Chapter 6 (process Syncharization)

1-Modern operating systems, such as Unix, execute processes ………….. (**concurrently**)

2-Although there is a **single Central Processor** (**CPU**), which execute the instructions of only one program at a time, the **operating system** rapidly switches the processor between different processes.

Some of these resources (such as **memory**) are simultaneously shared by all processes .

4- Such resources are being used **in parallel** between all running processes on the system .

5- Other resources must be used by **one** process at a time .

6- a **deadlock** is a situation wherein two or more competing actions are waiting for the other to finish, and thus neither ever does.

7- **Process synchronization** refers to the idea that multiple processes are to join up or handshake at a certain point to reach an agreement or commit to a certain sequence of action .

Or

"**Synchronization**" involves the orderly sharing of system resources by processes.

8- processes sharing resources on a computer must be properly managed in order to avoid "**collisions**".

9- If only one process is active, then no resource conflict exists , But when both processes are active, and they both arrive at the intersection simultaneously and Collision

---

10- the problem of synchronization is properly stated, Consider the following related definitions:

a- Critical Resource: A resource shared with constraints on its use (e.g., memory, files, printers, etc.)

b- Critical Section: Code that accesses a critical resource .

c- Mutual Exclusion: At most one process may be executing a Critical Section with respect to a particular critical resource simultaneously.

11- sequence of instructions that can be executed by only one process at a time. Such a sequence of instructions is called a **Critical Section.**

Or

That part of the program where the shared memory is accessed is called the **Critical Section.**

12- The key to preventing trouble involving shared storage to prohibit more than one process from reading and writing the shared data simultaneously.

13- To avoid race conditions and flawed results, one must identify **codes** in Critical Sections in each thread .

14- The characteristic properties of the code that form a Critical Section are:

a. Codes that reference one or more variables in a "read-update-write" fashion while any of those variables is possibly being altered by another thread.

b. Codes that alter one or more variables that are possibly being referenced in "read-update-write" fashion by another thread.

c. Codes use a data structure while any part of it is possibly being altered by another thread.
d. Codes alter any part of a data structure while it is possibly in use by another thread

15- each process executing the shared data (variables) excludes all others from doing so simultaneously (**Mutual Exclusion)**

16- in Mutual Exclusion ,we need **four** conditions to hold to have a good solution for the critical section problem (mutual exclusion):

1- No two processes may at the same moment inside their critical sections.

2- No assumptions are made about relative speeds of processes or number of CPUs.

3- No process outside its critical section should block other processes.

4- No process should wait arbitrary long to enter its critical section .

17-to avoid the ……………, devise a pre-protocol (or entry protocol) and a post-protocol (or exist protocol) to keep two or more threads from being in their critical sections at the same time. (**the Mutual Problem**) .

18- When one process is **updating shared modifiable data** in its critical section, no other process should allowed to enter in its critical section.

19- **Disabling interrupts** is sometimes a useful technique within the kernel of an operating system, but it is not appropriate as a general mutual exclusion mechanism for users' process.

20 – Disabling interrupts (Hardware Solution)

⇨ Each process disables all interrupts just after entering in its critical section and re-enable all interrupts just before leaving critical section.

⇨ With interrupts turned off the **CPU** could not be switched to other process. Hence, no other process will enter its critical and **mutual exclusion** achieved.

21-Lock Variable(Software Solution):

⇨ **Single or lock Variable initialize = 0 .**

a- When a process wants to enter in its critical section, **it first test the lock .**

❑ If lock is 0, the process first sets it to 1 and then enters the critical section.

❑ If the lock is already 1, the process just waits until (lock) variable becomes 0.

Note:

⇨ 0 means that no process in its critical section, and 1 means some process is in its critical section.

22-**Strict Solution:**

⇨ In ……………,the integer variable 'turn' keeps track of whose turn is to enter the critical section . (Strict Solution)

⇨ a-Initially, process A inspect turn, finds it to be 0, and enters in its critical section.
b-Process B also finds it to be 0 and sits in a loop continually testing 'turn' to see when it becomes 1

23-Continuously testing a variable waiting for some value to appear is called ………….
(the Busy-Waiting)

24- Sleep: It is a system call that causes the caller to block, that is, be suspended until some other process wakes it up.

25- Wakeup: It is a system call that wakes up the process.

26- Both 'sleep' and 'wakeup' system calls have **one parameter** that represents a memory address used to match up 'sleeps' and 'wakeups'.

27-System Calls (Sleep & Wakeup)

⇨ when a processes wants to enter in its critical section , it checks to see if entry is allowed. If it is not, the process goes into tight loop and waits (i.e., start busy waiting) until it is allowed to enter. This approach waste CPU-time.

28-Bounded Buffer Producers and Consumers:

1-there is a fixed buffer size finite numbers of slots are available

2-Two processes share a common, fixed-size (bounded) buffer. The producer puts information into the buffer and the consumer takes information out.

3-the producer-consumer problem also known as bounded buffer problem.

4-: To suspend the producers when the buffer is full, to suspend the consumers when the buffer is empty, and to make sure that only one process at a time manipulates a buffer so there are no race conditions or lost updates .

29-

a-Problem the Producer: The producer wants to put a new data in the buffer, but buffer is already full.

Solution: Producer goes to sleep and to be awakened when the consumer has removed data.

b- Problem Consumer : The consumer wants to remove data from the buffer, but buffer is already empty.

Solution: Consumer goes to sleep until the producer puts some data in buffer and wakes consumer up.

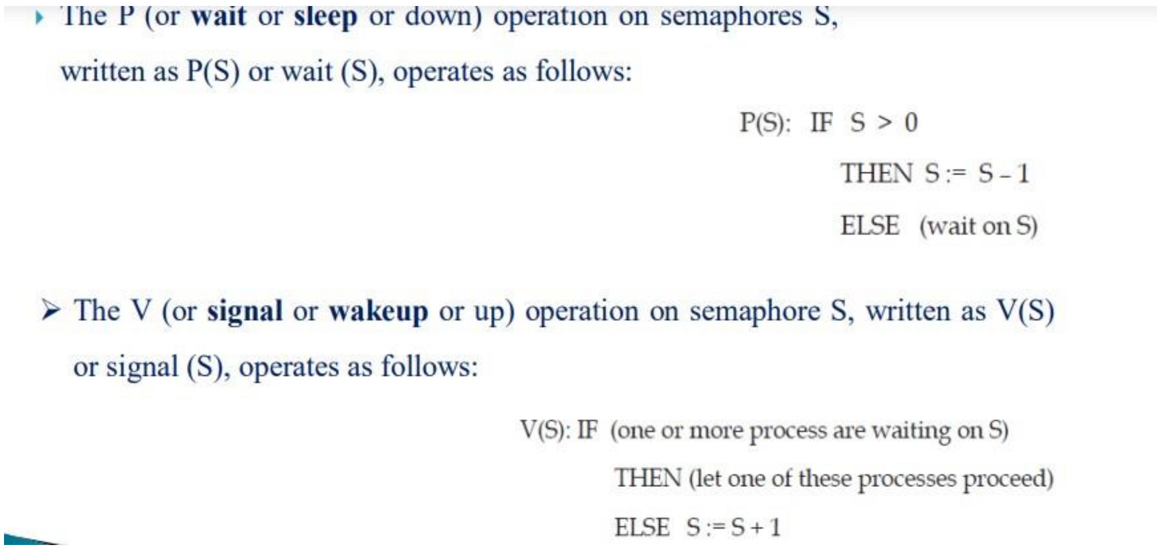
30-**semaphore** is a protected variable whose value can be accessed and altered only by the operations P and V and initialization operation called 'Semaphoiinitislize'.

⇨ There two Types from Semaphore:

1-Binary Semaphores can assume only the value 0 or the value 1

2-Counting Semaphore called general semaphores can assume only nonnegative values .

31-code in Sleep(wait or down) and Wakeup(Signal or Up)in Semaphore:



- The P (or **wait** or **sleep** or down) operation on semaphores S, written as P(S) or wait (S), operates as follows:

```
P(S):  IF  S > 0
           THEN  S := S - 1
           ELSE  (wait on S)
```

- The V (or **signal** or **wakeup** or up) operation on semaphore S, written as V(S) or signal (S), operates as follows:

```
V(S): IF  (one or more process are waiting on S)
          THEN (let one of these processes proceed)
          ELSE  S := S + 1
```

<u>Notes on Semaphore</u>

1-Operations P and V are done as single, indivisible, atomic action .

2-Mutual exclusion on the semaphore, S, is enforced within P(S) and V(S).

3-It is guaranteed that once a semaphore operations has stared, no other process can access the semaphore until operation has completed.

4-The other processes will be kept waiting, but the implementation of P and V guarantees that processes will not suffer indefinite postponement

5-If several processes attempt a P(S) simultaneously, only process will be allowed to proceed.

31- …………… solve the lost-wakeup problem. (Semaphore)

32-A **semaphore** s is an integer variable that apart from initialization, is accessed only through two standard atomic operations, wait and signal. these operations were originally termed p (for wait to test) and v (for signal to increment).

Or

- ⇨ Its purpose is to lock the resource being used

33-A **Semaphore** is hardware or a software tag variable whose value indicates the status of a common resource.

32-In multitasking operating systems, the activities are synchronized by using the **semaphore techniques.**

34-Modification to the integer value of semaphore in wait and signal operations must be executed **individually**

35-A **monitor** is a software synchronization tool with high-level of abstraction that

provides a convenient and effective mechanism for process synchronization.

36- **Deadlock** occurs when you have a set of processes [not necessarily all the processes

in the system], each holding some resources, each requesting some resources, and

none of them is able to obtain what it needs .

37- processes are deadlocked because all the processes are **waiting**.

38-Each member of the set of deadlocked processes is waiting for a resource that is owned by another deadlocked process.

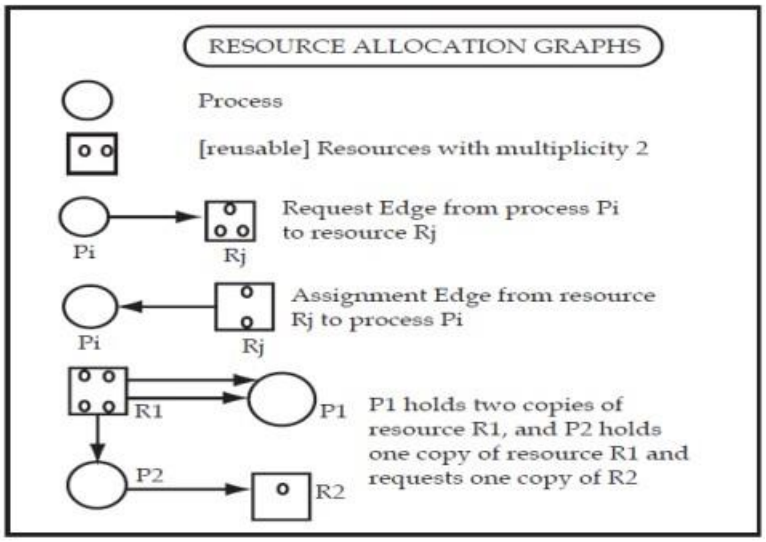
39- **Deadlock Conditions:**

- ⇨ Deadlock situation can arise if the following **four** conditions hold simultaneously in a system:
  1) Resources are used in mutual exclusion.
  2) Resources are acquired piecemeal (i.e. not all the resources that are needed to complete an activity are obtained at the same time in a single indivisible action).

3) Resources are not preempted (i.e. a process does not take away resources being held by another process).

4) Resources are not spontaneously given up by a process until it has satisfied all its outstanding requests for resources (i.e. a process, being that it cannot obtain some needed resource it does not kindly give up the resources that it is currently holding).

40-**Resource Allocation Graphs (RAGs)** :are directed labeled graphs used to represent, from the point of view of deadlocks, the current state of a system.


RESOURCE ALLOCATION GRAPHS
Process
[reusable] Resources with multiplicity 2
Pi
Rj
Request Edge from process Pi to resource Rj
Pi
Rj
Assignment Edge from resource Rj to process Pi
R1
P1
P2
R2
P1 holds two copies of resource R1, and P2 holds one copy of resource R1 and requests one copy of R2

41-**State transitions** can be represented as transitions between the corresponding resource allocation graphs .

42-There are three rules for state transitions:

1) Request: If process Pi has no outstanding request, it can request simultaneously any number (up to multiplicity) of resources R1 , R2 , ..Rm. The request is represented by adding appropriate requests edges to the RAG of the current state.

2) Acquisition: If process Pi has outstanding requests and they can all be simultaneously satisfied, then the request edges of these requests are replaced by assignment edges in the RAG of the current state

3) Release: If process Pi has no outstanding request, then it can release any of the resources it is holding and remove the corresponding assignment edges from the RAG of the current state.

43-important propositions about deadlocks and resource allocation graphs(RAG):

1) If a RAG of a state of a system is fully reducible (i.e. it can be reduced to a graph without any edges using ACQUISITION and RELEASE operations) then that state is not a deadlock state.

2) If a state is not a deadlock state, then its RAG is fully reducible (this holds only if you are dealing with reusable resources; it is false if you have consumable resources)
3) 3) A cycle in the RAG of a state is a necessary condition for that being a deadlock state .
4) 4) A cycle in the RAG of a state is a sufficient condition for that being a deadlock state only in the case of reusable resources with multiplicity one.

44- <u>Handling of Deadlocks:</u>

- There are Four ways to address the problem of deadlock in an operating system:

⇨ Prevent
⇨ Avoid
⇨ Detection and Recovery
⇨ Ignore

45-<u>Deadlock Prevention</u>:

1-Mutual exclusion:

- Removing the mutual exclusion condition means that no process may have exclusive access to a resource.
- Algorithms that avoid mutual exclusion are called non-blocking synchronization algorithms.

2- **Hold and wait:**

- The “ hold and wait ” conditions may be removed by requiring processes to Request all the resources they will need before starting up (or before embarking upon a particular set of operations)
- Another way is to require processes to release all their resources before requesting all the resources they will need. This too is often impractical. (Such algorithms, such as serializing tokens, are known as the all-or-none algorithms.)

No preemption:

- A “**no preemption**” (lockout) condition may also be difficult or impossible to avoid as a process has to be able to have a resource for a certain amount of time, or the processing outcome may be inconsistent, or thrashing may occur. However, inability to enforce preemption may interfere with a priority algorithm.
- Algorithms that allow preemption include **lock-free** and **wait-free** algorithms and **optimistic concurrency control**.

4-**Circular wait**:

- The circular wait condition: Algorithms that avoid circular waits include "disable interrupts during critical sections" , and "use a hierarchy to determine a partial ordering of resources" (where no obvious hierarchy exists, even the memory address of resources has been used to determine ordering) and Dijkstra's solution.

46-a **safe state** : a state which there is a sequence of allocations and releases of resources that allows all processes to terminate)

47-The Banker's Algorithm is used to determine if a request can be satisfied.It uses requires knowledge of who are the competing transactions and what are their resource needs.

48-**Deadlock Avoidance**:-

⇨ assuming that you are in a safe state (i.e. a state from which there is a sequence of allocations and releases of resources that allows all processes to terminate) and you are requested certain resources, simulates the allocation of those resources and determines if the resultant state is safe. If it is safe the request is satisfied, otherwise it is delayed until it becomes safe.

49-**Deadlock Detection & Recovery** :-

1- Often neither deadlock avoidance nor deadlock prevention may be used.

2- employing an algorithm that tracks resource allocation and process states and rolls back and restarts one or more of the processes in order to remove the deadlock .

3- Detecting a deadlock that has already occurred is easily possible since the resources that each process has locked and/or currently requested are known to the resource scheduler or OS .

50 In Ignore Deadlock , the Ostrich Algorithm it is hoped that deadlock doesn't happen.

⇨ Deadlock is unlikely to occur very often; a system can run for years without deadlock occurring.(Ignore Deadlock)

⇨ If the operating system has a deadlock prevention or detection system in place, this will have a negative impact on performance (slow the system down) .