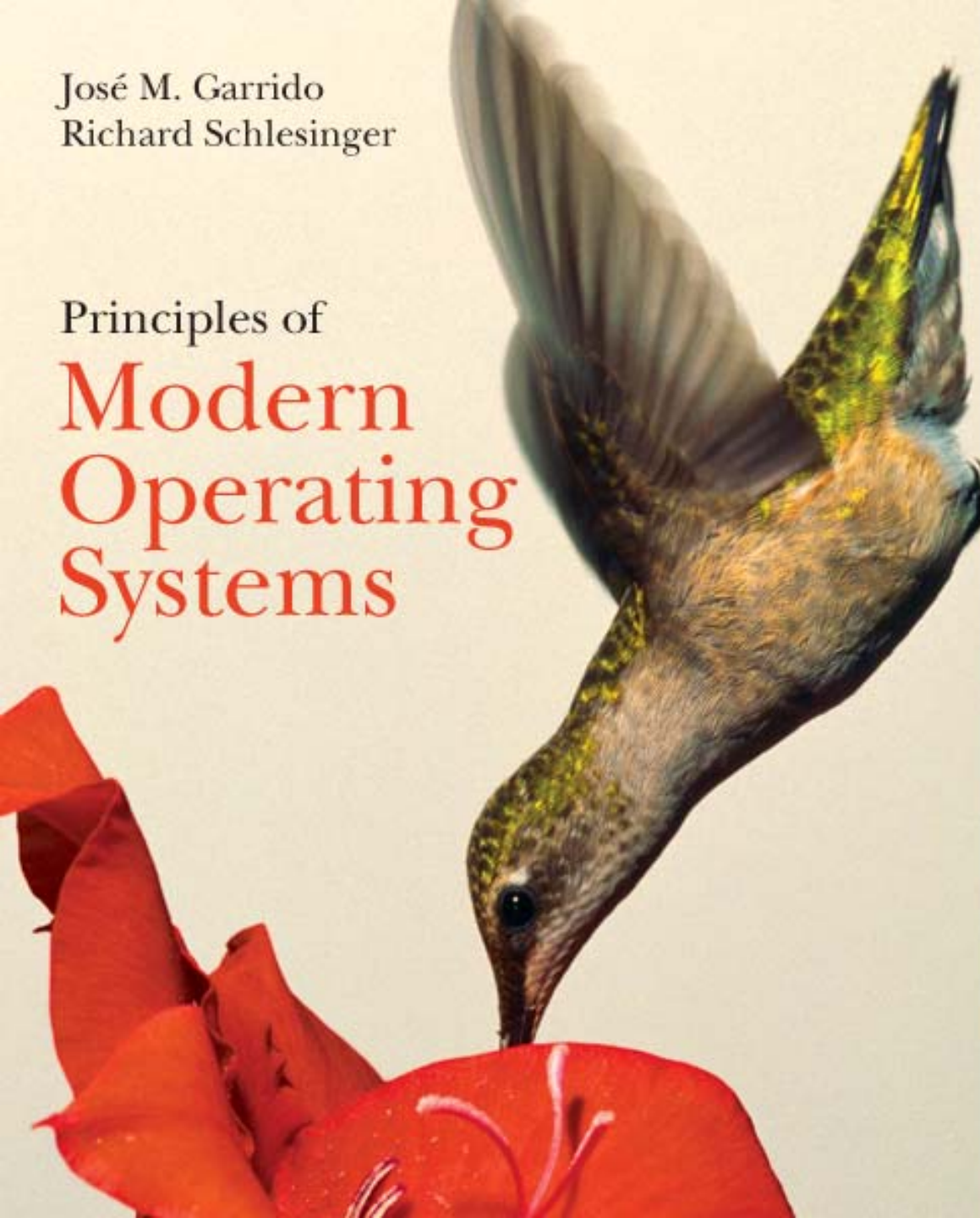


José M. Garrido
Richard Schlesinger

Principles of
**Modern
Operating
Systems**



Chapter 7

Deadlock

Resource Allocation and De-allocation

When a process needs resources, it will normally follow the sequence:

1. Request a number of instances of one or more resource types. If the resource units requested are not available, then process must wait
2. Acquire the resource(s), the OS allocates resources
3. Use the resource(s) for a finite period.
4. Release the resource(s).

Synchronizing Allocation of Resources

Semaphores can be used to synchronize the allocation and de-allocation of resources.

Multiple Resource Allocation Problem

- Several processes may compete for multiple resources
- If the allocation of multiple resources to processes is not done with proper synchronization, deadlock might occur
- This waiting state is much more involved than starvation

Deadlock Problem

- A set of blocked (suspended) processes
- Each process holding one or more resources
- Processes waiting to acquire one or more resources held by other processes

Example of Deadlock

- A system has two tape drives, which processes P1 and P2 need
- Processes P1 and P2 each acquire one tape drive and each wait for the other tape drive
- Processes P1 and P2 are blocked waiting indefinitely to acquire the other tape unit

Deadlock – General Definition

- A set of processes is in a deadlock state when every processes in the set is waiting for an event that can only be caused by another process in the set.
- The events of interest are:
 - allocations of resources
 - release of resources
- The resources are: CPU cycles, memory spaces, files, I/O devices in the computer system.

Disadvantage Of Deadlock

- Deadlocked processes can never complete execution because they are waiting indefinitely
- System resources are tied up because they are held by deadlocked processes.
- Deadlocks are undesirable because they normally slow down a system

Another Example Of Deadlock

- Consider a system with one printer and one disk unit. Processes P and Q both need the two resources.
- Suppose that process P is holding the printer, and process Q is holding the disk unit. Now P waits for the disk unit and Q waits for the printer.
- Process P is waiting for process Q to release the disk unit, and process Q is waiting for process P to release the printer.

Result

Both processes are deadlocked, they cannot proceed.

Conditions For Deadlock

A deadlock arises if four conditions hold *simultaneously* in a system:

- Mutual exclusion: only one process at a time can use a resource.
- Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by other processes.

Conditions for Deadlock (Cont.)

- No preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its operations.
- Circular wait: there exists a set $\{P_0, P_1, \dots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , but P_1 is waiting for a resource held by P_2 , ..., P_{n-1} is waiting for a resource held by P_n , and P_n is waiting for a resource held by P_0 .

About These Conditions

These conditions are necessary but not sufficient for deadlock to occur.

Resource Allocation Graph

A resource allocation graph shows the relationships between processes and resources.

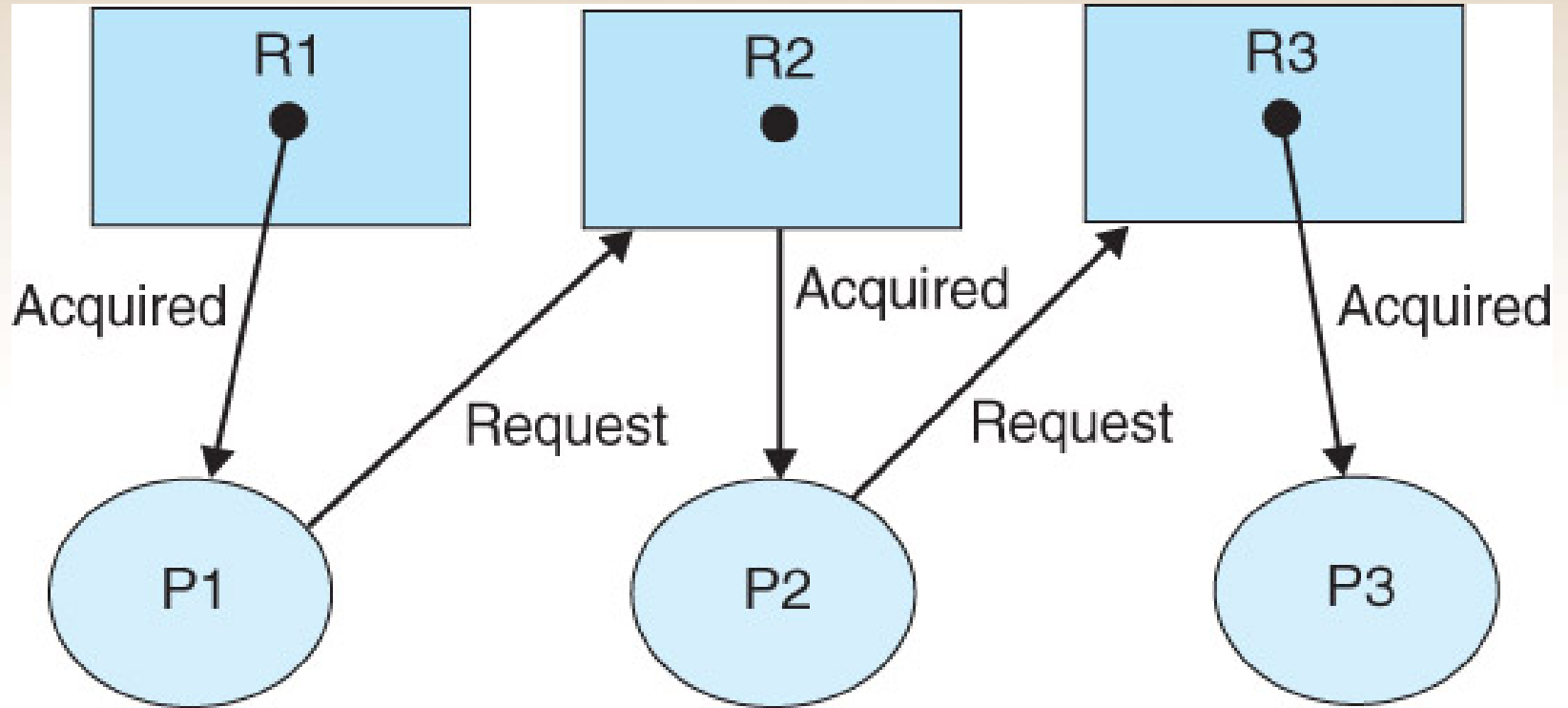
It consists of the following components:

- A set of processes $P = \{ P_0, P_1, \dots, P_i, \dots, P_n \}$
- A set of resource types,
$$R = \{ R_0, R_1, \dots, R_j, \dots, R_m \}$$
- A set of directed edges

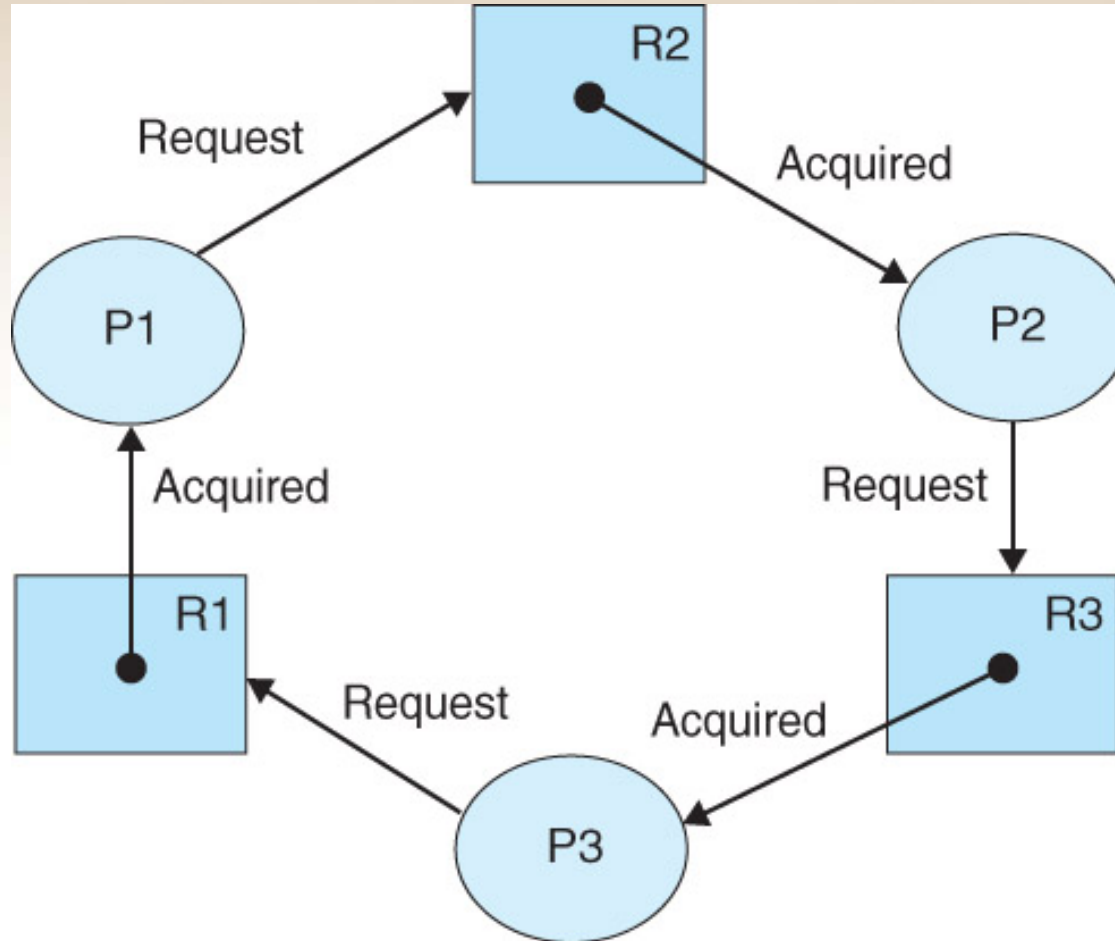
Directed Edges

- A request edge (P_i, R_j) from process P_i to resource R_j . Thus, P_i is waiting for an instance of resource type R_j .
- An assignment edge (R_j, P_i) from resource R_j to process P_i . It indicates that an instance of R_j has been allocated to process P_i .

A Resource Allocation Graph



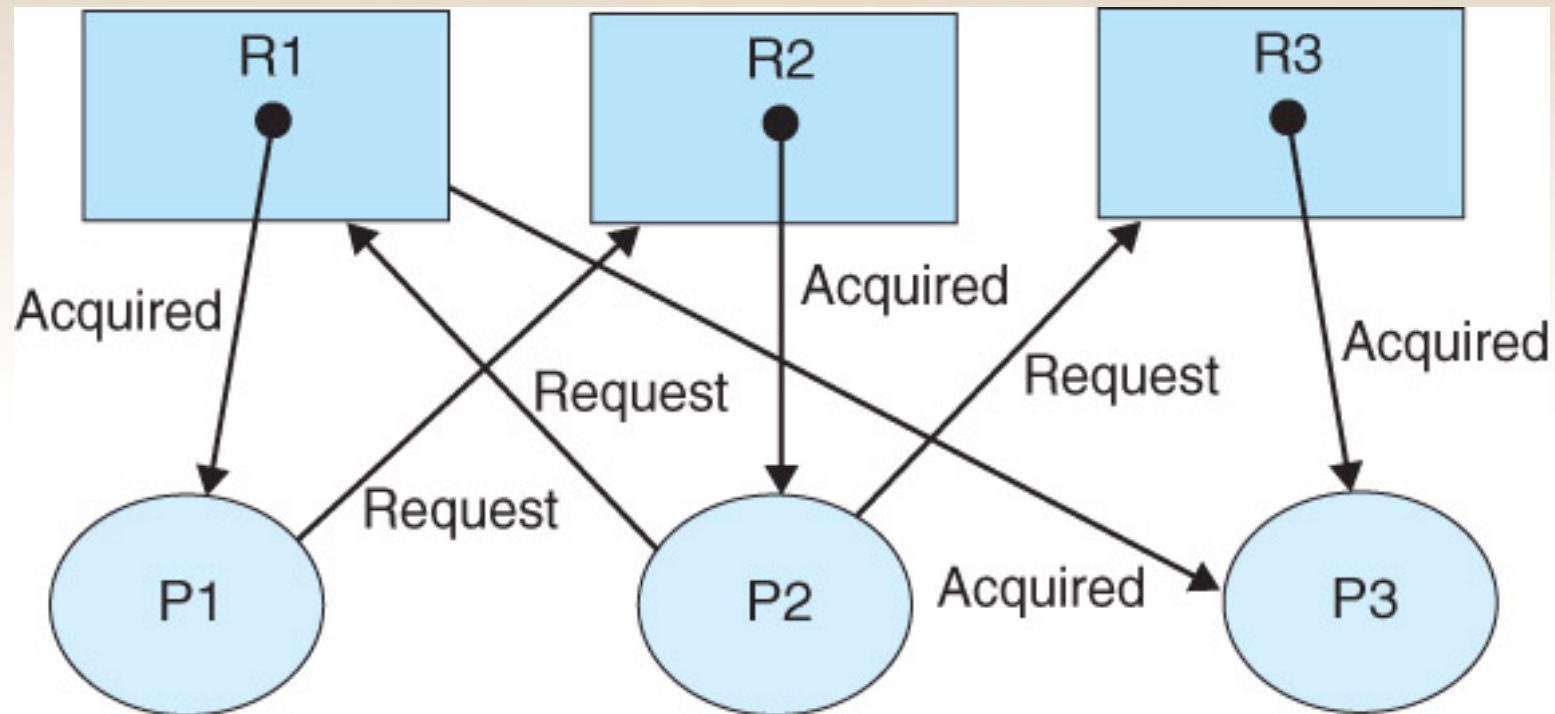
Resource Allocation Graph with a Cycle



Analysis Of This Graph

- This graph has a cycle.
- Processes p1, p2, and p3 are deadlocked.

Another Resource Allocation Graph



Analysis Of Resource Allocation Graph

- If the graph contains no cycles, no deadlock exists.
- If the graph contains cycles
 - and each resource type has only 1 instance, then a deadlock exists.
 - otherwise, a deadlock may exist.

Handling Deadlocks

General approaches

- Ensure that the system will never enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system.

Methods For Dealing with Deadlocks

- Deadlock prevention: disallow the existence of one of the four necessary conditions for deadlocks to occur.
- Deadlock avoidance: for each resources request which can be satisfied, determine whether the request should be delayed in order to avoid a possible deadlock in the future.
- Deadlock detection and recovery: detect the existence of deadlock; if it has occurred, take actions to remove it.

Deadlock Prevention

- Non-existence of the hold and wait condition
- Non-existence of the circular wait condition

Deadlock Prevention (1)

Non-existence of hold and wait:

- Allocation only if all resources requested are available
- Allocation of resources only when there is no previous allocation
- Low resource utilization
- Starvation possible

Deadlock Prevention (2)

Non-existence of circular wait:

- Impose a linear ordering of resource types.
- Processes must request resources in an increasing order of enumeration.

Disallow Circular Wait

- Impose a total ordering of all resource types, $R = \{R_1, R_2, \dots, R_m\}$.
- Define a one to one function, $F: R \rightarrow N$
- Require that each process requests resources in an increasing order of enumeration.
- Process P can request resources of type R_i , then resources of type R_j , only if:
 $F(R_j) > F(R_i)$

Example of Linear Ordering for Allocation of Resources

- A system has the following resources types:
 $R = \{\text{Disk, Tape, Printer}\}$
- The following ordering is defined for the resources with function F :
 $F(\text{Disk}) = 1; F(\text{Tape}) = 3; F(\text{Printer}) = 7$
- A process, P , has to request first an instance of Disk, then an instance of Tape, then an instance of Printer.

Deadlock Prevention (3)

- Non-existence of mutual exclusion; make each resource shareable (impossible for printers).
- Existence of preemption:
 - Process must release resources held when no further resource allocations are possible.
 - Preempt the desired resources from a waiting process which hold the resource.
 - A process will be restarted only when it can regain its old resources, and acquire the new resources requested.

Common Problems For Deadlock Prevention Methods

- Low device utilization
- Reduced system throughput

Why?

Simulation Models

Deadlock Prevention

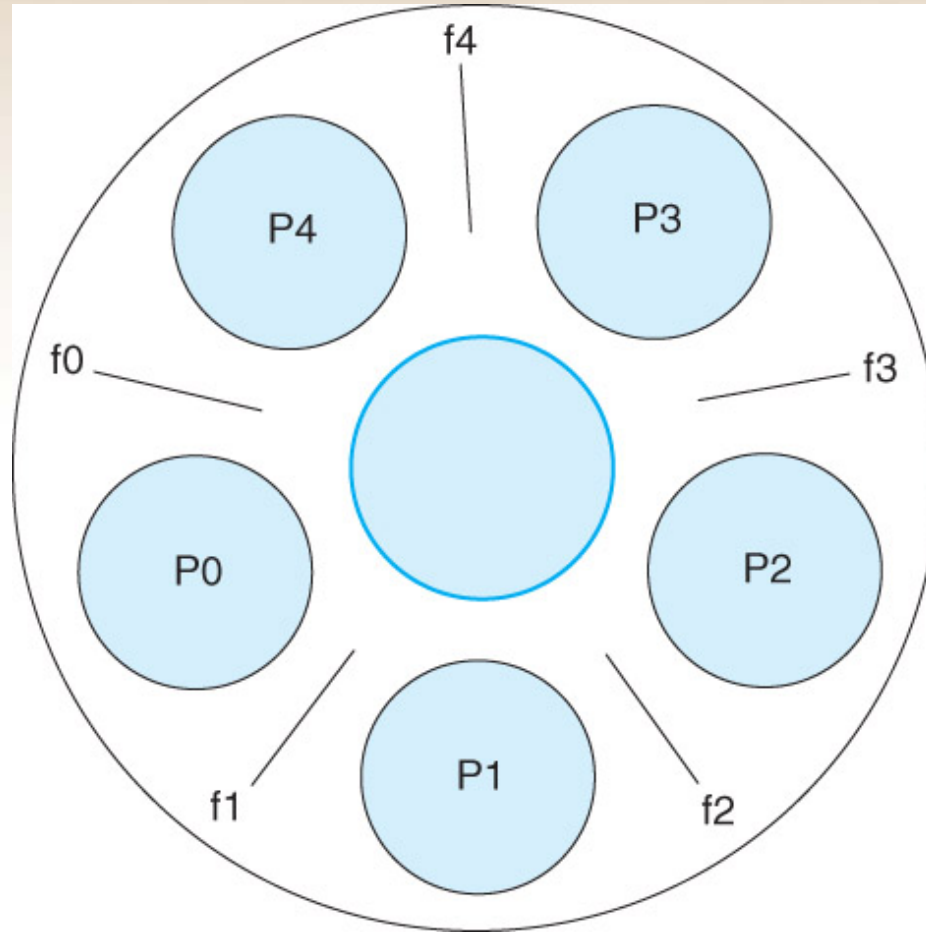
- The simulation models that apply the deadlock prevention techniques are based on the Five Philosophers Problem
- The first model always deadlocks – no deadlock techniques are applied in this model
- The second model applies the absence of the hold and wait condition
- The third model applies the absence of the circular wait condition



Dining Philosophers

- A monastery's dining table
 - Circular table
 - Five plates
 - Five forks (critical resources)
 - Plate of noodles at center of table (endless supply)
 - Philosophers, P0, P1, P2, P3, and P4, can only reach forks adjacent to their plate (left and right forks)

Dining Table



Dining-Philosophers Problem

Every philosopher needs two forks (or forks) and these must be accessed in a mutual exclusive manner

```
// Shared data
```

```
Semaphore fork[] = new Semaphore[5];
```

Philosophers Code

```
Philosopher run() {  
    while (true) {  
        // get left fork  
        fork[i].wait();  
        // get right fork  
        fork[(i + 1) % 5].wait();  
        // eat for a finite time interval  
        // return left fork  
        fork[i].signal();  
        // return right fork  
        fork[(i + 1) % 5].signal();  
        // think for finite time interval  
    }  
} // end run()
```



Why Synchronize?

- Philosophers only think and eat
- When a philosopher is hungry
 - obtains 2 forks (1 at a time)
 - Use both forks to eat
- When a philosopher has satisfied hunger returns both forks and goes back to think
- Problem: There are 5 competing philosopher processes running concurrently

Informal Solution to Deadlock

- Deadlock will not occur when the philosophers do not all sit at the table at the same time, therefore, if the five philosophers start thinking before eating, deadlock will not always happen.
- The reason for this is that the time interval to think is randomly generated so that the philosophers will not normally start to eat at the same time, consequently, deadlock will not always occur.
- This solution to deadlock shows that the difference in the ordering of events can make a big difference.
- The Java simulation model is in: `philos2.jar`; the C++ model is in `philos2.cpp`.

Model that Disallows Hold and Wait

- The two basic user-defined classes in the model are *Philoshw* and *Philosopher*.
- These classes are implemented in files `Philoshw.java` and `Philosophers.java` and are stored in the archive file `philoshw.jar`.
- The C++ implementation of this model is stored in file: `philoshw.cpp`.

Model that Disallows Circular Wait

- The two basic user-defined classes in the model are *Philoscw* and *Philosopher*.
- These classes are implemented in files `Philoscw.java` and `Philosophers.java` and are stored in the archive file `philoscw.jar`.
- The C++ implementation of this model is stored in file: `philoscw.cpp`.

Deadlock Avoidance

Requires that the system have additional a priori information available

- Each process declares the maximum need of resources
- The algorithm dynamically examines the resource-allocation state to ensure non-existence of circular-wait
- The system is kept in a safe state

Resource Allocation State

The current resource-allocation state is defined by the following data structures:

- **AVAILABLE**, a vector of (current) available resources units, for each resource type
- **A[i]**, allocated resource units for process P_i of each resource type
- **Max[i]**, maximum resource demand of process P_i , of each resource type

Informal Definition of 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
- A system is in a safe state only if there exists a safe sequence of allocations

Safe Sequence of Resource Allocation

- A system is in a safe state if a safe allocation sequence of processes $\langle P_1, P_2, \dots, P_n \rangle$ exists for the current resource allocation state
- A sequence of processes is a safe sequence for a given resource allocation state if for each process i , its maximum resource request can be satisfied by the current resources plus the total resources held by each P_j with $0 < j < i$

Unsafe State

- If a system is in an unsafe state, then some sequence of requests may lead “unavoidably” to a deadlock (i.e., the deadlock cannot be avoided by delaying the requests from the processes).
- An unsafe state is not necessarily a deadlock state and does not necessarily lead to deadlock state.

Banker's Algorithm

- There exists multiple instances of each resource type
- Each process must a priori claim maximum use of resources
- When a process requests a resource, it may have to wait
- When a process gets all its resources, it must return them in a finite period of time.

Banker's Algorithm

When a request for resources is made by process P_i :

- If $Request_i \leq Need_i$ then goto step 2. Otherwise there is an error since P_i has exceeded its maximum claim
- If $Request_i \leq available$ then goto step 3, otherwise P_i must wait
- The OS pretends to have allocated the requested resources to P_i by modifying the state as follows:
 - $AVAILABLE := AVAILABLE - Request_i$
 - $Allocation_i := Allocation_i + Request_i$
- If the resulting state is safe, P_i is allocated the resources. However, if the new state is unsafe, then P_i must wait and the state is restored

Data Structures for the Banker's Algorithm

- N is the number of processes, and m the resource types
- Available, a vector of length m indicating the number of available resources of each type.
- Max, an $n \times m$ matrix defining the maximum demand of each process, example : $\text{Max}[i,j] = k$.
- Allocation, an $n \times m$ matrix defining the number of resources of each type currently allocated to each process.
- Need, an $(n \times m)$ matrix for the remaining resource need of each process. $\text{Need}[i, j] = \text{max}[i, j] - \text{allocation}[i,j]$.
- We can treat each row in the in the matrices Allocation and Need as vectors, and associate them with process P_i .

Description of Banker's Algorithm

- A sequence $s = \langle s_1, s_2, \dots, s_n \rangle$ of all processes is safe sequence for a given resource allocation state if for $0 < i \leq n$,
$$\text{Max } [s_i] \leq \text{AVAILABLE} + \text{sum of } A[S_j], \quad 0 < j \leq i.$$
- If the resource need of process P_i is not available, then P_i could wait until all P_j has finished.

Example 1

A system exhibits the following state: there are three processes P0, P1, P2; total system resources: 12 units of magnetic tape.

Process	Max_need	Allocation
P0	10	5
P1	4	2
P2	9	2

Find out whether the system is in a safe state.

Solution Procedure

1. Compute the resources currently available, which is $(12 - 9) = 3$
2. Build the Need column with the numbers of resources that processes currently need to reach the maximum claim. These are 5, 2, and 7 (for P1, P2, and P3 respectively)
3. Select a process that can proceed to acquire its current need of resources
4. When the process releases the resources, compute the total number of available resources
5. Repeat from step 3

Final Solution

- Safe sequence of allocations found:
 <P1, P0, P2>
- Therefore, the given state is a safe state

Example 2

Assume that when in state 1, the OS allocates 1 more unit of tape. The new state is:

Process	Max_need	Allocations	Need
P0	10	5	5
P1	4	2	2
P2	9	3	6

Resources currently available: 2

Safe State?

- After process P1 completes and releases all the resources it held, the total resources available is 4, which is not sufficient to satisfy the needs of P0 or P2.
- Therefore, the state given is unsafe, no safe sequence of allocations is possible.

Example 3

A system has five processes {P0, P1, ... P4} and three resource types {A, B, C}, with 10, 5, and 7 instances respectively.

PROCESS	ALLOCATION			MAX			NEED		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	7	4	3
P1	2	0	0	3	2	2	1	2	2
P2	3	0	2	9	0	2	6	0	0
P3	2	1	1	2	2	2	0	1	1
P4	0	0	2	4	3	3	4	3	1

Solution

The system is currently in a safe state. The sequence: $\langle P1, P3, P4, P2, P0 \rangle$ satisfies the safety criteria.

Example 4 of Banker's Algorithm

Suppose P1 requests (1,0,2). The OS needs to check if $(1,0,2) \leq (3,3,2)$, which is true. The OS then pretends to grant this request, and determine if the new state is a safe state.

The sequence $\langle P1, P3, P4, P0, P2 \rangle$ is a safe sequence.

Can request for (3,3,0) by P4 be granted?

Can request for (0,2,0) by P0 be granted?

Deadlock Detection

- Allow system to enter deadlock state
- Run detection algorithm
- Recovery Scheme

Detection and Recovery

A detection and recovery scheme requires overhead that includes:

- Maintaining information and running the detection algorithm
- The potential losses inherent in recovering from deadlock

Deadlock Detection

A detection algorithm is invoked periodically to determine whether a deadlock has occurred.

- A system has a deadlock if and only if it is impossible to satisfy the resource requests of some processes.
- Formally, a system is in a deadlock-free state if there exists a deadlock-free sequence for the current resource allocation state.

Detection Approach

- The operating system can check for deadlock every time a resource is allocated; this is early detection.
- Other algorithms are used to detect cycles in the resource allocation graph. The techniques are based on incremental changes to the system state.
- The algorithms attempt to find a process with its resource requests that can be allocated with the currently available resources, the resources are allocated to the process, the resources are used by the process, and then the resources are released. This is repeated with the other processes.
- The processes that are in deadlock are identified and a procedure is initiated to stop deadlock.

Frequency of Running Deadlock Detection

- The deadlock detection algorithm can be invoked every time a process requests a resource that cannot be immediately granted (allocated). In this case, the operating system can directly identify the specific process that caused deadlock, in addition to the set of processes that are in deadlock.
- The detection algorithm is invoked periodically using a period not too long or not too short. Since deadlock reduces the CPU utilization and the system throughput, detection algorithm may be invoked when the CPU utilization drops below 40%.

System Recovery

- Abort all deadlocked processes
- Abort processes one at a time
- Preempt resources one at a time
- Rollback

Recovery: Process Termination

- Abort all deadlocked processes.
- Abort one process at a time until the deadlock cycle is eliminated.
- In which order should processes be aborted?
 - Priority of the process.
 - How long process has computed, and how much longer to completion.
 - Resources the process has used.
 - Resources process needs to complete.
 - How many processes will need to be terminated.
 - Is process interactive or batch?

Recovery (2)

- Selection of resources for preemption and processes to abort.
 - Successively preempt some resources from processes and allocate these resources to another process until the deadlock cycle is broken.
- Rollback: the system saves snapshots of a process' state at periodic intervals. To break a deadlock, the system aborts a process and later resume it at the most recent checkpoint; (technique also useful in system crashes).

Deadlock Recovery (3)

- Process termination - abort (terminate) one or more processes to break the circular-wait condition.
- Starvation is possible, the same process is selected from which to preempt resources, and it never completes.

Deadlock Detection Issues

- How often does a deadlock occur?
- How many processes will be affected by deadlock, when it happens? It is possible that the last request which could not be granted, completes a chain of waiting processes?
- Invoke the detection algorithm whenever the CPU utilization drops below 40%.
- If the detection algorithm is invoked at arbitrary points in time, there may be many cycles in the resource graph.

Combined Approach To Deadlock

Resources are partitioned into hierarchically ordered classes within each class the most appropriate technique is used. Consider a system which consists of the following classes of resources:

- Internal resources. (used by the OS, eg., PCBs)
- Central memory. Technique: preemption (swapping)
- Job resources. Assignable devices and files. technique: avoidance

Safety Algorithm

To find out whether a system is in a safe state or not.

1. Initialize $WORK := AVAILABLE$, vector of length

M $GRANT[ip] := FALSE$ for all $ip = 1, 2, \dots, N$.

2. Find i such that:

a. $GRANT[i] = FALSE$, and

b. $NEED\ i \leq WORK$.

If no such i exists, go to step 4.

3. $WORK := WORK + ALLOCATION_i$

$GRANT[i] := True$

go to step 2.

4. If $GRANT[i] = True$ for all i , then the OS is in a safe state.

Peterson's Algorithm

Initially, $\text{flag}[0], \text{flag}[1] = \text{false}$, $\text{turn} = 0$

Entry Section:

$\text{flag}[i] := \text{true};$

$j := (i + 1) \bmod 2;$

$\text{turn} := j;$

While ($\text{flag}[j]$ and $\text{turn} = j$) do skip;

Critical Section:

.

.

Exit Section:

$\text{flag}[i] := \text{false};$