Operating System

ACSE0403A

Unit: 3

Deadlock and Concurrent Processing

B Tech 4th Sem

Surbhi Jha
Assistant Professor
Department of CSE

Evaluation Scheme

Sl. No.	Subject Codes	Subject Name	Periods			Evaluation Scheme				End Semester		Total	Credit
			L	T	P	CT	TA	TOTAL	PS	TE	PE		
1	AAS0402	Engineering Mathematics-IV	3	1	0	30	20	50		100		150	4
2	AASL0401	Technical Communication	2	1	0	30	20	50		100		150	3
3	AIT0401	Software Engineering	3	0	0	30	20	50		100		150	3
4	ACSE0403A	Operating Systems	3	0	0	30	20	50		100		150	3
5	ACSE0404	Theory of Automata and Formal Languages	3	0	0	30	20	50		100		150	3
6	ACSAI0402	Database Management Systems	3	1	0	30	20	50		100		150	4
7	AIT0451	Software Engineering Lab	0	0	2				25		25	50	1
8	ACSE0453A	Operating Systems Lab	0	0	2				25		25	50	1
9	ACSAI0452	Database Management Systems Lab	0	0	2				25		25	50	1
10	ACSE0459	Mini Project using Open Technology	0	0	2				50			50	1
11	ANC0402 / ANC0401	Environmental Science*/Cyber Security*(Non Credit)	2	0	0	30	20	50		50		100	0
12		MOOCs** (For B.Tech. Hons. Degree)											
		GRAND TOTAL										**1100**	**24**

Subject Syllabus

<table>
<tr><th colspan="4">B. TECH. SECOND YEAR</th></tr>
<tr><td>Course Code</td><td>ACSE0403A</td><td>L T P</td><td>Credits</td></tr>
<tr><td>Course Title</td><td>Operating Systems</td><td>3 0 0</td><td>3</td></tr>
<tr><td colspan="4">Course objective:
The objective of the course is to provide an understanding of the basic modules and architecture of an operating system and the functions of the modules to manage, coordinate and control all the parts of the computer system. This course cover processor scheduling, deadlocks, memory management, process synchronization, system call and file system management.</td></tr>
<tr><td colspan="4">Pre-requisites:
1. Basic knowledge of computer fundamentals, Data structure and Computer organization.</td></tr>
<tr><td colspan="4">Course Contents / Syllabus</td></tr>
<tr><td>UNIT-I</td><td colspan="2">Fundamental Concepts of Operating System</td><td>8 Hours</td></tr>
<tr><td colspan="4">Introduction, Functions of Operating System, Characteristics of Operating System, Computer System Structure, Evolution of Operating Systems-Bare Machine, Single Processing, Batch Processing,Multiprogramming,Multitasking,Multithreaded,Interactive, Time sharing, Real Time System, Distributed System, Multiprocessor Systems, Multithreaded Systems, System Calls, System Programs and System Boot, Interrupt Handling, Operating System Structure- Simple structure, Layered Structure, Monolithic, Microkernel and Hybrid, System Components, Operating System Services, Case Studies: Windows, Unix and Linux.</td></tr>
<tr><td>UNIT-II</td><td colspan="2">Process Management</td><td>8 Hours</td></tr>
<tr><td colspan="4">Scheduling Concepts, Performance Criteria, Process States, Process Transition Diagram, Schedulers, Process Control Block (PCB), Process Address Space, Process Identification Information, Threads and their management, Types of Scheduling: Long Term Scheduling, Mid Term Scheduling, Short Term Scheduling, Pre-emptive and Non Pre-emptive Scheduling, Dispatcher, Scheduling Algorithm: FCFS, Non Pre-emptive SJF, Pre-emptive SJF, Non Pre-emptive Priority, Pre-emptive Priority, Round Robin, Multilevel Queue Scheduling and Multilevel Feedback Queue Scheduling.</td></tr>
<tr><td>UNIT-III</td><td colspan="2">Deadlock and Concurrent Processing</td><td>8 Hours</td></tr>
<tr><td colspan="4">Deadlock: System model, Deadlock characterization, Prevention, Avoidance and detection, Recovery from Deadlock,Principle of Concurrency, Process Synchronization, Producer / Consumer Problem, Mutual Exclusion, Critical Section Problem, Peterson's Solution, Lamport Bakery Solution, Semaphores, Test and Set Operation; Critical Section Problems and their solutions - Bound Buffer Problem, Reader-Writer Problem, Dining Philosopher Problem, Sleeping Barber Problem; Inter Process Communication Models and Schemes, Process Generation.</td></tr>
</table>

Subject Syllabus

UNIT-IV	**Memory Management**	**8 Hours**
Memory Management function, Address Binding Loading : Compile Time, Load Time and Execution Time, MMU, Types of Linking, Types of Loading, Swapping, Multiprogramming with Fixed Partitions, Multiprogramming with variable partitions, Memory Allocation: Allocation Strategies First Fit, Best Fit, and Worst Fit, Paging, Segmentation, Paged Segmentation, Virtual Memory Concepts, Demand Paging, Performance of Demand Paging, Page Replacement Algorithms: FIFO,LRU, Optimal and LFU, Belady's Anomaly, Thrashing, Cache Memory Organization, Locality of Reference.		
UNIT-V	**I/O Management and Disk Scheduling**	**8 Hours**
I/O Devices, and I/O Subsystems, I/O Buffering, I/O Ports, Disk Storage: Seek Time, Rotational Latency, Data Transfer Time, Average Access Time and Controller Time,Disk Storage Strategies, Disk Scheduling:FCFS, SSTF, SCAN, C-SCAN, LOOK and C-LOOK. Directory and Directory Structure, File		

System: File concept, File Access Mechanism: - Sequential Access, Direct Access and Index Access methods, File Allocation Method: Contiguous, Linked and Indexed, Free Space Management: -Bit Vector, Linked List, Grouping and Counting File System Implementation Issues, File System Protection and Security, RAID.

Syllabus For Unit-3

UNIT-III Deadlock and Concurrent Processing

Deadlock: System model, Deadlock characterization, Prevention, Avoidance and detection, Recovery from Deadlock, Principle of Concurrency, Process Synchronization, Producer / Consumer Problem, Mutual Exclusion, Critical Section Problem, Peterson's Solution, Lamport Bakery Solution, Semaphores, Test and Set Operation; Critical Section Problems and their solutions - Bound Buffer Problem, Reader-Writer Problem, Dining Philosopher Problem, Sleeping Barber Problem; Inter Process Communication Models and Schemes, Process Generation.

Branch wise Applications

- Airlines reservation system.
- Air traffic control system.
- Systems that provide immediate updating.
- Used in any system that provides up to date and minute information on stock prices.
- Defense application systems like RADAR.
- Networked Multimedia Systems.
- Command Control Systems.

Course Objectives

- Provide an understanding of the basic modules and architecture of an operating system and the functions of the modules to manage, coordinate and control all the parts of the computer system.
- Processor scheduling, deadlocks, memory management, process synchronization, system call and file system management.

Course Outcomes

Course outcome: After completion of this course students will be able to:

CO 1	Understand the fundamentals of an operating systems, functions and their structure and functions.	K1, K2
CO2	**Implement concept of process management policies, CPU Scheduling and thread management.**	**K5**
CO3	Understand and implement the requirement of process synchronization and apply deadlock handling algorithms.	K2,K5
CO4	Evaluate the memory management and its allocation policies.	K5
CO5	Understand and analyze the I/O management and File systems	K2, K4

Program Outcomes

1. Engineering knowledge
2. Problem analysis
3. Design/development of solutions
4. Conduct investigations of complex problems
5. Modern tool usage
6. The engineer and society
7. Environment and sustainability
8. Ethics:
9. Individual and team work
10. Communication
11. Project management and finance
12. Life-long learning

COs and POs Mapping

OPERATING SYSTEM (ACSE0403A)												
CODE	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
ACSE0403A .1	3	3	2	2	1	2	-	2	3	2	2	3
ACSE0403A .2	3	3	3	2	2	3	-	2	3	1	1	3
ACSE0403A .3	**3**	**3**	**2**	**2**	**2**	**2**	**-**	**2**	**2**	**3**	**1**	**3**
ACSE0403A .4	3	2	2	3	1	2	-	1	2	1	2	3
ACSE0403A .5	3	1	2	2	2	2	-	1	2	2	2	3
Average	3	2.4	2.2	2.2	1.6	2.2	-	1.8	2.2	1.8	1.6	3

Program Specific Outcomes(PSOs)

On successful completion of B. Tech. (I.T.) Program, the Information Technology graduates will be able to:

- **PSO1:-** Work as a software developer, database administrator, tester or networking engineer for providing solutions to the real world and industrial problems.
- **PSO2:**- Apply core subjects of information technology related to data structure and algorithm, software engineering, web technology, operating system, database and networking to solve complex IT problems.
- **PSO3:**-Practice multi-disciplinary and modern computing techniques by lifelong learning to establish innovative career.
- **PSO4:-**Work in a team or individual to manage projects with ethical concern to be a successful employee or employer in IT industry.

COs and PSOs Mapping

Course Outcomes	Program Specific Outcomes			
	PSO1	PSO2	PSO3	PSO4
ACSE0403A .1	**2**	**1**	**2**	**2**
ACSE0403A .2	2	2	1	2
ACSE0403A .3	2	3	3	2
ACSE0403A .4	2	2	1	2
ACSE0403A .5	2	2	2	2
Average	2	2	1.8	2

Program Educational Objectives (PEOs)

- **PEO1:**Apply sound knowledge in the field of information technology to fulfill the needs of IT industry.
- **PEO2:**Design innovative and interdisciplinary systems through latest digital technologies.
- **PEO3:**Inculcate professional – social ethics, team work and leadership for serving the society.
- **PEO4:**Inculcate lifelong learning in the field of computing for successful career in organizations and R&D sectors.

Faculty wise Result Analysis

Semester & Section	Subject Code	Result

End Semester Question Paper Templates (Offline Pattern/Online Pattern

Printed page: ....

Subject Code: ....

Roll No:

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY ,GREATER NOIDA

(An Autonomous Institute Affiliated to AKTU, Lucknow)

B.Tech/B.Voc./MBA/MCA/M.Tech (Integrated)

(SEM: ..... THEORY EXAMINATION (2020-2021)

Subject ....

Time: 3 Hours

Max. Marks:100

General Instructions:

- All questions are compulsory. Answers should be brief and to the point.
- This Question paper consists of ....pages & ...8.........questions.
- It comprises of three Sections, A, B, and C. You are to attempt all the sections.
- Section A -Question No- 1 is objective type questions carrying 1 mark each, Question No- 2 is very short

End Semester Question Paper Templates (Offline Pattern/Online Pattern

- **Section B** - Question No-3 is Long answer type -I questions with external choice carrying 6 marks each. You need to attempt any five out of seven questions given.
- **Section C** - Question No. 4-8 are Long answer type -II (within unit choice) questions carrying 10 marks each. You need to attempt any one part *a or b.*
- Students are instructed to cross the blank sheets before handing over the answer sheet to the invigilator.
- No sheet should be left blank. Any written material after a blank sheet will not be evaluated/checked.

End Semester Question Paper Templates (Offline Pattern/Online Pattern

		SECTION – A		CO
1.	Attempt all parts-		[10×1=10]	
	1-a.	Question-	(1)	
	1-b.	Question-	(1)	
	1-c.	Question-	(1)	
	1-d.	Question-	(1)	
	1-e.	Question-	(1)	
	1-f.	Question-	(1)	
	1-g.	Question-	(1)	
	1-h.	Question-	(1)	
	1-i.	Question-	(1)	
	1-j.	Question-	(1)	

End Semester Question Paper Templates (Offline Pattern/Online Pattern

2.	Attempt all parts-		[5×2=10]	CO
	2-a.	<u>Question-</u>	(2)	
	2-b.	<u>Question-</u>	(2)	
	2-c.	<u>Question-</u>	(2)	
	2-d.	<u>Question-</u>	(2)	
	2-e.	<u>Question-</u>	(2)	

End Semester Question Paper Templates (Offline Pattern/Online Pattern

SECTION – B				CO
3.	Answer any five of the following-		[5×6=30]	
	3-a.	Question-	(6)	
	3-b.	Question-	(6)	
	3-c.	Question-	(6)	
	3-d.	Question-	(6)	
	3-e.	Question-	(6)	
	3-f.	Question-	(6)	
	3-g.	Question-	(6)	

End Semester Question Paper Templates (Offline Pattern/Online Pattern

SECTION – C				CO
4	Answer any **one** of the following-		[5×10=50]	
	4-a.	**Question-**	(10)	
	4-b.	**Question-**	(10)	
5.	Answer any one of the following-			
	5-a.	**Question-**	(10)	
	5-b.	**Question-**	(10)	

End Semester Question Paper Templates (Offline Pattern/Online Pattern

6.	Answer any one of the following-			
	6-a.	Question-	(10)	
	6-b.	Question-	(10)	
7.	Answer any one of the following-			
	7-a.	Question-	(10)	
	7-b.	Question-	(10)	
8.	Answer any one of the following-			
	8-a.	Question-	(10)	
	8-b.	Question-	(10)	

Prerequisite and Recap

Prerequisite

- First get your computer hardware basics cleared.
- Digital logic and it's design (Basics will make understand storing memory and page faults, Difference between ram and rom. etc.,)
- Computer Organization and Architecture(Design of Computer architecture will help you understand computer peripherals and its storages, accessing them in operating system)
- Strong programming skills (Knowledge of C).

Recap

- To Understood the concept of process and process state
- To understood and analyze the various the of CPU Scheduling Algorithm
- To Understood the concept thread and their uses

Objective of Unit -3

- To develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks.
- To present a number of different methods for preventing or avoiding deadlocks in a computer system.
- To describe the various features of processes, including scheduling, creation, and termination.
- To explore interprocess communication using shared memory and message passing.
- To describe communication in client-server systems.
- To introduce the critical-section problem, whose solutions can be used to ensure the consistency of shared data.
- To present both software and hardware solutions of the critical-section problem.
- To examine several classical process-synchronization problems.

Unit- Recap

- To understand Processes and process state
- To analyze the critical section problem with various algorithm
- Illustrate the methods of process management, process synchronization.
- Demonstrate the various classical problem implemented by semaphore
- To understand the hardware synchronization and monitor
- To understand of deadlocks, which prevent sets of concurrent processes from completing their tasks.
- To understand different methods for preventing or avoiding deadlocks in a computer system.

NIET Greater Noida

Brief Introduction about the subject with video

An operating system acts as an intermediary between the user of a computer and computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs conveniently and efficiently

YouTube/other Video Links

- https://www.youtube.com/playlist?list=PLmXKhU9FNesSFvj6gASuWmQd23Ul5omtD

Prerequisite and Recap

- Basic knowledge of computer fundamentals.
- Basic knowledge of computer organization.
- Memory hierarchy
- Cache Organization
- Interrupt
- Registers
- Associative memory

Unit-3 Content

- Deadlock
- Deadlock characterization
- Deadlock Prevention
- Deadlock Avoidance
- Recovery from Deadlock
- Principle of Concurrency
- Process Synchronization & it’s Problems
- Mutual Exclusion

Unit-3 Content

- Critical Section Problem
- Peterson’s Solution
- Lamport Bakery Solution
- Semaphores
- Test and Set Operation
- Critical Section Problems and their solutions
- Inter Process Communication Models and Schemes
- Process Generation

Topic Objective

Deadlock	Understand the concept of Deadlock
Principle of Concurrency	Understand the concept of concurrent Processes
Process Synchronization	Understand the concept of Process Synchronization
Mutual Exclusion	Understand the concept of Mutual Exclusion
Semaphores	Understand the concept of Semaphore Solution
Critical Section	Understand the concept of Critical Section
Inter Process Communication	Understand the concept of IPC

Unit-3 Objective

After going through this unit, you should be able to:

- Understand the concept of Deadlock.
- Understand the concept of Process Synchronization
- Understand the concept of Critical Section
- Understand the concept of Inter Process Communication
- Understand the concept of Process Generation

Topic mapping with CO

- Deadlock (CO3)
- Deadlock characterization (CO3)
- Deadlock Prevention (CO3)
- Deadlock Avoidance (CO3)
- Recovery from Deadlock (CO3)
- Principle of Concurrency (CO3)
- Process Synchronization & it's Problems (CO3)
- Mutual Exclusion (CO3)

Topic mapping with CO

• Critical Section Problem	(CO3)
• Peterson's Solution	(CO3)
• Lamport Bakery Solution	(CO3)
• Semaphores	(CO3)
• Test and Set Operation	(CO3)
• Critical Section Problems and their solutions	(CO3)
• Inter Process Communication Models and Schemes	(CO3)
• Process Generation	(CO3)

Deadlock(C03)

- In a multiprogramming system, processes request resources. If those resources are being used by other processes then the process enters a waiting state. However, if other processes are also in a waiting state, we have deadlock.

- A set of processes is in a deadlock state if every process in the set is waiting for an event (release) that can only be caused by some other process in the same set.

Example

-System has 2 disk drives

-P1 and P2 process each hold one disk drive and each needs another one

Deadlock Characterization(CO3)

Deadlock can arise if four conditions hold simultaneously

- **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 .
- **No preemption**: a resource can be released only voluntarily by the process holding it, after that process has completed its task .
- **Circular wait:** there exists a set {P0, P1, …, Pn} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, …, Pn–1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0.

Resource-Allocation Graph(CO3)

A resource allocation graph is a set of vertices V and a set of edges E such that:

- V is partitioned into two types:

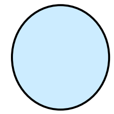
    P = {P1, P2, …, Pn}, the set consisting of all the processes in the system

    R = {R1, R2, …, Rm}, the set consisting of all resource types in the system

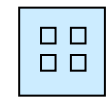
- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment edge** – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph(CO3)

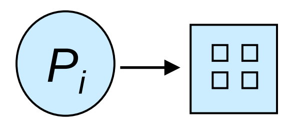
- Process



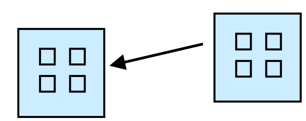
- Resource Type with 4 instances



- P_i requests instance of R_j

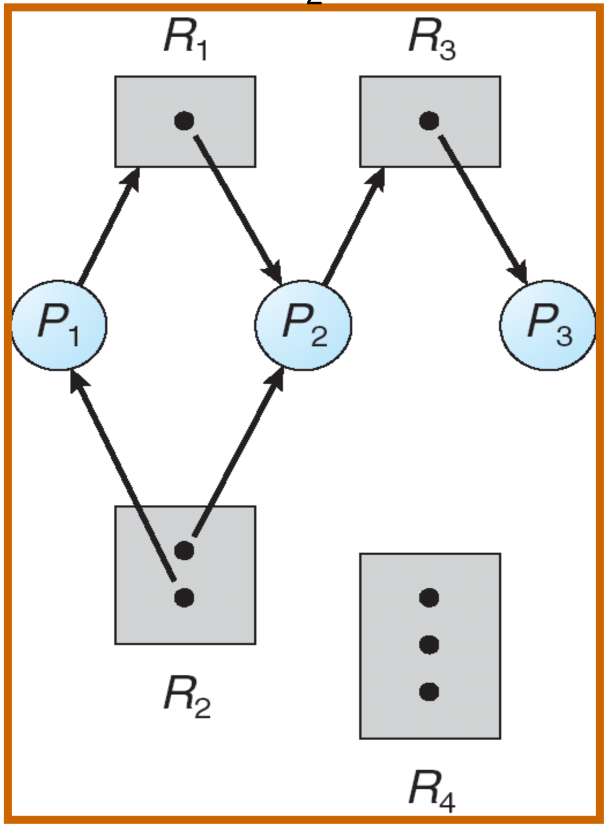


- P_i is holding an instance of R_j

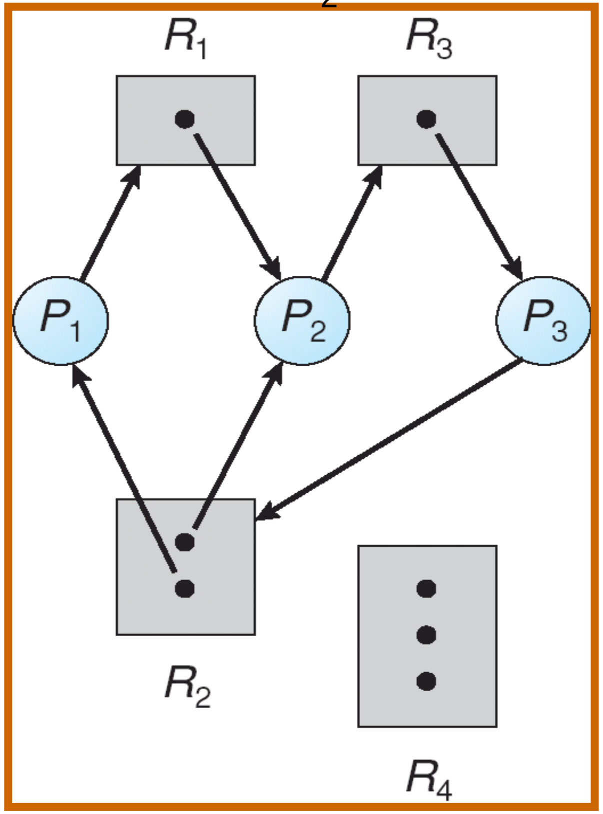


Resource Allocation Graph With A Deadlock(CO3)

Before P_3 requested an instance of R_2



After P_3 requested an instance of R_2



A Cycle In the Graph May cause Deadlock

Summary(CO3)

- 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 deadlock

Methods for Handling Deadlocks(CO3)

- **Prevention**
  - Ensure that the system will never enter a <u>deadlock</u> state
- **Avoidance**
  - Ensure that the system will never enter an <u>unsafe</u> state
- **Detection**
  - Allow the system to enter a deadlock state and then recover
- **Do Nothing**
  - Ignore the problem and let the user or system administrator respond to the problem; used by most operating systems, including Windows and UNIX

Deadlock Prevention(CO3)

•To prevent deadlock, we can restrain the ways that a request can be made

•Do not allow one of the four conditions to occur

•**Mutual Exclusion :**

- if no resource were ever assigned to a single process exclusively ,we would never have deadlock.
- Shared entities (read only files) don't need mutual exclusion (and aren't susceptible to deadlock.)
- Prevention not possible, since some devices are naturally non-sharable

Deadlock Prevention(CO3)

Hold And Wait :

we must guarantee that whenever a process requests a resource, it does not hold any other resources

- Require a process to request and be allocated all its resources before it begins execution
- allow a process to request resources only when the process has none
- Wait time out

Result: Low resource utilization; starvation possible

Deadlock Prevention(CO3)

No Preemption :

- If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
- Pi → Rj → Pj → Rk
- A process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.
- Allow preemption - if a needed resource is held by another process, which is also waiting on some resource, steal it. Otherwise wait.

Circular Wait :

- Impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

Deadlock Avoidance(CO3)

- Requires that the system has some additional a priori information available
- Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition
- Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes

Safe State(CO3)

•When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state

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

That is:

- •If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished
- •When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
- •When P_i terminates, P_{i+1} can obtain its needed resources, and so on

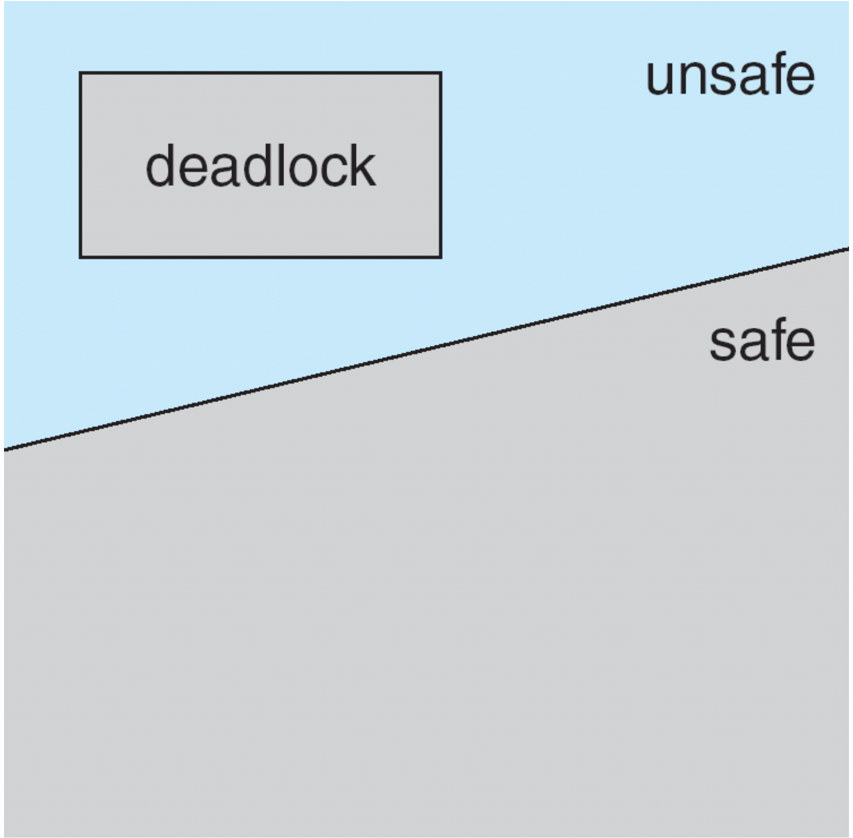
Safe State(CO3)

- If a system is in safe state $\Rightarrow$ no deadlocks

- If a system is in unsafe state $\Rightarrow$ possibility of deadlock

- Avoidance $\Rightarrow$ ensure that a system will never enter an unsafe state.

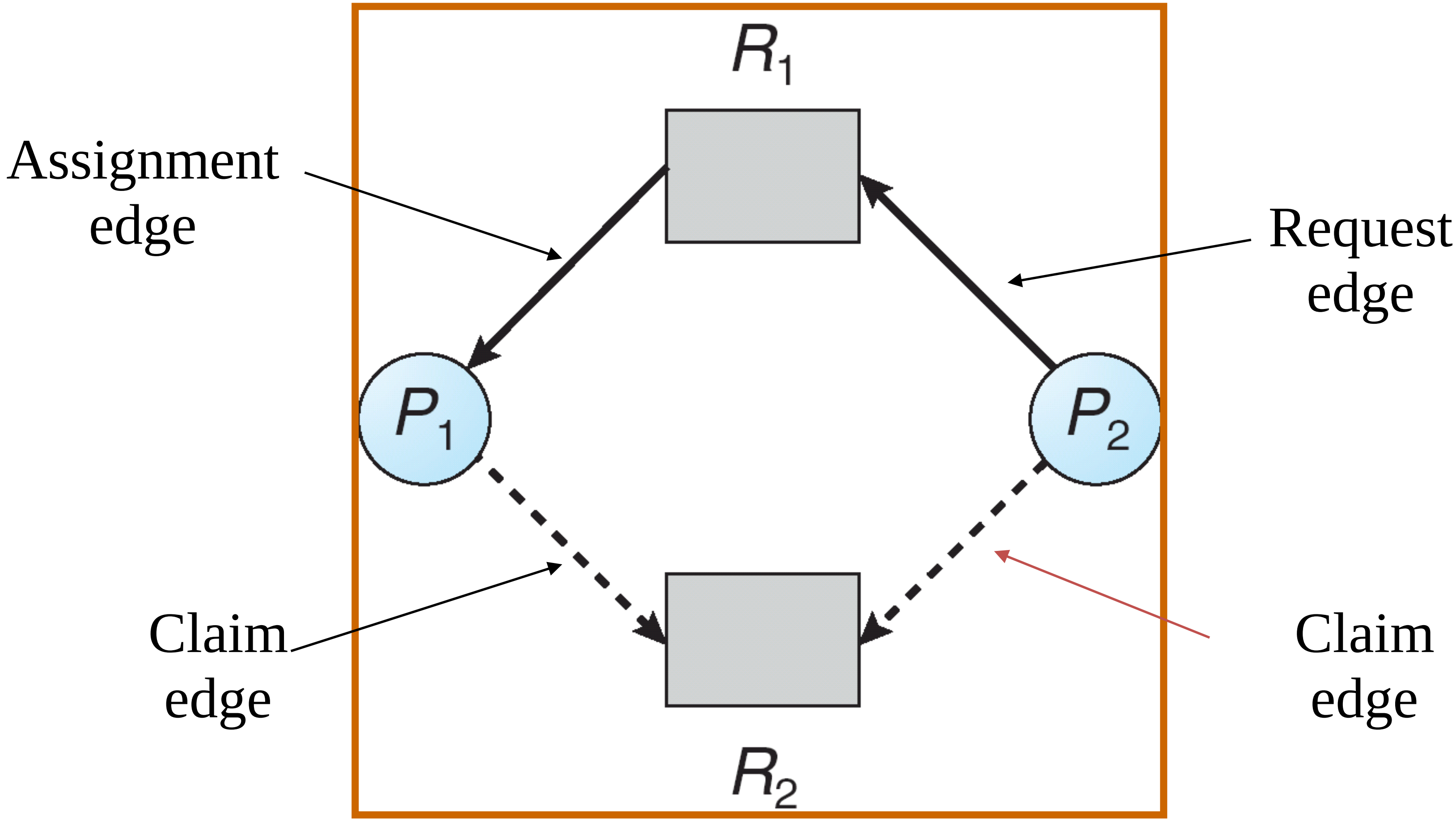
Safe State(CO3)



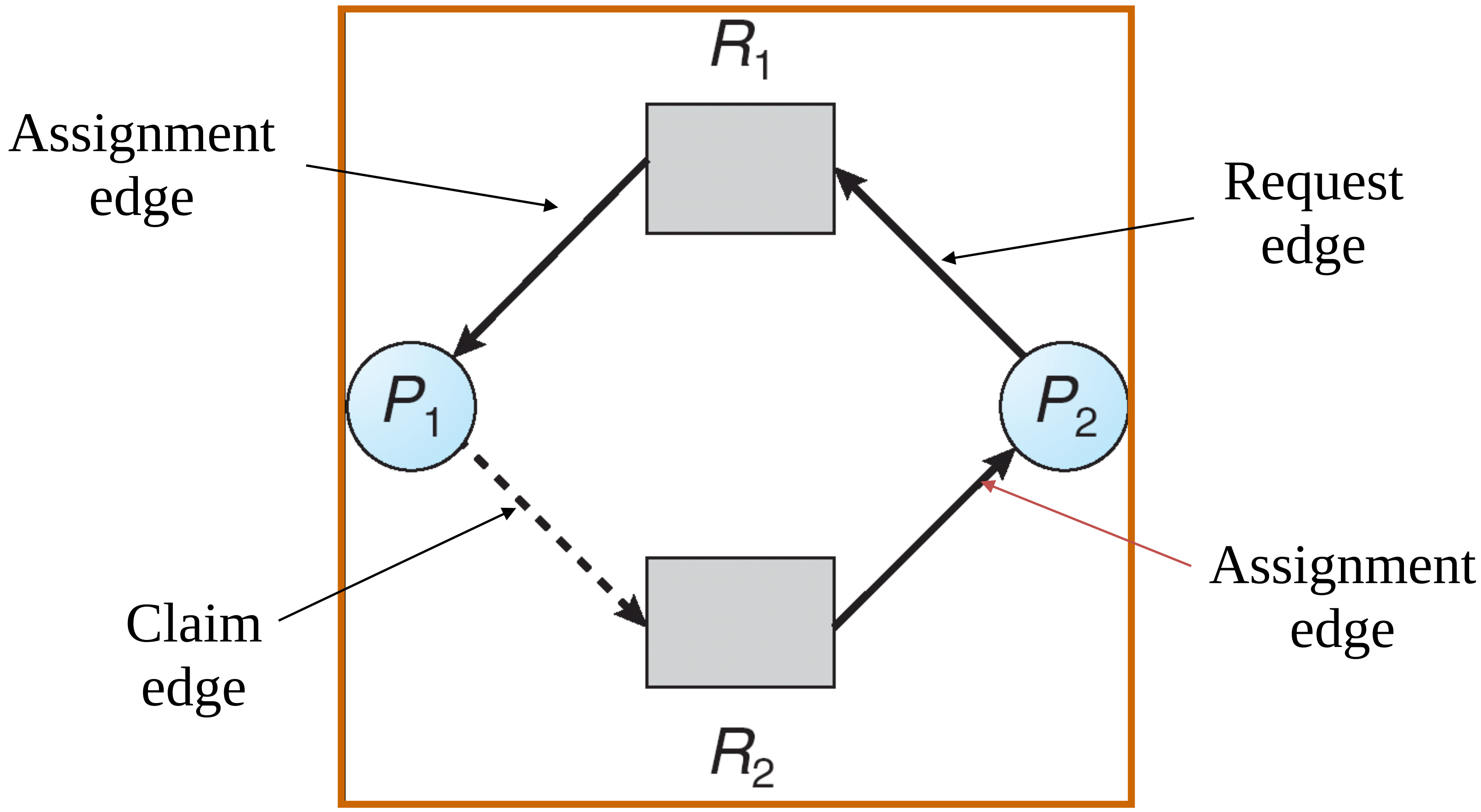
Avoidance algorithms(CO3)

- For a single instance of a resource type, use a resource-allocation graph
- For multiple instances of a resource type, use the banker's algorithm

Resource-Allocation Graph with Claim Edges(CO3)



Unsafe State In Resource-Allocation Graph (CO3)



Resource-Allocation Graph Algorithm(CO3)

- Suppose that process P_i requests a resource R_j
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph

Banker's Algorithm(CO3)

- Multiple instances
- Each process must a priori claim maximum use
- 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 amount of time

Data Structures for the Banker's Algorithm (CO3)

Let n = number of processes, and m = number of resources types.

- **Available**: Vector of length m. If available [j] = k, there are k instances of resource type R_j available.

- **Max**: n x m matrix. If Max [i,j] = k, then process P_i may request at most k instances of resource type R_j.

- **Allocation**: n x m matrix. If Allocation[i,j] = k then P_i is currently allocated k instances of $R_{j.}$

- **Need**: n x m matrix. If Need[i,j] = k, then P_i may need k more instances of R_j to complete its task.

Safety Algorithm(CO3)

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

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

2.Find an i such that both:

(a) Finish [i] = false
(b) $Need_i \leq Work$
If no such i exists, go to step 4

3. $Work = Work + Allocation_i$

Finish[i] = true
go to step 2

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

Resource-Request Algorithm for Process P_i(CO3)

Request$_i$ = request vector for process **P**$_i$. If **Request**$_i$ **[j] = k** then process **P**$_i$ wants **k** instances of resource type **R**$_j$

1. 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 not available
3. Pretend to allocate requested resources to **P**$_i$ by modifying the state as follows:

$$\textbf{Available} = \textbf{Available} - \textbf{Request}_i;$$
$$\textbf{Allocation}_i = \textbf{Allocation}_i + \textbf{Request}_i;$$
$$\textbf{Need}_i = \textbf{Need}_i - \textbf{Request}_i;$$

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

Example of Banker's Algorithm(CO3)

- 5 processes P_0 through P_4;

  3 resource types:

  A (10 instances), *B* (5 instances), and *C* (7 instances)

- Snapshot at time T_0:

	Allocation	Max	Available
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Example (Cont.)(CO3)

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

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

- The system is in a safe state since the sequence < P_1, P_3, P_4, P_2, P_0> satisfies safety criteria

P_1 Request (1,0,2)

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

	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Executing safety algorithm shows that sequence < **P_1, P_3, P_4, P_0, P_2**> satisfies safety requirement

Deadlock Detection(CO3)

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

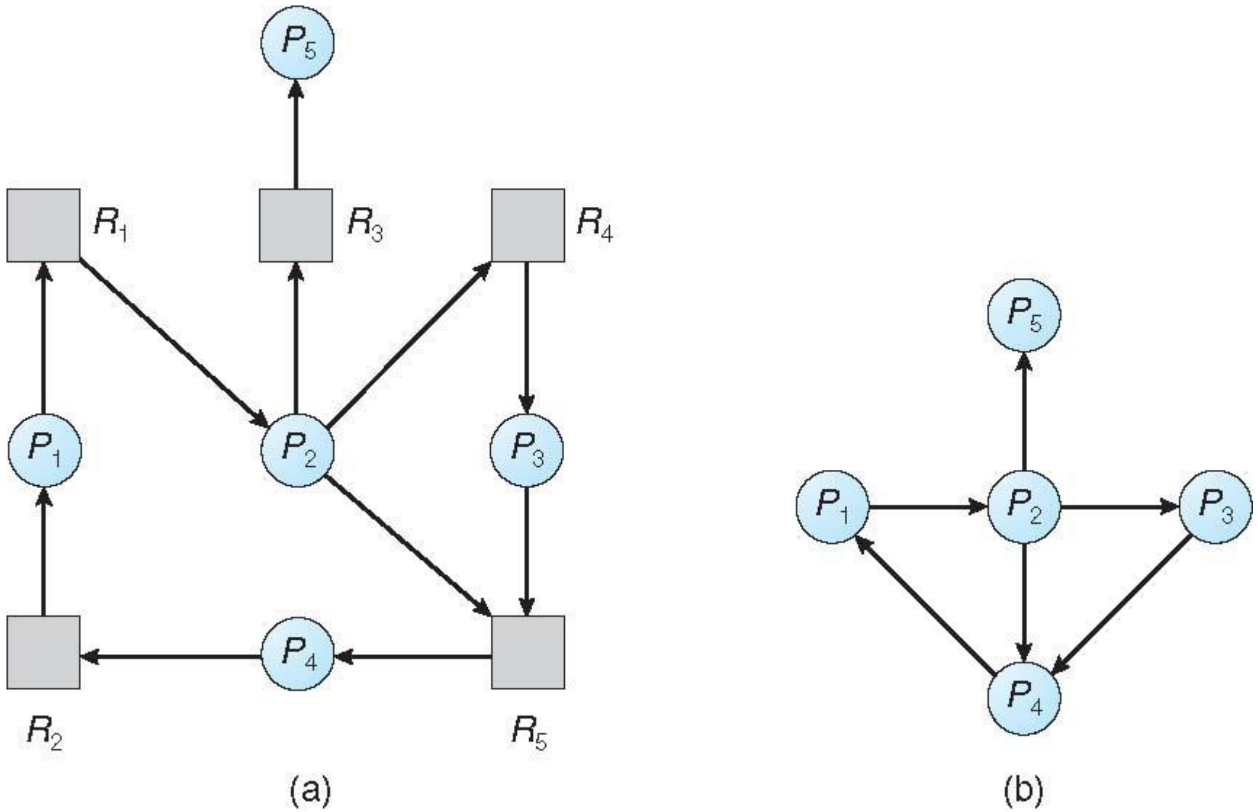
Single Instance of Each Resource Type(CO3)

- Maintain wait-for graph
  - Nodes are processes
  - $P_i \rightarrow P_j$ if P_i is waiting for P_j

- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock

- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph

Single Instance of Each Resource Type(CO3)



Resource-Allocation Graph   Corresponding wait-for graph

Several Instances of a Resource Type(CO3)

- **Available*:*** A vector of length *m* indicates the number of available resources of each type
- **Allocation*:*** An *n* x *m* matrix defines the number of resources of each type currently allocated to each process
- **Request*:*** An *n* x *m* matrix indicates the current request of each process. If *Request* [*i*][*j*] = *k*, then process P_i is requesting *k* more instances of resource type R_j.

Detection Algorithm(CO3)

1. Let **Work** and **Finish** be vectors of length **m** and **n**, respectively Initialize:
   (a) **Work = Available**
   (b) For **i = 1,2, …, n**, if $\mathbf{Allocation_i \neq 0}$, then
   Finish[i] = false; otherwise, **Finish[i] = true**
2. Find an index **i** such that both:
   (a) **Finish[i] == false**
   (b) $\mathbf{Request_i \leq Work}$
   If no such **i** exists, go to step 4
3. $\mathbf{Work = Work + Allocation_i}$
   Finish[i] = true
   go to step 2
4. If **Finish[i] == false**, for some **i**, $1 \leq \mathbf{i} \leq \mathbf{n}$, then the system is in deadlock state. Moreover, if **Finish[i] == false**, then $\mathbf{P_i}$ is deadlocked

Recovery from Deadlock(CO3)

Process Termination(simply kill one or more processes in order to break the circular wait) :

Two methods can be applied for process termination:

•Abort all deadlocked processes: This will break the deadlock cycle but at a great expense, since those processes may have completed for a long period of time and the results of these partial computations must be discarded and probably recomputed later.

•Abort one process at a time until the deadlock cycle is eliminated: This method increases overhead since after each process is killed a deadlock detection algorithm must be invoked to determine whether any processes are still deadlocked.

In which order should we choose to abort?

1. Priority of the process
2. How long process has computed, and how much longer to completion
3. Resources the process has used
4. Resources process needs to complete
5. How many processes will need to be terminated
6. Is process interactive or batch?

Recovery from Deadlock(CO3)

Resource Preemption(To eliminate deadlock we can preempt since resources from processes and give these resources to other processes until the deadlock cycle breaks):

It involves three issues:

•**Selecting a victim** – we must determine which resources and which processes are to be preempted and in which order to minimize cost.

•**Rollback** – we must determine what should be done with the process from which resources are preempted.

Solution: return to some safe state, restart process for that state

•**Starvation** – In a system it may happen that resources are always preempted for some process. As a result this process never complete its task. This situation is called starvation and needs to be avoided.

Solution: We must ensure that a process can be picked as a victim only a finite number of times.

same process may always be picked as victim, include number of rollback in cost factor

Cooperating Processes(CO3)

- Independent process cannot affect or be affected by the execution of another process
- Cooperating process (Dependent Process) can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

Producer-Consumer Problem(CO3)

- **Unbounded-buffer** places no practical limit on the size of buffer.
- **Bounded-buffer** assumes that there is a fixed buffer size.

Producer code

```
while (true) {
/* produce an item in next produced */
    while (counter == BUFFER_  SIZE) ;
/* do nothing */
buffer[in] = next_produced;
 in = (in + 1) %BUFFER_SIZE;
        counter++;
}
```

Producer-Consumer Problem(CO3)

Consumer code

```
while (true) {
while (counter == 0)
; /* do nothing */
next_consumed = buffer[out];
out = (out + 1) % BUFFER_SIZE;
            counter--;
/* consume the item in next consumed */
}
```

Race Condition(CO3)

- **counter++** could be implemented as

register1 = counter

register1 = register1 + 1

counter = register1

- **counter--** could be implemented as

  register2 = counter

  register2 = register2 - 1

  counter = register2

- Consider this execution interleaving with "count = 5" initially:

  S0: producer execute **register1 = counter** {register1 = 5}

S1: producer execute **register1 = register1 + 1** {register1 = 6}

S2: consumer execute **register2 = counter** {register2= 5}

S3: consumer execute **register2 = register2 – 1**{register2 = 4}

S4: producer execute **counter = register1** {counter = 6}

S5: consumer execute **counter = register2** {counter = 4}

Critical Section Problem(CO3)

- Consider system of **n** processes {$\mathbf{p_0, p_1, \ldots p_{n-1}}$}
- Each process has **critical section** segment of code

  Process may be changing common variables, updating table, writing file, etc

  When one process in critical section, no other may be in its critical section
- **Critical section problem** is to design protocol to solve this
- Each process must ask permission to enter critical section in **entry section**, may follow critical section with **exit section**, then **remainder section**

Critical Section(CO3)

General structure of process P_i

```
do {

    entry section

        critical section

    exit section

        remainder section

} while (true);
```

Algorithm for Process P_i(CO3)

```
do {
while (turn == j);
critical section
 turn = j;
remainder section
 } while (true);
```

Solution to Critical-Section Problem(CO3)

1. **Mutual Exclusion** - If process P_i is executing in its critical section, then no other processes can be executing in their critical sections

2. **Progress** - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely

3. **Bounded Waiting** -A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted

Peterson's Solution(CO3)

- Good algorithmic description of solving the problem
- Two process solution
- The two processes share two variables:
  - **int turn;**
  - **Boolean flag[2]**
- The variable **turn** indicates whose turn it is to enter the critical section
- The **flag** array is used to indicate if a process is ready to enter the critical section. **flag[i] = *true*** implies that process $\mathbf{P_i}$ is ready!

Algorithm for Process P_i(CO3)

```
do{
flag[i] = true;
 turn = j;
while (flag[j] && turn = = j);
 critical section
flag[i] = false;
remainder section
 } while (true);
```

Provable that the three CS requirement are met:

1. Mutual exclusion is preserved

   P_i enters CS only if:

   either **flag[j] = false** or **turn = i**

2. Progress requirement is satisfied
3. Bounded-waiting requirement is met

Semaphore(CO3)

- Synchronization tool that provides more sophisticated ways for process to synchronize their activities.
- Semaphore *S* – integer variable
- Can only be accessed via two indivisible (atomic) operations

wait() and signal()

Definition ofthewait()operation

```
wait(S) {
while (S <= 0)
; // busy wait   S--;
}
```

Definition of the signal() operation signal(S) {

```
S++;
}
```

Semaphore Usage(CO3)

- **Counting semaphore** – integer value can range over an unrestricted domain
- **Binary semaphore** – integer value can range only between 0 and 1
- Can solve various synchronization problems
- Consider P_1 and P_2 that require S_1 to happen before S_2

Create a semaphore "**synch**" initialized to 0

P1:

S_1;

signal(synch); P2:

wait(synch); **S_2;**

Can implement a counting semaphore ***S*** as a binary semaphore

Semaphore Implementation(CO3)

- Must guarantee that no two processes can execute the **wait()**and **signal()** on the same semaphore at the same time
- Thus, the implementation becomes the critical section problem where the **wait** and **signal** code are placed in the critical section
  - Could now have **busy waiting** in critical section implementation
    - But implementation code is short
    - Little busy waiting if critical section rarely occupied
- Note that applications may spend lots of time in critical sections and therefore this is not a good solution

Classical Problems of Synchronization(CO3)

- Classical problems used to test newly-proposed synchronization schemes
  - Bounded-Buffer Problem
  - Readers and Writers Problem
  - Dining-Philosophers Problem

Bounded-Buffer Problem(CO3)

- ***n*** buffers, each can hold one item
- Semaphore **mutex**initialized to the value 1
- Semaphore **full**initialized to the value 0
- Semaphore **empty** initialized to the value n

Bounded Buffer Problem (Cont.) (CO3)

The structure of the producer process

```
do {
...
/* produce an item in next_produced */
...
wait(empty);  wait(mutex);
...
/* add next produced to the buffer */
...
signal(mutex);  signal(full);
}while(true);
```

Bounded Buffer Problem (Cont.) (CO3)

The structure of the consumer process

```
do {
      wait(full);  wait(mutex);
...
/* remove an item from buffer to next_consumed */
...
signal(mutex);  signal(empty);
...
/* consume the item in next consumed */
...
}while(true);
```

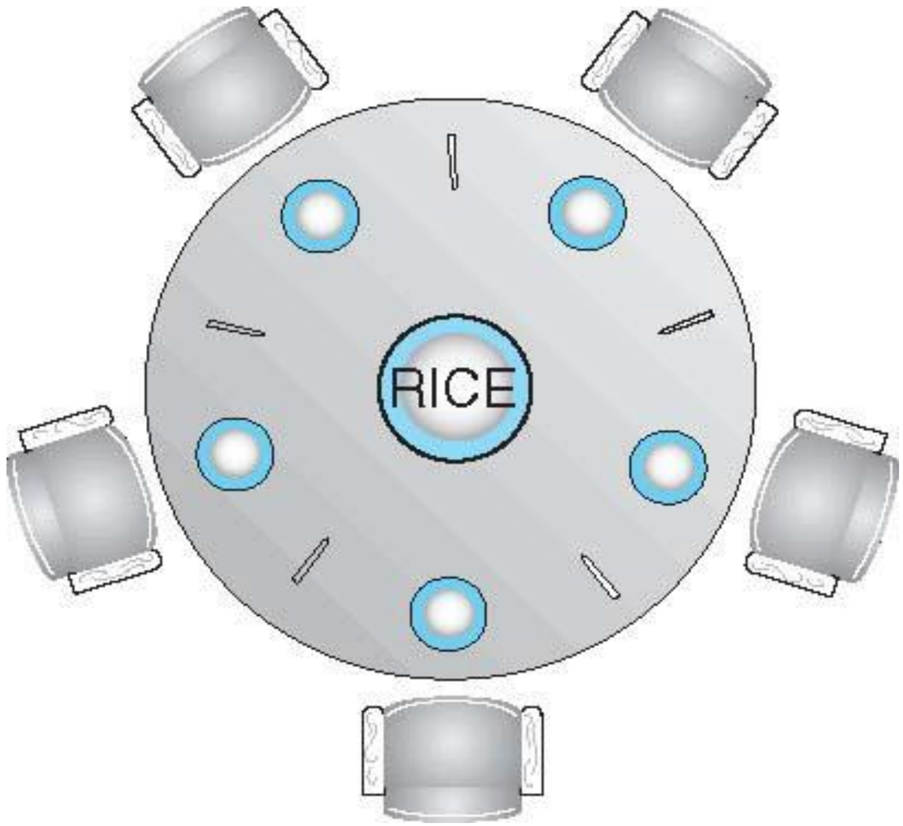
Readers-Writers Problem(CO3)

- A data set is shared among a number of concurrent processes
  - Readers – only read the data set; they do ***not*** perform any updates
  - Writers – can both read and write
- Problem – allow multiple readers to read at the same time
  - Only one single writer can access the shared data at the same time
- Several variations of how readers and writers are considered -all involve some form of priorities
- Shared Data
  - Data set
  - Semaphore **rw_mutex** initialized to 1
  - Semaphore **mutex** initialized to 1
  - Integer **read_count** initialized to 0

The structure of a writer process

```
Do
{
wait(rw_mutex);
    ...
/* writing is performed */
    ...
signal(rw_mutex);
}
while (true);
```

Dining-Philosophers Problem(CO3)



Dining-Philosophers Problem(CO3)

- Philosophers spend their lives alternating thinking and eating
- Don't interact with their neighbors, occasionally try to pick up 2 chopsticks (one at a time) to eat from bowl
  - Need both to eat, then release both when done
- In the case of 5 philosophers
  - Shared data
    - Bowl of rice (data set)
    - Semaphore chopstick [5] initialized to 1

Dining-Philosophers Problem Algorithm(CO3)

The structure of Philosopher *i*:

```
do {
wait (chopstick[i] );
wait (chopStick[ (i + 1) % 5] );

//   eat  signal (chopstick[i] );

signal (chopstick[ (i + 1) % 5] );

//   think


} while (TRUE);
```

What is the problem with this algorithm?

Dining-Philosophers Problem Algo. (Cont) (CO3)

- Deadlock handling
  - Allow at most 4 philosophers to be sitting simultaneously at the table.
  - Allow a philosopher to pick up the forks only if both are available (picking must be done in a critical section.
  - Use an asymmetric solution-- an odd-numbered philosopher picks up first the left chopstick and then the right chopstick. Even-numbered philosopher picks up first the right chopstick and then the left chopstick.

Problems with Semaphores(CO3)

- Incorrect use of semaphore operations:
  - signal (mutex) .... wait (mutex)
  - wait (mutex) ... wait (mutex)
  - Omitting of wait (mutex) or signal (mutex) (or both)
- Deadlock and starvation are possible.

Inter-process Communication(CO3)

Cooperating processes need **interprocess communication** (IPC) mechanism that will allow them to exchange data and information

Two models of IPC

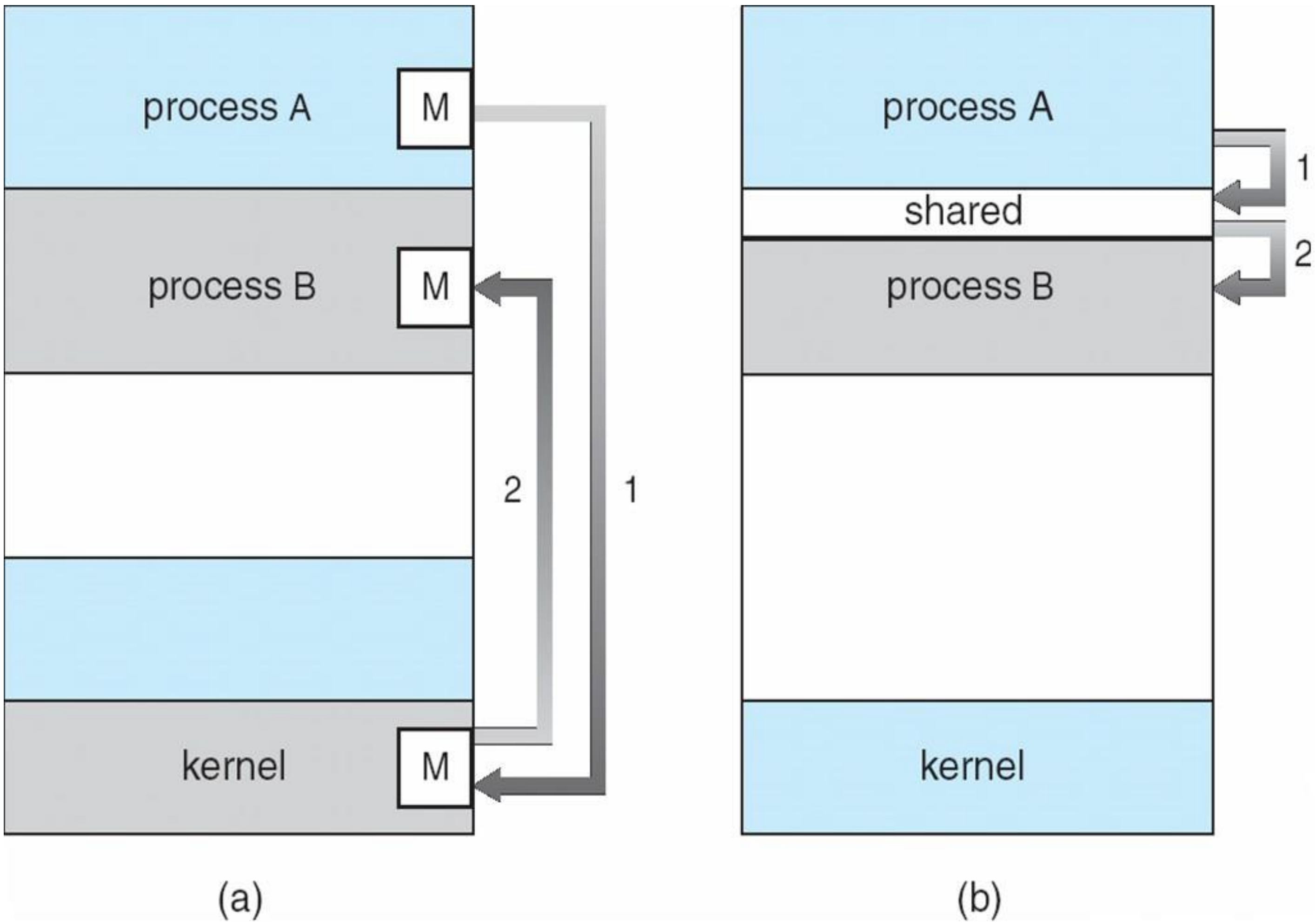
Shared memory:

- a region of memory that is shared by cooperating processes is established
- processes can exchange information by reading and writing data to the shared region

Message passing:

- communication takes place by means of messages exchanged between the cooperating processes.

Communications Models(CO3)



Shared memory vs. Message Passing(CO3)

- **Message Passing is** useful for exchanging smaller amounts of data easier to implement for intercomputer communication
- **Shared memory is** faster as message passing systems are typically implemented using system calls and thus require the kernel intervention in shared-memory systems, systems calls are required only to establish shared-memory regions and all accesses are treated as classical memory accesses and no assistance from the kernel is required.

Shared Memory(CO3)

- Shared memory allows multiple processes to share virtual memory space
- This is the fastest but not necessarily the easiest way for processes to communicate with one another
- In general, one process creates or allocates the shared memory segment
- The size and access permissions for the segment are set when it is created
- The process then attaches the shared segment, causing it to be mapped into its current data space
- If needed, the creating process then initializes the shared memory

Shared Memory (cont.) (CO3)

- Once created, and if permissions permit, other processes can gain access to the shared memory segment and map it into their data space
- Each process accesses the shared memory relative to its attachment address
- While the data that these processes are referencing is in common, each process uses different attachment address values
- For each process involved, the mapped memory appears to be no different from any other of its memory addresses.

Message Passing(CO3)

Message Passing provides a mechanism for processes to communicate and to synchronize their actions without sharing the same address space.

IPC facility provides two operations:

send (message)

receive (message)

If P and Q wish to communicate, they need to:

- establish a communication link between them
- exchange messages via send/receive

Direct Communication(CO3)

Each process that wants to communicate must explicitly name the recipient or sender of the communication:

send (*P, message*) – send a message to process P

receive(*Q, message*) – receive a message from Process Q

Properties of communication link

•Links are established automatically

•A link is associated with exactly one pair of communicating processes

•The link may be unidirectional, but it is usually bi-directional

Indirect Communication(CO3)

Messages are directed and received from **mailboxes** (also known as **ports**)

Primitives are defined as:

send(*A, message*) – send a message to mailbox A

receive(*A, message*) – receive a message from mailbox A

A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed

- each mailbox has a unique *id*
- processes can communicate only if they share a mailbox
- a communication link may be associated with many processes
- each pair of processes may share several communication links
- link may be unidirectional or bi-directional

Daily Quiz

1. Semaphore is a/an _______ to solve the critical section problem.

A. hardware for a system
B. special program for a system
C. integer variable
D. none of the mentioned.

2.Bounded waiting implies that there exists a bound on the number of times a process is allowed to enter its critical section ____________

A. after a process has made a request to enter its critical section and before the request is granted
B. when another process is in its critical section
C. before a process has made a request to enter its critical section
D. none of the mentioned

Daily Quiz

3. When several processes access the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called?

A. dynamic condition
B. race condition
C. essential condition
D. critical condition

4. Mutual exclusion can be provided by the ___________

A. mutex locks
B. binary semaphores
C. both mutex locks and binary semaphores
D. none of the mentioned

Daily Quiz

5. Which process can be affected by other processes executing in the system?

A. Cooperating process

B. child process

C. parent process

D. init process

6. If a process is executing in its critical section, then no other processes can be executing in their critical section. This condition is called?

Mutual exclusion

critical exclusion

synchronous exclusion

asynchronous exclusion

Daily Quiz

7. Concurrent access to shared data may result in ____________

A. data consistency
B. data insecurity
C. data inconsistency
D. none of the mentioned

8. Which of the following conditions must be satisfied to solve the critical section problem?

Mutual Exclusion

Progress

Bounded Waiting

All of the mentioned

Weekly Assignment

- Name some classic problem of synchronization
- Define entry section and exit section.
- What are the requirements that a solution to the critical section problem must satisfy?
- Explain two primitive semaphore operations

Faculty Video Links, Youtube & NPTEL Video Links and Online Courses Details

Youtube/other Video Links

- https://www.youtube.com/watch?v=_zOTMOubT1M
- https://www.youtube.com/playlist?list=PLmXKhU9FNesSFvj6gASuWmQd23Ul5omtD
- https://www.youtube.com/watch?v=x_UpLHXF9dU
- https://www.youtube.com/watch?v=cviEfwtdcEE
- https://nptel.ac.in/courses/106108101

Old Question Papers

Last 12 Years University paper Link:-

https://drive.google.com/drive/folders/1UPXvZ7NY09OunSLrEkTFWpw_M3xLzjhs?usp=sharing

Expected Questions for University Exam

Expected Unit Wise Question link:

https://drive.google.com/drive/folders/1-4f69kIkRvcOYgr_P_VzFgXtRzsS4ilV?usp=sharing

MCQs

1. Semaphore is a/an _______ to solve the critical section problem.

A. hardware for a system
B. special program for a system
C. integer variable
D. none of the mentioned

2. What are the two atomic operations permissible on semaphores?
A. Wait
B. Stop
C. Hold
D. none of the mentioned

MCQs

3. Peterson's solution can be applied for how many process?

A. 4

B. 3

C. 2

D. Any number of processes

4. Semaphore is a __________ based solution

A. Software

B. Hardware

C. Both software and hardware

D. None

MCQs

5. The switching of the CPU from one process or thread to another is called ____________

A. process switch
B. task switch
C. context switch
D. **all of the mentioned**

6. If no cycle exists in the resource allocation graph ____________

A. then the system will not be in a safe state
B. **then the system will be in a safe state**
C. all of the mentioned
D. none of the mentioned

MCQs

7. The dining – philosophers problem will occur in case of _____________

5 philosophers and 5 chopsticks
4 philosophers and 5 chopsticks
3 philosophers and 5 chopsticks
6 philosophers and 5 chopstick

8. In the bounded buffer problem _____________

A. there is only one buffer
B. there are n buffers (n being greater than one but finite)
C. there are infinite buffers
D. the buffer size is bounded

MCQs

9. To ensure difficulties do not arise in the readers – writers problem _______ are given exclusive access to the shared object.

A. Readers
B. Writers
C. readers and writers
D. none of the mentioned

10. What are the two kinds of semaphores?

A. mutex & counting
B. binary & counting
C. counting & decimal
D. decimal & binary

Glossary Questions

Choose the correct option:

1. An edge from process Pi to Pj in a wait for graph indicates that ____________.
2. The wait-for graph is a deadlock detection algorithm that is applicable when ____________.
3. If the wait for graph contains a cycle ____________.

a)then a deadlock exists

b)all resources have a single instance

c)Pi is waiting for Pj to release a resource that Pi needs

Glossary Questions

Choose the correct option:

1. __________is the deadlock avoidance algorithm.
2. The circular wait condition can be prevented by ____________.
3. The bounded buffer problem is also known as ____________

a)Producer – Consumer problem

b)defining a linear ordering of resource types

c)Banker's Algorithm

Old Question Papers

B. TECH.

THEORY EXAMINATION (SEM–IV) 2016-17

OPERATING SYSTEM

Time : 3 Hours — ***Max. Marks : 100***

Note : *Be precise in your answer. In case of numerical problem assume data wherever not provided.*

SECTION – A

1. **Attempt all of the following questions:** **10 x 2 = 20**

- **(a)** Difference between Process and Program.
- **(b)** Explain Context Switching.
- **(c)** What is Demand paging?
- **(d)** Explain Concept of Virtual Memory.
- **(e)** Difference between Directory and File.
- **(f)** Define multiprogramming system.
- **(g)** Difference between External and Internal Fragmentation.
- **(h)** What is Critical Section?
- **(i)** Explain threads.
- **(j)** Define operating system explain in short.

Old Question Papers

SECTION – B

2. Attempt any five of the following questions: 5 x 10 = 50

(a) Write down the different types of operating system

(b) What is Kernel? Describe various operations performed by Kernel.

(c) What is the cause of Thrashing? What steps are taken by the system to eliminate this problem?

(d) What do you understand by Process? Explain various states of process with suitable diagram. Explain process control block.

(e) Give the principles, mutual exclusion in critical section problem. Also discus how well these principles are followed in Dekker's solution.

(f) State the Producer-consumer problem. Given a solution to the solution using semaphores.

(g) Explain File organization and Access mechanism.

(h) Explain the services provided by operating system.

Old Question Papers

SECTION – C

Attempt any two of the following questions: **2 x 15 = 30**

3 (i) What is a deadlock? Discuss the necessary conditions for deadlock with examples

(ii) Describe Banker's algorithm for safe allocation.

4 What do you mean by cashing, spooling and error handling, explain in detail. Explain FCFS, SCAN & CSCAN scheduling with eg.

Expected Questions for University Exam

1. Define race condition.
2. Write and explain Peterson solution to the critical section problem.
3. Discuss product-consumer problem with semaphore.
4. Define critical section.
5. State the Readers/Writers problem with readers having higher priority. Give solution of the problem using semaphores.
6. Explain dinning philosopher problem & its solution.
7. Write a short note on inter-process communication with its advantages.
8. Define critical section.
9. Explain two primitive semaphore operations
10. Name some classic problem of synchronization

Expected Questions for University Exam

12. Write a note on Deadlock Prevention.

13. Write down the steps of Deadlock Detection

14. List out the necessary conditions for deadlock to occur.

Recap of Unit

In this module, we have studied the following:

- Deadlocks
- Deadlock Detection and Recovery
- Cooperating process
- Producer-Consumer Problem
- Race condition
- Critical Section
- Peterson’s Solution
- Classical Problems of Synchronization
  - Bounded-Buffer Problem
  - Readers and Writers Problem
  - Dining-Philosophers Problem
- Problems with Semaphores
- Inter-process Communication
  - Shared memory
  - Message passing

References

Books :

1. Silberschatz, Galvin and Gagne, “Operating Systems Concepts”, Wiley
2. SibsankarHalder and Alex A Aravind, “Operating Systems”, Pearson Education
3. Harvey M Dietel, “ An Introduction to Operating System”, Pearson Education
4. D M Dhamdhere, “Operating Systems : A Concept basedApproach”, McGraw Hill.
5. Charles Crowley, “Operating Systems: A Design-Oriented Approach”, Tata McGraw Hill Education”.
6. Stuart E. Madnick & John J. Donovan, “ Operating Systems”, Tata McGraw

Thank You