., all of the processes can

be run to completion) Safe State
A state is safe if the system can allocate resources to each process (up to its
maximum) in some order and still avoid a deadlock (i.e., all of the processes can
be run to completion) **Safe State**
A state is safe if the system can allocate resources
maximum) in some order and still avoid a deadlock
be run to completion)
System is in safe state if there exists a sequence **Safe State**
 **Safe state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock (i.e., all of the processes can be run to completion)

System is in safe**
- Safe State
A state is safe if the system can allocate resources to each process (up to its
maximum) in some order and still avoid a deadlock (i.e., all of the processes can
be run to completion)
System is in safe state if Sate State is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock (i.e., all of the processes can be run to completion)
System is in safe state if there e A state is safe if the system can allocate resources to each promaximum) in some order and still avoid a deadlock (i.e., all of be run to completion)
System is in safe state if there exists a sequence $\langle P_1, P_2, \ldots,$ pro

Safe, Unsafe, Deadlock State □ If a system is in safe state no deadlocks

-
- deadlock
-
- resource, system is in unsafe state \Rightarrow no deadlocks

system is in unsafe state \Rightarrow possibility of

adlock

bidance \Rightarrow ensure that a system will

ver enter an unsafe state

When a process requests an available

resour

Avoidance Algorithms
e of a resource type **Avoidance Algorithms**

• Single instance of a resource type

• Use a resource-allocation graph • Use a resource-allocation graph **Avoidance Algorithms**

• Single instance of a resource type

• Use a resource-allocation graph

• Multiple instances of a resource type

• Use the Banker's Algorithm **Avoidance Algorithms**

Ingle instance of a resource type

• Use a resource-allocation graph

Intiple instances of a resource type

• Use the Banker's Algorithm

- -
- -

Resource-Allocation Graph Scheme
 Resource-Allocation Graph Scheme
 Resource $P_i \rightarrow R_j$ **indicated that process** P_j **may request resource** R_j **. Property:**
 Resource-Allocation Grapl
 Claim edge $P_i \rightarrow R_j$ indicated that process P_j may reque represented by a dashed line
 Claim edge converts to request edge when a process re CE-Allocation Graph Scheme

indicated that process P_j may request resource R_j ;

shed line

to request edge when a process requests a resource **Resource-Allocation Graph Scheme**
Claim edge $P_i \rightarrow R_j$ indicated that process P_j may request resource R_j ,
represented by a dashed line
Claim edge converts to request edge when a process requests a resource
Request edg **Example 2**
 Example 2
 Example 2
 Example 2
 Example 2
 Claim edge $P_i \rightarrow R_j$ **indicated that process** P_j **may request resource** R_j **;

Process request edge converted to an assignment edge when the resource is allo EXECUTE:**
 Resource-Allocation Graph Scheme

• Claim edge $P_i \rightarrow R_j$ indicated that process P_j may request resource R_j ;

• Claim edge converts to request edge when a process requests a resource

• Request edge conver Resource-Allocation C

Claim edge $P_i \rightarrow R_j$ indicated that process P_j m

represented by a dashed line

Claim edge converts to request edge when a pr

Request edge converted to an assignment edge

to the process

When a r **Example 19 Allocation Graph Scheme**

• Claim edge $P_i \rightarrow R_j$ indicated that process P_j may request resource R_j ;

• Claim edge converts to request edge when a process requests a resource

• Request edge converted to an Resource-Allocation Grand Claim edge $P_i \rightarrow R_j$ indicated that process P_j may represented by a dashed line
Claim edge converts to request edge when a proces
Request edge converted to an assignment edge wh
to the process
W

-
-
- **EXERICAL RESOURCE-ALLOCATION GRAPH Scheme**

 Claim edge $P_i \rightarrow R_j$ indicated that process P_j may request resource R_j ;

 Claim edge converts to request edge when a process requests a resource

 Request edge converted
-
- Resources must be claimed a priori in the system $\footnotesize{\textcircled{\textup{a}}}$
 $\footnotesize{\textcircled{\textup{c}}}_{\text{Dperating System Concepts 10^m} \text{ Edition}}$ 8.26 Silberschatz, Galvin and Gagne ©2018

Banker's Algorithm

- □ Banker's Algorithm
□ The banker's algorithm is used when we have Multiple instances of a
resource type **Sanker's Algorithm
The banker's algorithm is used when we have resource type
The Single instance of a resource type, we can us** ○ For Single instance of a resource type, we can use resource-allocation graph

a For Single instance of a resource type, we can use resource-allocation graph

(see Silberschatz)
— Inker's Algorithm

e banker's algorithm is used when we hav

source type

For Single instance of a resource type, we can

(see Silberschatz)

e name was chosen because the algorithn
	-
- The banker's algorithm is used when we have Multiple instances of a

resource type

 For Single instance of a resource type, we can use resource-allocation graph

(see Silberschatz)

The name was chosen because the alg The banker's algorithm is used when we have Multiple instances of a
resource type
 \overline{P} For Single instance of a resource type, we can use resource-allocation graph
(see Silberschatz)
The name was chosen because the al in such a way that it could no longer satisfy the needs of all its customers The cannot declare to maximum and the matter matter of the maximum resource of a resource type, we can use resource-allocation graph

(see Silberschatz)

The name was chosen because the algorithm could be used in a

banki For Single instance of a resource type, we can use resource-allocation graphs (see Silberschatz)
The name was chosen because the algorithm could be used in a
banking system to ensure that the bank never allocated its avail
-

Data Structures for Banker's Alg.

-
- □ Data Structures for Banker's Alg.
■ These data structures encode the state of the resource-allocation system
■ Let n be the number of processes, and m be the number of resources □ Data Structures for Banker's Alg.
■ These data structures encode the state of the resource-allocation system
■ Let n be the number of processes, and m be the number of resources
types types Data Structures for Banker's Alg.
■ These data structures encode the state of the resource-allocation system
■ Let n be the number of processes, and mbe the number of resources
types
■ Resource: Vector of length m indicat Data Structures for Barriers for B
These data structures encode the state of the the number of processes, and m be
types
Resource: Vector of length m indicates total
in the system.
Available: Vector of length m indicates t <p>■ These data structures encode the state of the resource-allocation system</p>\n<p>■ Let n be the number of processes, and m be the number of resources types</p>\n<p>■ Resource: Vector of length m indicates total amount of each resource in the system.</p>\n<p>■ Available: Vector of length m indicates the number of available resources of each type.</p>\n<p>■ If available[j] = k, there are k instances of resource type R_j available</p> These data structures encode the state of the
Let n be the number of processes, and m be t
types
Resource: Vector of length m indicates total an
in the system.
Available: Vector of length m indicates the nur
of each type.
 These data structures encode the state of the resource-allocation system
et n be the number of processes, and m be the number of resources
pes
Resource: Vector of length m indicates total amount of each resource
in the sy
-
- -

Data Structures for Banker's Alg.

- **Data Structures for Banker's Alg.**

Max: n × m matrix defines the maximum demand of each process. If

Max [i, j] = k, then process P_i may request at most k instances of

resource type R_j Data Structures for Banker's Alg.

Max: $n \times m$ matrix defines the maximum demand of each process. If

Max [i, j] = k, then process P_i may request at most k instances of

resource type R_j

Allection: n u.m. matrix. If **Data Structures for Banker**
Max: n × m matrix defines the maximum demand of
Max [i, j] = k, then process P_i may request at most k
resource type R_j
Allocation: n × m matrix. If Allocation[i, j] = k then P_i **Data Structures for Banker's Alg.**

■ Max: n × m matrix defines the maximum demand of each process. If

Max [i, j] = k, then process P_i may request at most k instances of

resource type R_j

■ Allocation: n × m matri **is Alg.**
 instances of
 is currently allocated
 is currently allocated Data Structures for Banker

Max: n × m matrix defines the maximum demand of

Max [i, j] = k, then process P_i may request at most k

resource type R_j

Allocation: n × m matrix. If Allocation[i, j] = k then P_i

k in **Data Structures for Banker's Alg.**

■ Max: n × m matrix defines the maximum demand of each process. If

Max [i, j] = k, then process P_i may request at most k instances of

resource type R_j

■ Allocation: n × m matri Max: n × m matrix defines the maximum demand of each
Max [i, j] = k, then process P_i may request at most k instant
resource type R_j
Allocation: n × m matrix. If Allocation[i, j] = k then P_i is cur
k instances of R_j Max: n × m matrix defines the maximum demand of each process. If
Max [i, j] = k, then process P_i may request at most k instances of
resource type R_j
Allocation: n × m matrix. If Allocation[i, j] = k then P_i is curre
-
-

Banker's Algorithm: an Example Banker's Algorithm: an Exa

Is the system safe?

Resource = $\frac{R_0 - R_1 - R_2 - R_3}{5 + 7 + 2 + 3}$ Avail = $\frac{R_0 - R_1 - R_2}{1 + 1 + 1}$

 $(1,1,0,1)$ $\xrightarrow{P_2} (1,2,1,1)$ $\xrightarrow{P_1} (2,4,1,1)$ $\xrightarrow{P_3} (4,7,2,1)$ $\xrightarrow{P_0} (5,7,2,3)$ $\xrightarrow{P_4} (5,7,2,3)$ P_2 (1,2,1,1) $\frac{P_1}{\longrightarrow}$ (2,4,1,1) $\frac{P_3}{\longrightarrow}$ (4,7,2,1) $\frac{P_0}{\longrightarrow}$ (5,7,2,3) $\frac{P_4}{\longrightarrow}$ (5,7,2,3)

Banker's Algorithm: an Example

Banker's Algorithm: an Example

\n
$$
P_4
$$
 asks for an instance of R_0 , is it allocated?

\n**Resource =** $\frac{R_0 - R_1 - R_2 - R_3}{5 - 7 - 2 - 3}$

\n**Available**

\n $\frac{R_0 - R_1 - R_2 - R_3}{R_0 - R_1 - R_2 - R_3}$

\n $\frac{R_0 - R_1 - R_2 - R_3}{R_0 - R_1 - R_2 - R_3}$

\n $\frac{R_0 - R_1 - R_2 - R_3}{R_0 - R_1 - R_2 - R_3}$

\n $\frac{R_0 - R_1 - R_2 - R_3}{R_0 - R_1 - R_2 - R_3}$

\n $\frac{R_0 - R_1 - R_2 - R_3}{R_0 - R_1 - R_2 - R_3}$

\n $\frac{R_0 - R_1 - R_2 - R_3}{R_0 - R_1 - R_2 - R_3}$

\n $\frac{R_0 - R_1 - R_2 - R_3}{R_0 - R_1 - R_2 - R_3}$

\n $\frac{R_0 - R_1 - R_2 - R_3}{R_0 - R_1 - R_2 - R_3}$

\n $\frac{R_0 - R_1 - R_2 - R_3}{R_0 - R_1 - R_2 - R_3}$

\n $\frac{R_0 - R_1 - R_2 - R_3}{R_0 - R_1 - R_2 - R_3}$

Max

Allocation Need

 $(0,1,0,1) \xrightarrow{P_2} (0,2,1,1)$

Banker's Algorithm: an Example

 $(0,1,0,1)$ $\xrightarrow{P_2} (0,2,1,1)$ $\xrightarrow{P_1} (2,4,1,1)$ $\xrightarrow{P_3} (4,7,2,1)$ $\xrightarrow{P_0} (5,7,2,3)$ $\xrightarrow{P_4} (5,7,2,3)$ P_2 (0,2,1,1) $P_1 \rightarrow (2,4,1,1)$ $P_3 \rightarrow (4,7,2,1)$ $P_0 \rightarrow (5,7,2,3)$ $P_4 \rightarrow (5,7,2,3)$

Safety Algorithm
Einish be vectors of length m and n respectively **Safety Algorithm**

bet *Work* and *Finish* be vectors of length *m* a

itialize:

Work =Available

Finish[i] = false for i = 0, 1, ..., n -1

nd an i such that both: Safety Algorithm

et Work and Finish be vectors of length m and n, re

initialize:

Work = Available

Finish[i] = false for i = 0, 1, ..., n -1

ind an i such that both:

(a) Finish[i] = false

(b) Need_i ≤ Work

If no s

Safety Algorithm

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

Initialize:

Work = Available **Safety Algorithm**

Finish be vectors of length *m* and *n*, respectively

itialize:

Work = Available

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

and an i such that both:

a) Finish[i] = false **2. Find an i such that both:**

2. Find an i such that both:

2. Find an i such that both:

2. Find an i such that both:

4. Finish[i] = false

4. Finish[i] = false

4. Finish[i] = false

4. Finish[i] = false

4. Work **SATETY AIGOTITINT**

et *Work* and *Finish* be vectors of length *m*

initialize:

Work = Available

Finish[i] = false for i = 0, 1, ..., n -1

ind an i such that both:

(a) Finish[i] = false

(b) Need_i ≤ Work

If no su et *Work* and *Finish* be vectors of length *m* and *n*, respectively
itialize:
Work = Available
Finish[i] = false for i = 0, 1, ..., n -1
ind an i such that both:
(a) Finish[i] = false
(b) Need_i ≤ Work
If no such i exi

Initialize:

- -
	-

- 1. Let *Work* and *Finish* be vectors of length *m* and *n*, r

Initialize:

Work = Available

Finish[i] = false for i = 0, 1, ..., n -1

2. Find an i such that both:

(a) Finish[i] = false

(b) Need_i ≤ Work

If no such Initialize:

Work = Available

Finish[i] = false for i = 0, 1, ..., n -1

Find an i such that both:

(a) Finish[i] = false

(b) Need_i ≤ Work

If no such i exists, go to step 4

Work = Work + Allocation_i

Finish[i] = t Multimartize:

Work = Available

Finish[i] = false for i = 0, 1, ..., n -1

Find an i such that both:

(a) Finish[i] = false

(b) Need_i ≤ Work

If no such i exists, go to step 4

Work = Work + Allocation_i

Finish[i] = Finish[i] = false for i = 0, 1, ..., n -1

2. Find an i such that both:

(a) Finish[i] = false

(b) Need_l ≤ Work

If no such i exists, go to step 4

3. Work = Work + Allocation_i

Finish[i] = true

go to step 2

4. If
-

Resource-Request Algorithm

Securet: < Need as to step 2. Otherwise, raise error condition, since

- Resource-Request Algorithm

1. If Request_{i ≤} Need_i go to step 2. Otherwise, raise error condition, since

process has exceeded its maximum claim

2. If *Request: < Available*, go to step 3. Otherwise *P* must wait, si Resource-Request Algorithm

If Request_i \leq Need_i go to step 2. Otherwise, raise error condition, since

process has exceeded its maximum claim

If Request_i \leq Available, go to step 3. Otherwise P_i must wait,
- **Resource-Request Algorithm**

2. If Request_i \leq Need_i go to step 2. Otherwise, raise error condition, since

process has exceeded its maximum claim

2. If Request_i \leq Available, go to step 3. Otherwise P_i mu Resource-Request Algorithm

If Request_i \leq Need_i go to step 2. Otherwise, raise error cond

process has exceeded its maximum claim

If Request_i \leq Available, go to step 3. Otherwise P_i must wait,

resources
- **3.** If Request_i \leq Need_i go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If Request_i \leq Available, go to step 3. Otherwise P_i must wait, since resources are n follows: **• If safe the resources has exceeded its maximum claim**
 • Request_i ≤ Available, go to step 3. Otherwise P_i must wait, since

• esources are not available

• Pretend to allocate requested resources to P_i by modi resources are not available

3. Pretend to allocate requested resources to P_i by modifying the state as

follows:
 $A\text{valid}ble = Available - Request;$
 $A\text{local} = N\text{equal} - Request;$
 $N\text{eed}_i = N\text{equal} - Request;$
 \cdot If safe \rightarrow the resources are allocated t 3. Pretend to allocate requested resources to P_i by modifying the stand follows:
 α vailable = Available - Request;

Allocation_i = Allocation_i + Request;

Need_i = Need_i - Request;
 \cdot If safe \rightarrow the resour

Available = Available – Request;

Allocation_i = Allocation_i + Request_i;

 $Need_i = Need_i - Request_i;$

-
-

Banker's Algorithm Disadvantages □ Banker's Algorithm Disadvantages

Maximum resource requirement must be stated in advance

□ There must be a fixed number of resources to allocate

□ The algorithm is very conservative □ Banker's Algorithm Disadvantages

■ Maximum resource requirement must be stated in advance

■ There must be a fixed number of resources to allocate

■ The algorithm is very conservative

■ It limits access to resources □ Banker's Algorithm Disadvantag
■ Maximum resource requirement must be stated in advance
■ There must be a fixed number of resources to allocate
■ The algorithm is very conservative
■ It limits access to resources **Banker's Algorithm Disadvanta**
Maximum resource requirement mustbe stated in advanta
There must be a fixed number of resources to allocate
The algorithm is very conservative
a It limits access to resources

-
-
- -

Deadlock Detection and Recovery

- □ If a system does not employ either a deadlock prevention or a deadlock Deadlock Detection and Recovery

If a system does not employ either a deadlock prevention or a deadlock

avoidance algorithm, then a deadlock situation may occur

Then, the system must provide Deadlock Detection and Recove

■ If a system does not employ either a deadlock prevention or

a voidance algorithm, then a deadlock situation may occur

■ Then, the system must provide

■ An algorithm that examines the st □ An algorithm that examines the state of the system to determine

an algorithm, then a deadlock situation may occur

Then, the system must provide

■ An algorithm that examines the state of the system to determine

wheth Deadlock Detection and Recover

In system does not employ either a deadlock prevention or a

bidance algorithm, then a deadlock situation may occur

en, the system must provide

An algorithm that examines the state of the The system does not employ either a deadlock prevention or a deadlock
avoidance algorithm, then a deadlock situation may occur
Then, the system must provide
a An algorithm that examines the state of the system to determin
- -
	-

**Deadlock Detection

tance of Each Resource Type

Clien graph or a variant of it, called a wait-for graph Deadlock Detection
Single Instance of Each Resource Type**
 Example 2016 Tesource-allocation graph or a variant of it, called a wait-for graph
 Example 10.4
 Example 10.4
 Example 10.4
 Example 10.4
 Example 10. Example 19 Single Instance of Each Resource Type

• Use resource-allocation graph or a variant of it, called a wait-for graph

• Maintain wait-for graph

• Nodes are processes • Deadlock Detection
• Single Instance of Each Resource Ty
• Single Instance of Each Resource Ty
• Nodes are processes
• $P_i \rightarrow P_j$ if P_i is waiting for P_j
• Priodically invoke an algorithm that searches for a cycle in **is ware controlled in the graph of Picket Conduct and Single Instance of Each Resource Type

• Use resource-allocation graph or a variant of it, called a wait-for graph

• Nodes are processes

•** $P_i \rightarrow P_j$ **if** P_i **is waiti Single Instance of Each Resource Type**
Use resource-allocation graph or a variant of it, called a wait-for graph
Maintain wait-for graph
Nodes are processes
• $P_i \rightarrow P_j$ if P_i is waiting for P_j
Periodically invoke an a **Example 10.1**
 Example Instance of Each Resource Type

• Use resource-allocation graph or a variant of it, called a wait-for graph

• Naintain wait-for graph

• Nodes are processes

• $P_i \rightarrow P_j$ if P_i is waiting for P **Single Instance of Each Resource Type**

Use resource-allocation graph or a variant of it, called a wait-for graph

Maintain wait-for graph

• Nodes are processes

• $P_i \rightarrow P_j$ if P_i is waiting for P_j

Periodically invo

-
- Maintain wait-for graph
	-
	- $P_i \rightarrow P_j$ if P_i is waiting fo
-
-

Deadlock Detection

-
- □ Deadlock Detection
○ Several instances of each resource type
○ The algorithm is very similar to the banker's algorithm, with two main O DE CHI THE ALGORY OF ALGORY OF A THE ALGORY OF A THE ALGORY AND NOT A THE ALGORY AND NOT A THE MATTER CHI THE POST OF A THE MATTER CHI THE POST OF A differences
	- **The matrices Max and Need are replaced with an** $n \times m$ **matrix Request indicates** the current request of each produce type
the algorithm is very similar to the banker's algorithm, with two main
ferences
The matrices Max and Need are replaced with an $n \times m$ matrix Request indicates
the current request o the current request of each process. If Request[i][j] = k, then process P_i is requesting Framework and instances of each resource type
that instances of each resource type
e algorithm is very similar to the banker's algorithm, with two main
ierences
The matrices Max and Need are replaced with an $n \times m$ matrix First a process of each resource type

	The algorithm is very similar to the banker's algorithm, with two main

	ifferences

	The matrices Max and Need are replaced with an $n \times m$ matrix Request indicates

	the current reques ■ The algorithm is very similar to the banker's algorithm, with two main
differences

	■ The matrices Max and Need are replaced with an $n \times m$ matrix Request indicates

	the current request of each process. If Request[i][j
		- finished.
	- processes are deadlocked

Deadlock Detection: an Example

$$
(0,0,0,0,1) \xrightarrow{P_2} (0,0,0,1,1)
$$
 Yes,

- **Detection Algorithm**

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

Initialize:

a) Work = Available Initialize: **Detection Algorithm**

et Work and Finish be vectors of length m and n, respectively

itialize:
 a) Work = Available

b) For $i = 1,2, ..., n$, if Allocation_i < 0, then

Finish[i] = false; otherwise, Finish[i] = true **Detection Algorithm**
 rk and *Finish* be vectors of length *m* and *n*, respectively

Finish is example to the straight of $i = 1, 2, ..., n$, if *Allocation*_i < 0, then
 Finish[i] = *false*; otherwise, *Finish*[i] = *tr* **1.** Let Work and Finish be vectors of length m and n, respectively

Initialize:
 a) Work = Available
 b) For $i = 1, 2, ..., n$, if Allocation_i < 0, then

Finish[i] = false; otherwise, Finish[i] = true

2. Find an index **DETECTION AIGOTITIM**

et Work and Finish be vectors of length m and n, responitialize:

a) Work = Available

b) For $i = 1, 2, ..., n$, if Allocation_i < 0, then

Finish[i] = false; otherwise, Finish[i] = true

ind an index
	- a) Work = Available
	- **a)** Work = Available

	b) For $i = 1, 2, ..., n$, if Allocation_i < 0, then

	Finish[i] = false; otherwise, Finish[i] = true

	ind an index i such that both:
 a) Finish[i] = false
 b) Request_i < Work

	If no such i exis
- -
	- b) Request_i < Work

Detection Algorithm (Cont.) **Detection Algorithm (C**

Work = Work + Allocation_i

Finish[i] = true

go to step 2

f Finish[i] == false for some *i* 1 < *i* < *n*, then the

- 3. Work = Work + Allocation,
	-

Detection Algorithm (Co

ork = Work + Allocation_i

nish[i] = true

go to step 2

inish[i] == false, for some i, 1 ≤ i ≤ n, then the system

adlock state Moreover if *Finish*[i] == false, then P.i **1. If Finish and Solution Algorithm (Cont.)**

1. If Finish $[I] = true$

go to step 2
 4. If Finish $[i] =$ false, for some i, 1 $\le i \le n$, then the system is in

deadlock state. Moreover, if Finish $[i] =$ false, then P_i is de deadlock state. Moreover, if \boldsymbol{Finish} [*i*] == *false*, then \boldsymbol{P}_i is $\,$ deadlocked $\,$ go to step 2
4. If *Finish[i]* == *false*, for some *i*, $1 \le i \le n$, then the system is in
deadlock state. Moreover, if *Finish[i]* == *false*, then P_i is deadlocked
Algorithm requires an order of O(*m* x n^2) operatio go to step 2
4. If *Finish[i]* == *false*, for some *i*, $1 \le i \le n$, then the system is in deadlock state. Moreover, if *Finish[i]* == *false*, then P_i is deadlocked state.
Algorithm requires an order of O($m \times n^2$) oper

Algorithm requires an order of O($m \times n^2$) operations to detect
whether the system is in deadlocked state
Operating System Concepts – 10th Edition 8.43 Silberschatz, Galvin and Gagne ©2018

Deadlock Recovery □ △ Abort all deadlocked processes
△ Abort all deadlocked processes
△ One of the most common solution adopted in operating system □ One of the most common solution adopted in operating systems

- -
	-
- □ Deadlock Recovery
Abort all deadlocked processes
□ One of the most common solution adopted in operation
Back up each deadlocked process to some previo □ Back up each deadlocked process to some previously defined checkpoint, ■ Deadlock Recovery
Abort all deadlocked processes
■ One of the most common solution adopted in operating
■ Expensive solution
Back up each deadlocked process to some previou
and restartall process
■ Original deadlock may Abort all deadlocked processes

□ One of the most common solution adopted in operating systems

□ Expensive solution

Back up each deadlocked process to some previously defined che

and restart all process

□ Original dea National deadlocked processes

□ One of the most common solution adopted in operating systems

□ Expensive solution

Back up each deadlocked process to some previously defined checkpoin

and restart all process

□ Origina
	-
	-

Detection-Algorithm Usage **Detection-Algorithm Usage**

• When, and how often, to invoke depends on:

• How often a deadlock is likely to occur? **Detection-Algorithm Usage**

Vhen, and how often, to invoke depends on:

• How often a deadlock is likely to occur?

• How many processes will need to be rolled back? **• Detection-Algorithm Usage**

• When, and how often, to invoke depends on:

• How often a deadlock is likely to occur?

• How many processes will need to be rolled back?

• one for each disjoint cycle

- -
	- -
- The resource graph and so we would not be able to tell which of the
many deadlocked processes "caused" the deadlock.

Operating System Concepts 10th Edition 8.46 Silberschatz, Galvin and Gagne ©2018 **Detection-Algorithm Usage**

• The mass of the settion algorithm is invoked arbitrarily, there may be many cycles in

• Source graph and so we would ■ Usage

The resource graph and so we would not be able to **Detection-Aigorithm Usage**

When, and how often, to invoke depends on:

■ How often a deadlock is likely to occur?

■ How many processes will need to be rolled back?

■ version for each disjoint cycle

If detection algor When, and how often, to invoke depends on:

• How often a deadlock is likely to occur?

• How many processes will need to be rolled back?

• one for each disjoint cycle

If detection algorithm is invoked arbitrarily, there

Deadlock Recovery

- □ Deadlock Recovery
□ Successively abort deadlocked processes until deadlock no
□ longer exists **Deadlock Recessively abort deadlocked processively abort deadlocked procession and processively abort deadlocked procession and procession and procession and processed algebra and processed and processed algebra and proce** Deadlock-Recovery
iccessively abort deadlocked processes until deadlock no
nger exists
incurs considerable overhead, since after each process is aborted, a
deadlock-detection algorithm must be invoked
iccessively preempt r ■ Successively abort deadlocked processes until deadlock no

longer exists

■ Incurs considerable overhead, since after each process is aborted, a

deadlock-detection algorithm must be invoked

■ Successively preempt reso
	- □ Incurs considerable overhead, since after each process is aborted, a
-

Deadlock Recovery
methods requires some selection criteria to cho

- □ The last two methods requires some selection criteria to choose the victim process □ Deadlock Recovery
□ The last two methods requires some selection criteria to
the victim process
□ Choose the process with the
□ least amount of processor time consumed so far
□ least amount of output produced so far □ Deadlock Recovery

The last two methods requires some selection criteria to choose

the victim processor

Choose the processor time consumed so far

■ least amount of output produced so far

■ most estimated time remain □ **Deadlock Recovery**

The last two methods requires some selection criteria to choose

the victim process

Choose the process with the

□ least amount of processor time consumed so far

□ least amount of output produced □ Deadlock Recovery

The last two methods requires some selection crite

the victim process

Choose the process with the

□ least amount of processor time consumed so far

□ least amount of output produced so far

□ most **DEADIOCK NECOVET Y**

The last two methods requires some selection criteria to choothe victim process

Choose the process with the
 Example 1 least amount of processor time consumed so far
 Example 1 least amount of o The last two methods requires some set
the victim process
Choose the process with the
a least amount of processor time consumed so
a least amount of output produced so far
a most estimated time remaining
a least total res
- -
	-
	-
	-
	-

-
-
-

End of Chapter 8

