Date: 7-1-24

Multithreading Models & Thread States in OS.

Multithreaded Models:

→ User threads: are ~~managed~~ supported above the kernel. Ty are managed without kernel support.

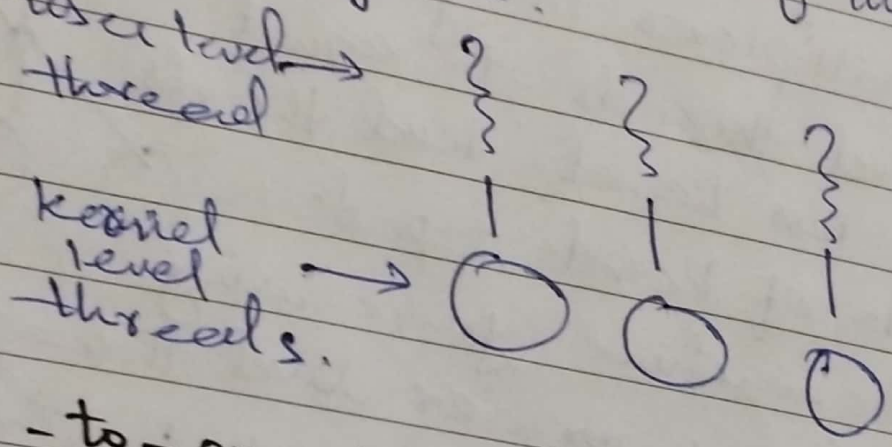
→ Kernel threads: Supported ty managed directly by the OS.
A relationship must exist between user ty kernel level threads.

1. One-to-One Model:

- maps each user thread to kernel thread
- when a thread makes a blocking system call, another thread can be run
- Multiple threads can be run parallely on multiprocessors. → Advantage
- Drawback: Every user thread requires to make a kernel thread. ty it can burden the performance of an application

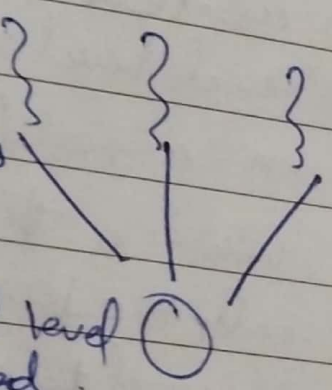
Date:

- there is restriction on number of threads supported by system.



2. Many-to-one Model:

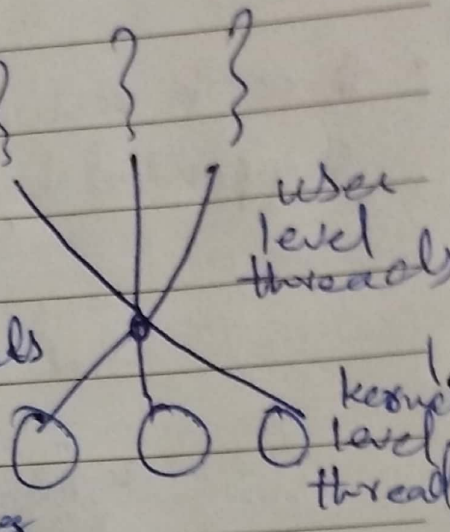
- maps many user level threads to one kernel-level thread.
- Thread management is done by user space using Thread Library, hence it is efficient → Advantage.
- Drawback: If a thread makes a blocking system call, entire process will be blocked.



One thread can access kernel at a time so no parallel execution on multiprocessors

Date:

3. Many-to-Many Model:



- multiplexes many user-level threads to equal or less kernel-level threads
- No. of kernels depends on the machine you're using
- As many user level threads can be created as necessary. & the corresponding kernel level threads can run in parallel on a multiprocessor → Advantages.
- When a thread makes blocking system call, the kernel can schedule another thread to execute.

Date:

Relationship B/w Threads & Processes

Threads : Processes

① 1 : 1

Each thread of execution is a unique process with its own address space & resources. Allows parallel execution.

Ex: UNIX. (Traditional OS).

② M : M.

One process contain multiple threads. A process has an address space & resources, which every thread shares.

Ex: Modern OS, Linux, Windows.

③ 1 : M

A thread can migrate from one process environment to another through inter-process communication.

In distributed OS, such as in Clouds OS.

Ex RA (Clouds), Emerald

Date:

(iv) M : N
- Both 1:M & M:1 approaches.
  Ex: Experimental OS (TRIX).
- Multiple threads are mapped to multiple processes
- Threads & processes are managed independently.

Symmetric Multiprocessing, SMP
A form of multiprocessing that allows the OS to execute on any available processor or on several available processors simultaneously.
- Kernel can execute on any processor.
- Each process does self-scheduling from the pool of available processes or threads; allowing parallel execution.

A multiprocessor OS must provide all multiprogramming system functions + other features to accomodate multiprocessors

Date:

Multiprocessing OS Design Consideration

① Simultaneous concurrent processes / threads on OS

② Scheduling: Since any processor can do scheduling, so there must be some scheduling policy to avoid. and corruption

③ Synchronization: Synchronization is a facility to enforce mutual exclusion & event ordering for multiple active processes.

④ Memory Management:
Paging mechanism, on different processors
Segmentation, when several processors share the same page or a segment to enforce data consistency.

⑤ Reliability & fault tolerance,
effective process failure handling.

NBP

Date:

Microkernels:

- A small privileged OS core.
- Provides process scheduling, memory management & interprocess communication services
- Relies on processes to perform some other functions ~~by OS kernel~~; user-level processes called servers.

Benefits of Microkernel Organization.

- Extensibility: New services can be added or existing ones can be modified without modifying core kernel.
- Flexibility: Components can be added or replaced without affecting core kernel.
- Portability: change needed to adapt on new processor is done in microkernel.

14TH AUGUST

HAPPY INDEPENDENCE DAY

Date:

- Reliability: Modular design can be tested rigorously.
- Distributed System Support: Messages are sent without knowing the targeted machine, maintain isolation.
- Object Oriented OS: Components are objects, interconnected by an interface.

Microkernel Design:

- Low-level memory management: mapping each virtual page to a physical page frame.
- Interprocess communication: Facilitates communication b/w processes.
- I/O & interrupt management.

Date: 7-Jan-24

CONCURRENCY: Mutual Exclusion & Synchronization

The central purpose of OS design is to manage processes & threads.

Multiprogramming: Management of multiple processes within a uniprocessor system.

Multiprocessing: Management of multiple processes within a multiprocessor

Distributed Processing: Management of multiple processes executing on multiple, distributed computer system.

Multiple processes or threads, being executed on distinct systems without knowing the essential details of processing architecture or targeted machines.

Concurrency is the fundamental of all these techniques & OS design

NBP

Date:

Concurrency: Ability of an OS to execute multiple set of tasks or processes simultaneously.
Concurrency arises in three different content.

1. Multiple Applications: Multiprogramming allow processing time to be dynamically shared between no. of active applications.
2. Structured Applications: an extension modular design & structured programming. Some applications can be effectively programmed as a set of concurrent processes.
3. Operating System Structure: The same structuring advantages apply to system programs. OS itself is implemented as a set of process / threads.

Date:

The basic requirement for support of concurrent processes is the ability to enforce mutual exclusion.
Mutual exclusion is the ability to exclude all other processes from a course of action while one process is granted the ability.

Concurrency & Shared data:

- Concurrent processes may share data to support IPC.
- Threads in a process share same address space (global).
- Concurrently sharing data may cause problems
  e.g. lost updates.

Date:

Key terms related to Concurrency:

① Atomic Operations: An indivisible function or action with a sequence of instruction & is not interruptable. The instructions will execute as a group or will not execute. It wont effect System state. Atomicity ensures isolation from concurrent processes.

② Critical Section: A section of code in the process, that requires any shared resources. & that must not be executed while another process is in a corresponding section of code.

Simply, a resource is in one processes use & it is in critical section. Second process also needs the resource but can't access it until Process one completes

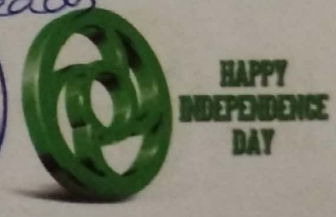
Date:

(3) Deadlock: deadlock arises when two process are waiting for each other to do something, either releasing a resource or genrating an output.

(4) Livelock: livelock arises when processes changes their states for no useful reason. In response to other processes, without any useful work.

(5) Mutual Exclusion: It is a requirement, It ensures that when a process is in a critical section using shared resources, no other process can be in a corresponding critical section to access the same resources.

(6) Race Condition: A situation in which multiple threads / processes read & write a shared data item. The final result depends upon their relative timing of execution. Lastly, which threads executed? (Order of the execution)



Date:

(2) Starvation: A situation in which a process is not being choosen by the dispatcher although it is runnable.

Principle of Concurrency::

In a multiple-processor system, it is possible to inteleave the execution of multiple processes and to overlap them.

Both techniques can be viewed as examples of concurrent processing & both present the same problem.

In multiprogramming system, the relative speed of execution of processes can't be predicted.

It depends on other processes activities, OS interrupt handling techniques, & scheduling policies of OS.

Date: Sept 8, 2018

* Speak with honesty, think with sincerity and act with integrity.

Difficulties of Concurrency:

① Sharing of global resources. e.g. read & write operations on a global variable.

② Manage & allocate resources optimally e.g. a process has a resource & suspends in between. the resource is locked with the process leading to deadlock.

③ Difficulty to locate programmatical errors. because are non-deterministic & reproducible.

MY SCHOOL

Date 8-Jan-24.

OS Concerns

Design & Management issues raised by the existence of concurrency.

- The OS must be able to keep track of processes (using process control block)
- OS must allocate & deallocate resources to processes
- OS must protect data & physical resources concerned with each process against other processes.
- Functioning of output of process, must be independent of speed of execution relative to other concurrent processes.

Process Interaction:

They are classified as the degree of awareness upto which they are aware of each other existence

- Processes unaware of each other:
  - independent process, not attended to work together
  - processes executing independently are in competition of resources. The OS must regulate the access of resources.
  - Timing of process may be affected,

Date

- Potential Control Problems: Mutual exclusion, deadlock, starvation.
- Processes indirectly aware of each other.
- these processes are not necessarily aware of each other by process Ids, but that share access to some object, e.g. I/O buffer.
- They are related as cooperative.
- Such processes exhibit cooperation in sharing the common object.
- Results of one process may depend on others as well as time of process.
- Potential Control Problem: Mutual exclusion, deadlock, starvation, data coherence (a relationship b/w data maintain)
- Processes directly aware of each other.
- These process are aware of each other by they can communicate by process ID.
- They exhibit cooperation by communication.
- Results of one process may depend on others as well as time of process
- Potential Control Problem: Deadlock, Starvation

MY SCHOOL

Date

Requirements for mutual exclusion:

1. Mutual exclusion must be enforced: only one process at a time is allowed into its critical section, among all processes having similar critical section for the same resource.
2. A process that halts in it is not in its critical section must not interfere other processes
3. No deadlock or starvation: Process should not be delayed indefinitely, requiring access to critical section
4. If critical section is free, a process requiring access to it must be permitted to enter in it
5. No predictions or assumptions are made about relative process speed or no. of processors
6. A process remain in its critical section for a finite time

The Critical Section Problem:

- When a process executes, that exploits resource or manipulates shared data, the process is in critical section for that
- For execution of critical section, mutual exclusion must be enforced, to ensure only one process in

Date _____

its critical section at a time.

- Mutual exclusion ensures, data coherence.
  Data coherence: there is a shared ~~resource~~ data that is a relationship with other data, they should maintain their coherence after the sequence of processes.
- Deadlock: Process P1 has Resource R_2 on it is in its critical section. Process P_2 has resource R_1 on it is in its critical section. P1 is waiting for R_1 on P_2 is waiting for R_2, both are waiting for each other to leave their critical section, they are deadlocked.
- Starvation: A process is repeatedly denied access to a resource (critical section). Process P1, P2, P_3. P1 ~~is~~ is in critical section, P_2 on P_3 are waiting P1 free, P_2 occupies critical section, P_2 free, before P_3, due to priority P1 access critical section it is repeating, P_3 is in starvation.

Date

Mutual Exclusion: Hardware Support

1. Interrupt Disabling:

- It works in uniprocessor systems.
- Uniprocessor system: process can interleave but can't overlap (execute concurrently).
- One process at a time will be in critical section, disabling interrupt, no other process can demand critical section for a resource
- To guarrantee mutual exclusion, it is sufficient to disable interrupts
- Disabling interrupts, interleaving will be reduced
- It will not works in multiprocessor systems

2. Simple-Machine Instructions

- On multiprocessors system, processors share same main memory
- There isn't any interrupt mechanism b/w processors to achieve mutual exclusion
- At hardware level, access to a memory location, will not allow any other access. the access is blocked for any other instruction

Date _____

MY SCHOOL

Compare & Swap Instruction.

- There are two atomic operations, compare & swap.
- First, at a certain memory location, it will compare with a test value, if it matches, it will swap ~~update~~ the referenced location with new value otherwise not.
  In both cases, it will return old value.
- From the example code, where bolt is that memory location initialized to zero, test value is 0 & new value is 1. Then, memory will be updated & it will have access to critical section, if old value is 1, then it will do nothing & wait for its turn to enter in critical section.
- The wait mode is -> busy waiting or spin waiting.
- A process leaves critical section resets bolt to 0. The other process which didnot get access for CS, was checking continously & will get access now. but only one at a time.

Date

Exchange Instruction:

- This is another instruction to achieve mutual exclusion
- bolt is in memory & key is on local variable or register
- bolt n key value will be exchanged.
- when bolt is zero, process enters in critical section n after completion, it'll set bolt to 0.
- If bolt is 1, then it'll wait to enter in critical section

$$bolt + \sum_i key_i = n.$$

- bolt = 0 ⇒ no process is in critical section
- bolt = 1 → exactly one process is in critical section or the process, whose key value is 0.

Properties of Special Machine Instructions:

Advantages:

- applicable to any number of processes, both in uni n multiprocessor system sharing main memory
- simple n easy to verify.
- multiple critical sections can be handled each with their own bolt variable.

Date

MY SCHOOL

Disadvantages:

- Busy waiting is employed for processor consuming processors time
- Starvation is possible, if more than one process are waiting for CS, any process can be denied access.
- Deadlock is possible: P1 is in CS, P2 with higher priority interrupts P1. P2 asks for resource using by P1, it will not be granted due to mutual exclusion. ??

SEMAPHORES:

- Semaphore is an integer variable initialized with a non-negative value.
- It is used for signaling a process. for cooperation b/w them
- Two operations
  1. decrement semaphore, semWait operation, if value becomes -ve, then the process will be blocked.
  2. increment a semaphore, semSignal operation,

Date

after incrementing, if value ≤ 0, then the blocked process due to semWait will be unblocked.

- To began, it can initialize with ≥ 0 value.
- value → +ve integer, shows that no. of processes that can issue a wait & immediately continue to execute.
- If the semaphore value = 0, (by initializing to 0 or by. no. of processes = initial semaphore values issued wait), then the next process to issue wait will be blocked. & semphore => -ve.
- -ve value of semphore = no. of processes waiting to be unblocked.
- When semaphore is -ve, each signal (semSignal) will unblock one process.
- We can't say it before, that the decrement will block a process.
- After an increment, a process is woken up & they both run concurrently.

Date _____

- Signaling semaphore doesn't ensure a process is waiting, unblocked processes can be one / zero.

Binary Semaphore:

- initialized by 0 or 1.
- SemWait operation checks if $s == 0$, then the process will be blocked. If $s == 1$, the value is changed to 0 & process executes
- SemSignal operation checks if there are blocked processes ($s \leq 0$). If yes, then process is unblocked & $s = 1$.

Non-binary semaphore is also called counting semaphore or general semaphore

Mutex:

- Like binary semaphore
- difference is, a process that locked the mutex (set value to 0) must be the one to unlock it (set value to 1).

Date

- A queue is used to maintain blocked processes for both counting & binary semaphores.
- It uses FIFO, process blocked first will be unblock first.
- Strong Semaphore: is the one that uses FIFO policy.
- Weak Semaphore, is the one that does not specify any order, by which processes are unblocked.
- Strong semaphores guarantee freedom from starvation & it is convenient while weak doesn't & isn't.

Producer / Consumer Problem:

```
semWait (Semaphore s) {
    s = s - 1;
    if (s < 0)
        add process in blocked list
    else
        return;
}
```

```
semSignal (Semaphore s) {
    s = s + 1;
    if (s <= 0)
        wake up process in ready queue
    else
        return;
}
```

Date

MY SCHOOL

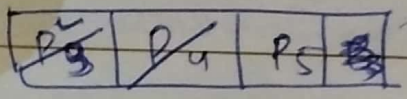
If S is a +ve integer > 0, then we can say S is the no. of processes that can enter into CS.

If S is 0, then no next process can execute in Critical section & will be blocked.

if S is -ve integer < 0, then it shows the total number of blocked processes.

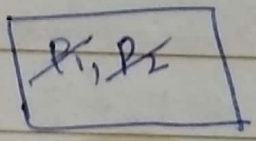
Let S = 2 1 0 -1 -2 -3

Sem Wait -2 -3 -2 -1



Blocked

Critical section



P3

P3, P4

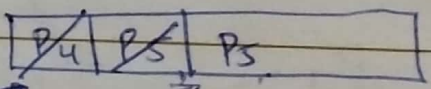
Sem Signal.

3 - 3 + 2 → 3 : W → 2 S

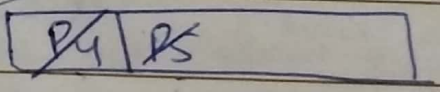
3u - 2d → 1

Let S = 3 + 2 1 0 -1 -2

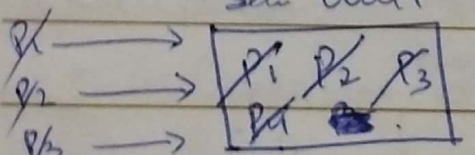
Sem wait -1 0 +1 -1 -2



B



R.



Sem Signal.

Date

For Binary Semaphore.

```
SemWait (Semaphore S){
    if (S == 1)
        S = 0;
    else {
        Block this process
    }
}
```

```
SemSignal (Semaphore S) {
    if blocked list is empty
        S = 1;
    else {
        wake up a process from blocked list
    }
}
```

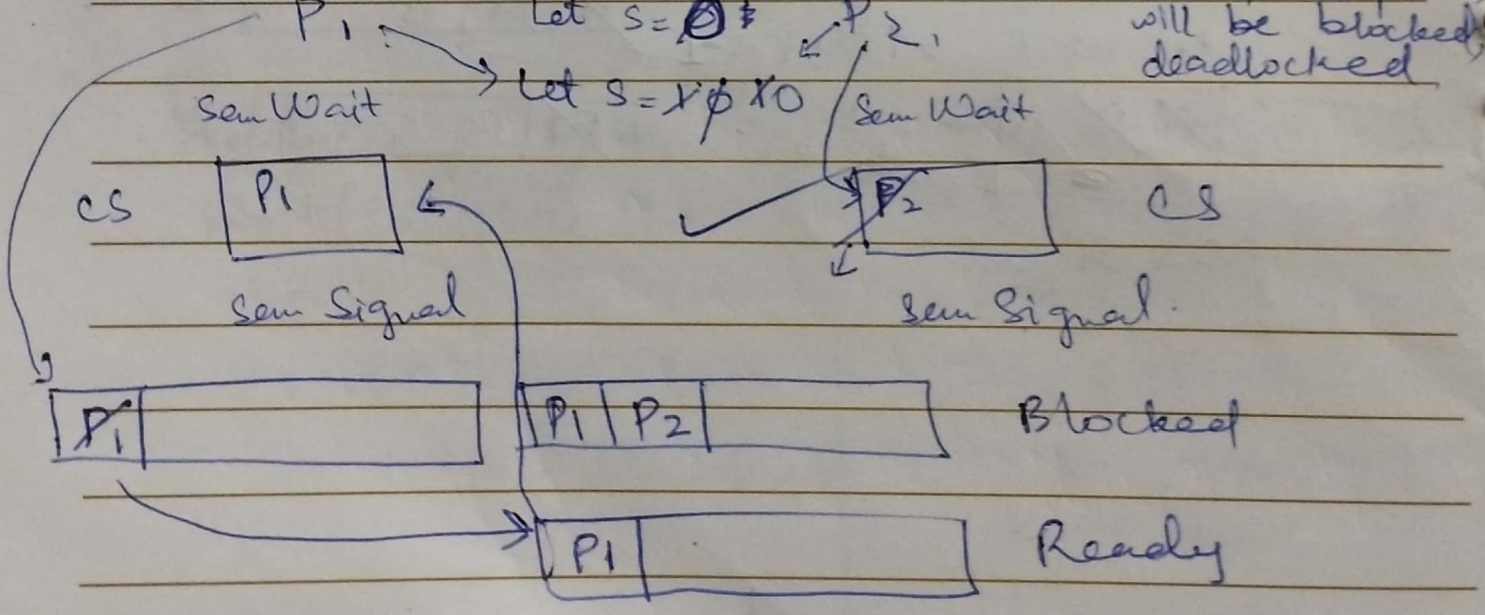
S = 1 → success enter Critical section

S = 0 → unsuccessful Block the process

S = 1 (if block list is empty)

S = 1 (no change)

if S is initialized to 0, all process will be blocked, deadlocked

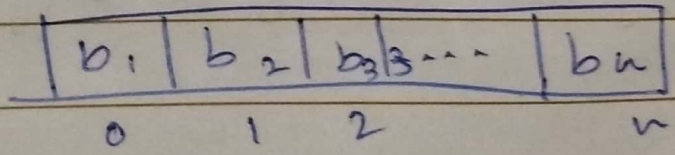


Head Office: 84, Aibak Block, New Garden Town, Lahore. +92 42 35941984-6
E-mail: info@myschools.pk www.myschools.pk

Date 15-1-24.

The Producer/Consumer Problem.

- A common problem faced in concurrent processes
- Statement: "There are one or more producers generating some type of data (records, characters) and placing these in a buffer. There is a single consumer that is taking items out of the buffer one at a time. The system is constrained to ~~be~~ prevent the overlap of buffer operations. That is, only one agent (producer or consumer) is allowed to access buffer at any one time".
- Problem: A producer won't try to add data if it is full and consumer won't try to remove item from an empty buffer.
- Producer process to add data into a buffer
- Consumer Process to remove data from a buffer
- Buffer is shared between both processes.
- They are cooperative.
- Let's assume a buffer is an infinite linear array



Date

Producer Function

```
Produce ( ){
  while (true){
    // produce item v
    b[in] = v;
    in++;
}
```

↓

producer generates an item v, store it into a buffer, and increment an index → in

Consumer function

```
consume( ){
  while (true) {
    while (in <= out)
      // do nothing
    w = b[out];
    out++;
    // consume item w.
```

↓

consumer first make sure, it does not read from an empty buffer. ~~in c~~ it will consume only if in > out

Implementing the system using Binary Semaphore.

- Keep track the number of items in the buffer using variable n (n = in - out).
- Semaphore S is used to force mutual exclusion.
- Semaphore delay is used to make consumer wait using semWait, if the buffer is empty.

```
semWaitB(SemaphoreS){
  if (S==1)
      S=0;
  else
      Block
}
```

```
SemSignalB(Semaphore S){
  if(block list is empty)
      S=1
  else
      wake up
}
```

Date:

```
int n;      =0.
binary_semaphore S=1, delay=0;
producer(){
  while(true){
    produce();
    semWaitB(s);
    append();
    n++;
    if(n==1)
      semSignalB(delay);
    semSignal(s)
  }
}
```

```
consumer(){
  semWaitB(delay);
  while(true){
    semWaitB(s);
    take();
    n--;
    semSignalB(s);
    consume();
    if(n==0)
      semWaitB(delay);
  }
}
```

Lets run this

S. n delay. Semaphores

Blocked

1 semWaitB(S).
semSignalB(delay)

S	delay	n	Semaphore Processes.	
1	0	1	wait(S), n++, signal(D), signal(S)	P
1	0	0	wait(D), wait(S), n--, signal(S),	Block wait(D) C
1	0	1	wait(S), n++, C→ signal(D), signal(S)	P
1	0	0	Prev C(last) n==0 wait(D)X, wait(S), n--, signal(S),	Block
1	0	-1		wait(D), wait(S), n--, signal(S)

There is no data, producer is generating, consumer is trying to get from empty

```
SemWaitB(Semaphore S){
    if (S==1)
        S=0
    else
        block
}
```

```
SemSignalB(Semaphore S){
    if block list is empty
        S=1
    else
        wake up.
}
```

value of n has reached -1, means, no element in the buffer, consumer is trying to get It leads to deadlock condition.

A helping variable can be made in consumer in critical section.

Producer

```
int n=0;
binSemaphore s=1, delay=0
produce(){
  while(true){
    produce();
    SemWaitB(s);
    append();
    n++;
    if(n==1)
      SemSignalB(delay);
    SemSignalB(s);
```

Consumer

```
consume(){
  int m;
  SemWaitB(delay);
  while(true){
    SemWait(s);
    take();
    n--;
    m=n;
    SemSignal(s);
    consume();
    if(m==0)
      SemWaitB(delay);
  }
```

Blocked

Date

s	d	n	semaphore processes.	
1	1	1	wait(s), n++, signal(D), signal(s)	P
1	1	0	wait(D), wait(s), signal(s), n--, wait(D) Block	C.
1	1	1	wait(s), n++, signal(D), signal(s)	P
~~1~~ 0	0	0	~~wait(D), wait(s), n--, signal(s),~~ wait(D), wait(s), n--, m=-1, signal(s)	
1	0	0	wait(D), wait(s), n--, m=0, signal(s), wait(D) Block	C
1	0	1	wait(s), n++, signal(delay), signal(s)	
1	0	0	wait(D), wait(s), n--, m=0, signal(s), wait(d),	

now, consumer won't get any data from buffer, until the producer has produced any data.

Solution using General (counting) Semaphore:

- Here, n is a semaphore now, but its value is the number of data items in buffer.
- no delay semaphore here.

MY SCHOOL

Semaphore n=0, s=1; Date

```
producer() {                    consumer() {
  while(true) {                   semWait(n);
    produce();                    semWait(s);
    semWait(s);                   take();        } CS.
    append();     } C.S           semSignal(s);
    semSignal(s);                 consume();
    semSignal(n);                 }.
  }
```

① • Suppose, semSignal(s) & semSignal(n) are interchanged.
• This makes semSignal(s) to be done in producer's critical section, with no interfere by consumer or other producer.
• This would not affect program, because consumer must wait on both n & s semaphores.

② • Suppose, semWait(n) & semWait(s) operations are interchanged.
• This is not acceptable, since semWait(n) ensures) Semaphore = 1, in this case, there is a data in buffer, then

consumer should enter its critical section.

- But if semWait(s) implements before en. allow the critical section to run, then if n = 0, it will be the condition that consumer is trying to get data from an empty buffer.
- The system is deadlocked.
- So, the operations should be implemented in the correct sequence to avoid dead locks & enforce mutual exclusion, also avoiding race condition (?).

Solution using finite buffer, buffer is circular.

Producer:

```
while (true) {
// Produce an item v.
   while ((in + 1) % n == out)
      // do nothing
   b[in] = v
   in = (in + 1) % n
```

Consumer:

```
while(true) {
   while(in == out) {
      // do nothing
   w = b[out]
   out = (out + 1) % n
   // consume item w.
```

```
semWait(semaphore s){
    s--;
    if(s.count<0){
        block
    }
}
```

```
semSignal(semaphore s){
    s.count++;
    if(s.count<=0){
        wake up
    }
}
```

Date ~~let e = to~~

```
const int sizeOfBuffer = __
semaphore n=0, s=1, e = sizeOfBuffer
```

```
producer(){
    while(true){
        Produced();
        semWait(e);
        semWait(s);
        append();
        semSignal(s);
        semSignal(n);
    }
}
```

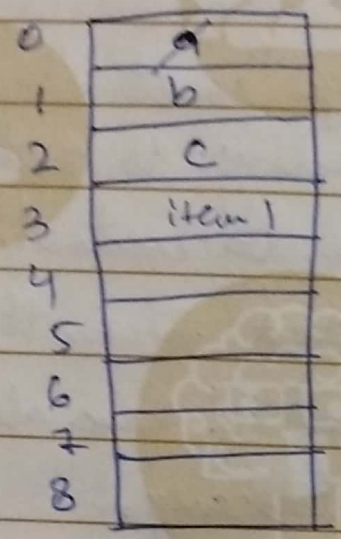
```
consumer(){
    while(true){
        semWait(n);
        semWait(s);
        take();
        semSignal(s);
        semSignal(e);
        consume();
    }
}
```

- This implementation uses counting semaphores
- The process will block, if producer inserts in full buffer or consumer tries to get data from an empty buffer.
- It will unblock on consumer → item inserted
  producer → item removed.

S	e	n		
~~1~~	~~2~~	~~1~~	wait(e), wait(s), signal(s), signal(n)	P
~~1~~	~~3~~	~~0~~	wait(n), wait(s), signal(s), signal(e)	C
1	3	-1	wait(n) block	C
~~1~~	~~2~~	~~0~~	wait(e), wait(s), signal(s), signal(n)	P
~~1~~	~~2~~	0	wait(s), signal(s), signal(e) wake up	C

Date

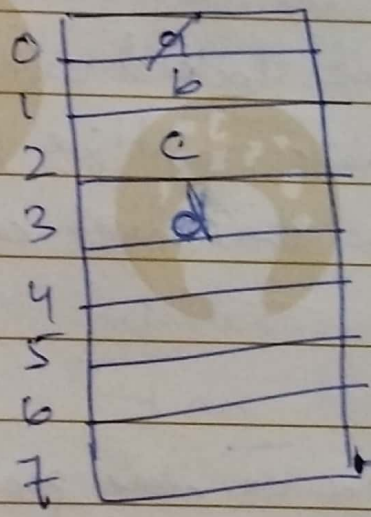
Total = 9. in out
n = Filled = 3. 8 0
4 1



S	n	e	
1	3	6	
0	4	5	Producer.
1	3	6	Consumer.

no preemption example.

Total = 8. in ; out
n = 3 4 1



S	n	e	
1	3	8	
0		4	Producer can't wait (preempted)
1	2	5	Consumer
1	3	5	Producer.

Date

Monitors:

- Monitor is a programming language construct that provides equivalent functionality to that of semaphores & that is easier to control.
- The construct have been implemented in multiple programming languages e.g. Java & have also been implemented as a program library.
- Monitor lock, can be placed on any object.
- E.g. all linked lists can be locked with a lock. or every linked list can have multiple locks for each or every node in the list can have a lock.

Monitor & with Signal:

- A software module with one or more procedures an initialization sequence & local data.

Characteristics:

→ local variables are accessible only by monitors procedure. they're private.

→ process enters monitor by invoking one of its procedures.

Date _____

- Only one process may be executing in monitor at a time. Others trying to invoke monitors are blocked.

→ The first two characteristics resembles object oriented design. Indeed an OO. OS / PL can implement monitor as an object with special characteristics

→ Ensuring one process at a time make monitor ~~to enforces~~ mutual exclusion.

→ For concurrent processing, monitor must ensure synchronization.

Condition Variables:

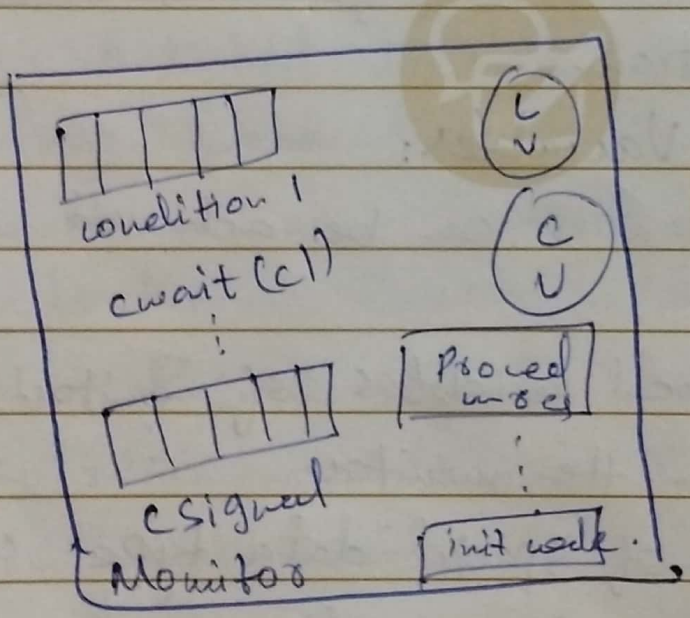
- Synchronization can be achieved using condition variables.
- They are local variables of monitor. Is accessible only within the monitor.
- They are of special data type on are operated on by these two functions.

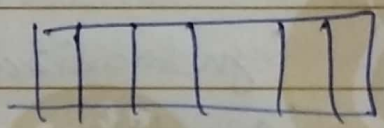
→ cwait(c): Suspend execution of calling process on condition 'c'. The monitor is now available to use by another process.

Date

csignal(c): Resume the execution of some process blocked after a cwait on the same condition. If there are several such processes, choose one of them; if there is no such procaccess, do nothing

- This wait & signal operations are different than semaphore operations.
- The signal is lost, if a process in monitor signals & no task is waiting on the condition.




Process waiting to enter in monitor.

Structure of a monitor.

- A process enters in monitor by invoking one of its procedures
- One process enters in monitor at a time & next

Date _____

MY SCHOOL

will be blocked or placed in a queue

- A process in the monitor may be temporarily blocked on some condition 'c' by issuing 'wait(c)' then it waits for 'c' to change to reenter. (enter in condition queue)
- csignal(c) will alert that condition has changed

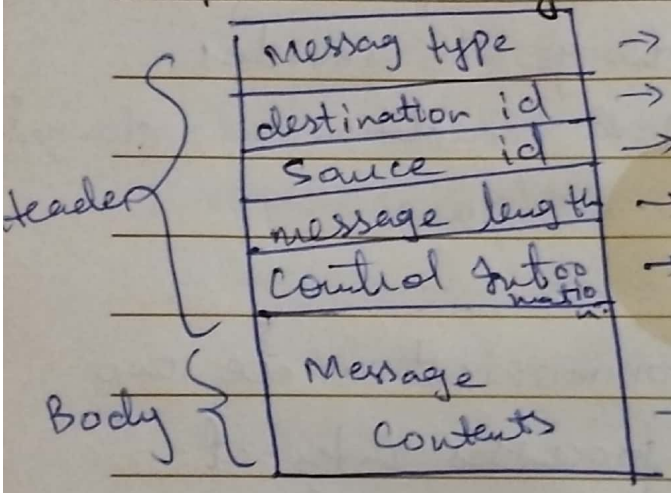
Message Passing:

- Synchronization and Communication are two basic requirements when processes interact.
- Synchronization to enforce mutual exclusion.
- Communication to exchange information as the processes are cooperative.
- Message passing approach can do both.
- This can be implemented in distributed systems, shared-memory multiprocessor & uniprocessor systems.
- The function of message passing includes two primitives
  1. Send (destination, message)
  2. recieve (source, message).
- A process sends information using send primitive, message is information, ~~destination~~ destination of another process.

Date

- A process recieves information using recieve primitive message → information, source of the process.

General Message Format:



- message type → Type to discriminate b/w different messages
- destination id → the targeted processes
- Source id → the current processes
- message length → number of characters
- Control Information → e.g. Pointer of a linked list of messages, a priority field
- Message Contents → actual message. "Hello world".

Date 17-1-24

DEADLOCKS

- A set of blocked processes, each holding a resource, and waiting for other processes to acquire another resources. The processes are deadlocked.
- It is a permanent blocking of a set of processes that either compete for system resources or communicate with each other.
- Example # 1

  Two processes P_1 & P_2. A system has two disk drives. P_1 & P_2 respectively holding one of disks. Each need other drive. Deadlocked ☹
- Example # 2

  Semaphore 'a' and 'b' → initialized to 1.

P_0	P_1
semWait(a) → process enters CS for a, a=0	semWait(b) Process enters CS for b, b=0
semWait(b) → b=0, blocked	semWait(a) a=0, blocked

Deadlocked ☹

MY SCHOOL

Date

System Model:

- A system consists of finite no. of resources to be distributed among no. of competing processes.
- Resources include Memory space, CPU cycles, files & I/O devices
- If a system has two processors, then ~~the~~ each resource type has two instances
- A process must request a resource before using it & must release the resource after using it.
- A process can request multiple resources.
- → The sequence
  - → **Request:** Process requests resource. If not available, the process must wait until grants access.
  - → **Use:** Process can operate (utilize/use the resource.
  - → **Release:** Process releases the resource to be used by another process requesting for it.
- The request & release are system calls.

Date

Deadlock Characterization:

There are some conditions which ensures deadlocks Necessary Conditions.

1. **Mutual Exclusion:** Only one process can use a resource at a time & other processes requesting it are blocked.
2. **Hold & wait:** A process must hold at least one resource and waiting for the assignment of another one
3. **No preemption:** A process can't be preempted. A resource can't be removed from a process due to priority of another process.
4. **Cicular wait:** A closed chain of processes exist, such that each process holds atleast one resource needed by the next process in chain. (necessary & sufficient condition)

   Set of processes $\{P_0, P_1, \ldots, P_n\}$.

   $P_0 \xrightarrow{\text{waits}} P_1$, $P_1 \xrightarrow{\text{waits}} P_2$, ..., $P_n \xrightarrow{\text{waits}} P_0$.

   This condition implies hold & wait condition.

- Mutual Exclusion, Hold & wait and no preemption tells possibility of a deadlock
- Adding cicular wait tells the existence of deadlo

Date

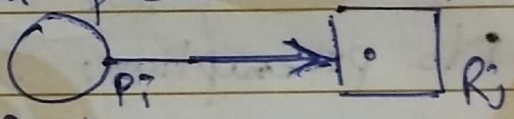
Resource Allocation Graph:

- Directed Graph, deadlock can be described using graphs.
- Also called System Resource - Allocation Graphs
- Set of vertices called nodes divided in Two Process nodes & Request nodes.

$\{P_1, P_2, ..., P_n\}$ $\{R_1, R_2, ..., R_n\}$

- A directed edge from process to a resource

$P_i \rightarrow R_j$.



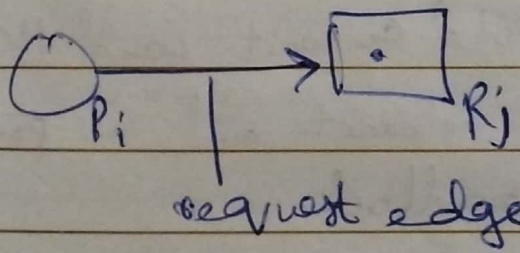
It signifies, a process P_i has requested an instance of resource R_j. It is currently waiting

- A directed edge from resource to process

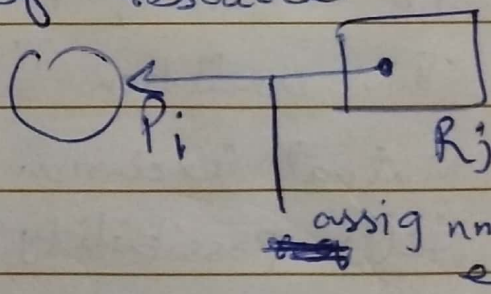
$R_j \rightarrow P_i$ or $P_i \leftarrow R_j$

It signifies that an instance of resource R_j is assigned / allocated to process P_i

- represents instance of resource



request edge.

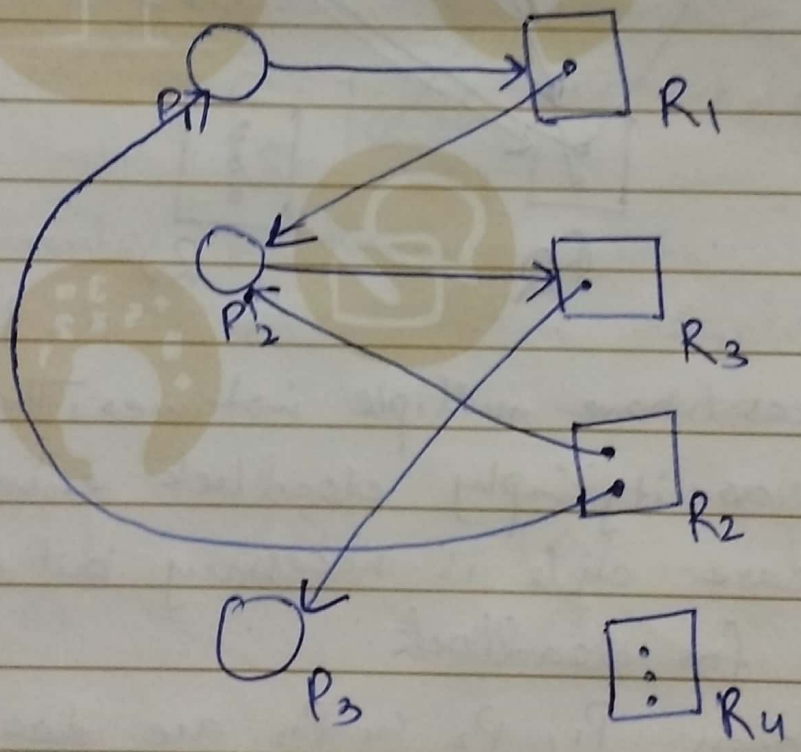


assignment edge

Date

- $P = \{P_1, P_2, P_3\}$.
- $R = \{R_1, R_2, R_3, R_4\}$.
- $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P$
  $R_3 \rightarrow P_3\}$.

R_1 (1 instance), R_2 (2 instances)
R_3 (1 instance), R_4 (3 instances)



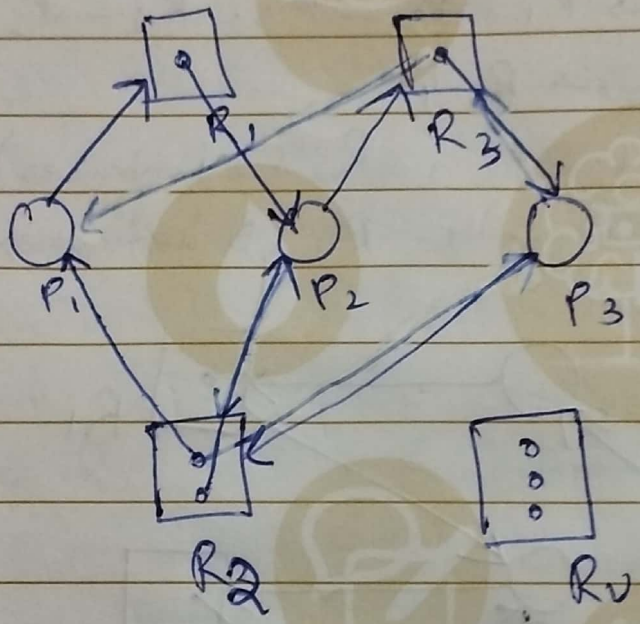
- If the graph contains a cycle, deadlock may exist
- If there is a cycle in each resource has single instance, then deadlock exists.

Cycle is a necessary & sufficient condition for deadlock occurence

Date

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

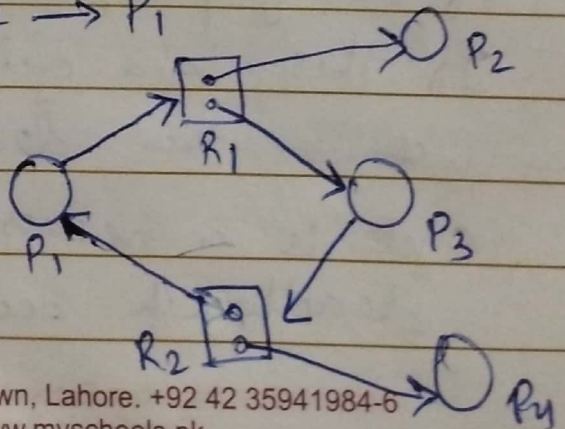
$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$.



- If resources have multiple instances, then cycle doesn't necessarily imply deadlock existence
- In this case cycle is necessary but not sufficient condition for deadlock.
- Here process P_1, P_2 & P_3 are deadlocked

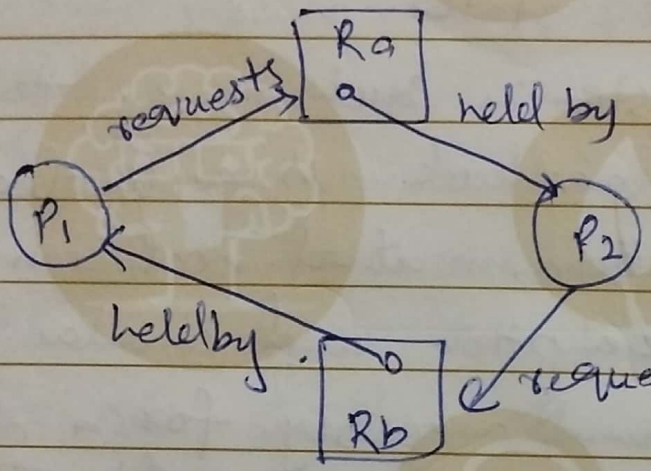
$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

- No deadlock, as soon as P_4 & P_2 release resource R_2 & R_1, they be used by P_1 & P_3.

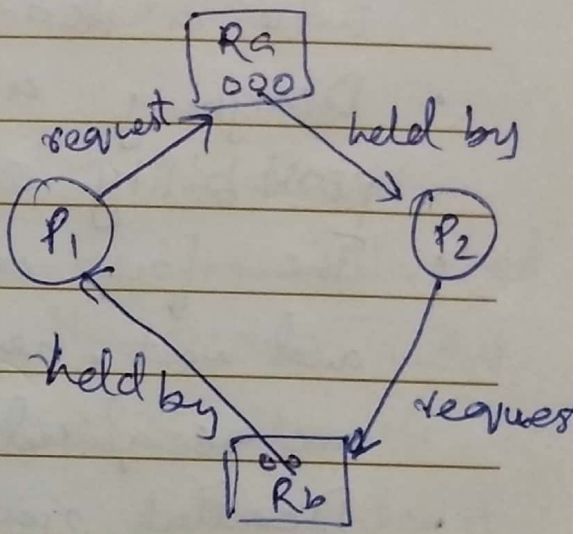


Date _____

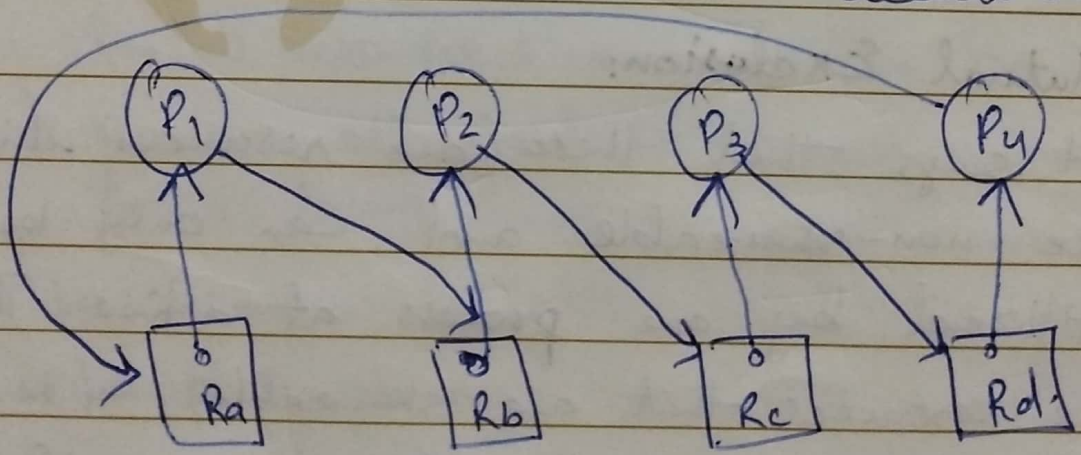
- If the graph has no cycle, no deadlock.
- If the graph contains a cycle, may or may not be deadlock



Circular wait



cycle but no deadlock.



Date

Methods for Handling Deadlocks.

① Deadlock Prevention:

- It ensures that the system will never enter into a deadlock state.
- Designing a system in such a way, that the possibility of deadlock is excluded.
- The four conditions, mutual exclusion, hold and wait, no-preemption and circular wait are required simultaneously for a deadlock.
- Deadlock prevention work that ~~any one~~ at least of the conditions not hold, prevent it, then we can prevent deadlock.

1. Mutual Exclusion:

It says that these are resources which are non-shareable and can only be utilized by one process at a time. There are resources that are shareable so hence don't require mutual exclusive access & thus can't be involved in a deadlock.

Printer is a non-shareable resource.

Date

Read-only files are a shareable resource.

→ Conclusively, we can say that it is hard to prevent mutual exclusiveness due to non-shareable resources, hence can't prevent deadlocks

2. Hold & Wait:

- It can be prevented by stating the execution after every required resource has been allocated to the process & blocking the process until it gets simultaneous access for all.
- There are two disadvantages which makes it inefficient

1 → A process is waiting for a long time, waiting for all resources requests to be fullfilled

2 → resources allocated to a process, may remain unused for a long time.

Low resource utilization, starvation is possible.

3. No preemption:

- A process P_1 holding some resources & request some other resources, if other resources can't be access, process P_1 preempt all its current

MY SCHOOL

Date

resources and request them again together with additional resources.

- If a process P_1 request a resource, which is held by process P_2, then OS may preempt resources from P_2
- This approach is practical only if the resources state can be saved & restored easily.

4. Circular Wait:

- It can be prevented by defining total ordering / linear ordering of resources types.

Let tape drive = 1
disk drive = 5.
printer = 12.

a process can only request next resources in ascending order.

If process P_1 has asked for disk drive which is 5, then it can't access tape drive because. (1 < 5) but can access printer because, it is 5 < 12

Date

@ Deadlock Avoidance:

- With deadlock avoidance, a decision is dynamically made, whether the current resource allocation request, if granted, will lead to a deadlock.
- It requires future process resource requests.
- The system will know the complete sequence of requests, it can decide whether or not a process should wait in order to avoid deadlock
- To implement this, the system will have information of available resources, resources allocated to each process & future requests.
- The simplest & useful model requires that each Process declare the maximum no. of resources of each type, it may need.
- Deadlock avoidance algorithm dynamically examines resource-allocation states, to ensure that a circular wait condition can never exist.
- Resource allocation state: no. of available & allocated resources & max. no. of required resources

Date

Safe state:

- System can allocate resources in some order in no deadlock is encountered.
- Safe sequence → system is in safe state
- Pi requesting resources & gets them, in Pj is not affected. It is safe sequence.
- If Pi requires a resource in it is not available, then it can wait, until Pj has finished, then it will take them in continues its execution & terminates, the Pi+1 continues & so on.
- If there is no such sequence, then it is unsafe.
- An unsafe state may lead to a deadlock.
- We have to avoid the system to enter in an unsafe state.

Date _____

MY SCHOOL

(B) Deadlock Detection:

- It follows the approach, that deadlock has occured, now check the deadlock and apply an algorithm to recover back from it.
- Detection & recovery has to deal with various overheads.
  1. Runtime cost to maintain the information by executing detection algorithm
  2. Potential losses inherent in recovering from a deadlock.

Recovery from Deadlock:

→ After detecting deadlock, operator can be informed about deadlock & it will be recover manually.

→ Another approach is to recover automatically.

1. Abort / Process termination.
2. Resources preemption.

Date

Process termination:

(a) Terminate all deadlocked process at once.
- May be, the process have computed for a long time.

(b) Terminate one process at a time until the deadlock cycle is eliminated.
after one termination, deadlock detection algorithm will check if deadlock is still or not. This reduces the overhead.

Resource Preemption:

- Preempt the resources from a process until the deadlock cycle has broken.
- Preempting resources requires following 3 issues to be addressed.

(1) Selecting a victim:
which processes & which resources are to be preempted?

(2) Rollback: after preemption, what should be done with that process? Roll it back

Date

to a safe state; totally; Restart it, ready queue

(b) Starvation: How to ensure that resources will not always be preempted from same process

cost factors. only a finite no of times, a process should be chosen as victims to avoid starvation.

add no. of roll backs in cost factor.

MY SCHOOL

Date 25-01-24.

MEMORY MANAGEMENT

- In uniprogramming system.
  main memory is divided into two parts.
  ① one part for OS
  ② one part for program currently executing.
- In multiprogramming systems, the user part of memory is further subdivided to accomodate multiple processes.
- This subdivision is dynamically carried out by OS, it is called memory management.

Memory Management teems

Frame: A fixed-length block of main memory.

Page: A fixed-length block of data that resides in secondary memory (disk). A page of data may ~~contain~~ tamporarily be copied into a frame of main memory.

Date _____

MY SCHOOL

Segment: A variable-length block of data that resides in secondary memory. An entire segment may temporarily be copied into an available region of main memory (segmentation) or the segment may be divided into pages which can be individually copied into main memory (combined segmentation or paging).

Memory:

- Collection of data in a specific format
- Consists of a large array of words or bytes, each with its own address.
- Internal storage area of the computer.
- It stores all kind of data or information

Types of Memory:

1. **Physical Memory:**
   - primary or main memory, hardware component inside a computer
   - tangible & volatile (loses its contents when

Date

the power is turned off.
- directly accessible by CPU & other hardware components.
- Ex: RAM, ROM.

2. Logical Memory:
- a memory space that a program can use.
- an abstraction of physical memory that the OS ~~can~~ provides to applications & processes
- Programs can access to memory addresses
- Provides a logical address space for each process, independandent of actual physical memory
Ex: CPU memory → cache, embedded memory

3. Virtual Memory:
- An imaginary memory area, supported by some OS (eg Windows).
- An extension of logical memory.
- VM is a memory management technique that provides an illusion to applications that they have access to large, contiguous block of memory even if the physical memory is fragmented / or insufficient.

Date _____

MY SCHOOL

- allow running processes to use more memory than physically available.
- using disk storage as an extension of physical RAM memory

→ If the size of program is > available memory (Physical one) then virtual memory is used.

Available Memory:

- Amount of Physical memory not in use.
- Loading the OS takes up memory as well.
- When system boots up, available memory amount drops.
- Each time you open a program, it is loaded into your computer's memory, & therefore reduces your available memory.

Memory Management:

- Functionality of an OS to handle & manage primary memory (RAM eg)
- Subdividing available memory among different processes is called memory management.

Date

- keep track of all available memory locations that allocation & deallocation either it is allocated to some process or not
- decides which process will get which part of memory at any time.
- OS need to achieve to broad tasks

Memory-Management-Unit (MMU)

1. Each process must have enough memory space to execute & a processes can not run into ~~neither be~~ memory space of another process and not be run into by another process.
2. There are different types of memory in the system & all must be used properly to to achieve effectiveness.

Memory Management Unit (MMU)

How OS handles: RAM ?...

→ It keeps tracks of parts of memory in use. memory space

→ Allocate & deallocate memory to & from processes when needed

→ Swapping & paging b/w main memory & disk

Date _____

- The principal operation of memory management is to bring processes into main memory for execution by the processor.

Static & Dynamic Memory:

① Static Memory: SRAM - Static RAM
faster, less volatile
requires more power & it is expensive.

② Dynamic Memory: DRAM - Dynamic RAM.
RAM must be constantly refreshed, reenergized or it'll loose its contents.

Partitioning:

① Fixed Partitioning. The available memory, for use by processes. The simplest scheme is to partition available memory into fixed size regions with fixed boundaries.
It is static. A process may be loaded into equal or greater size of a partition.
It is simple & little OS overhead.

Date

Internal fragmentation due to fixed sizes → inefficient use of memory.

~~(2) Dyna~~

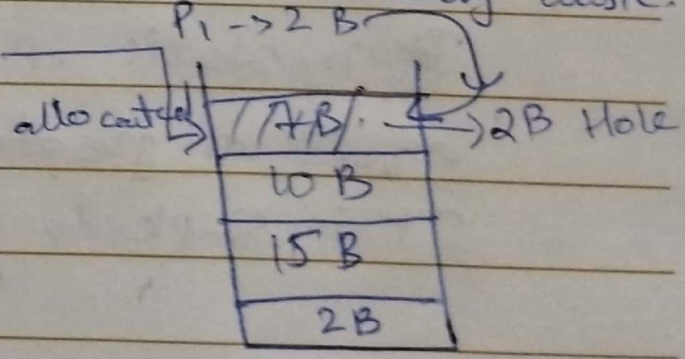
(3) Dynamic Partitioning:

- Partitions are created dynamically, a process loaded into partition will be of same size.
- No internal fragmentation.
- Inefficient use of processor due to compaction to counter external fragmentation.

3 Ways in which data is put in RAM Placement algorithms:

(1) First Fit:

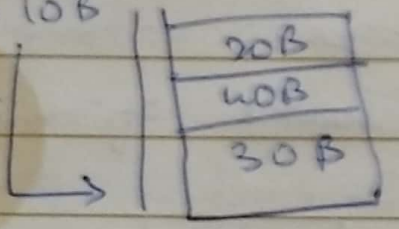
- Begins to scan memory from the begining and chooses the first available block that is large enough.
- Faster in allocation but leads to memory waste.



Date

2. Worst Fit:

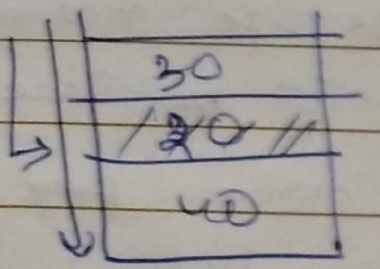
- It scans the entire available memory & allocate process in the largest block.
- Least efficient, waste of Memory & time
- It make big wholes $P_1 \rightarrow 10B$ in memory. & slow as it traverse whole list.



3. Best Fit:

- Chooses the block that is closest in size to the requested process
- Best use of memory.
- Slow because traverse whole list than allocate.
- It'll choose closest blocks, but it can make small wholes in each block, . . .
- Less space but more time (times space tradeoff)

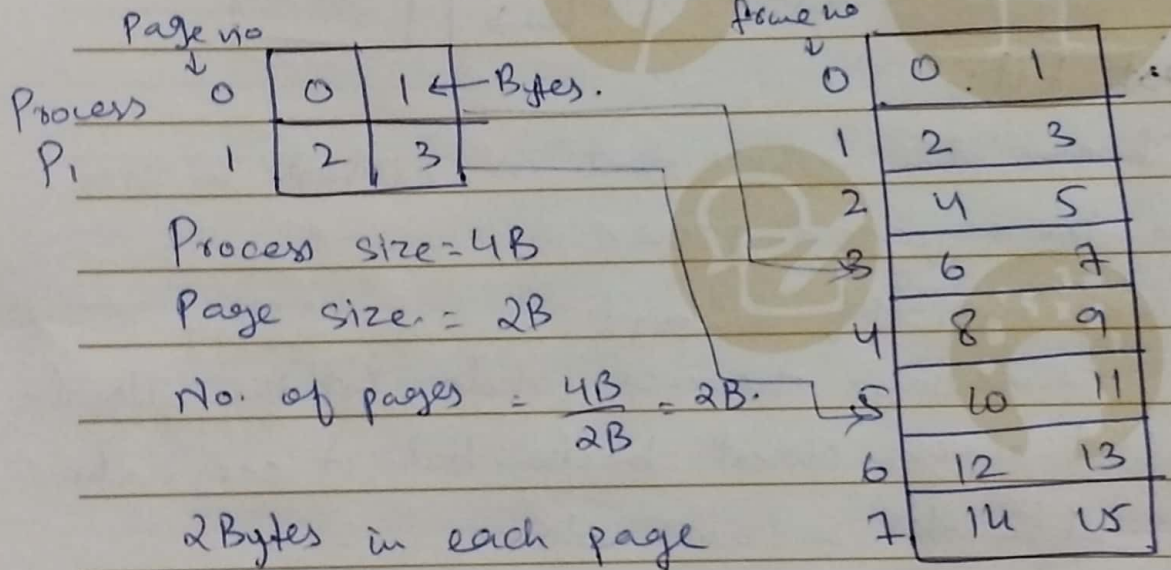
$P_1 = 18B$



Date

Paging:

- Another memory management technique.
- A logical concept.
- A process is divided into small chunks called pages.
- Chunks of memory are called frames.



Process size = 4B

Page size = 2B

No. of pages = $\frac{4B}{2B}$ = 2B.

2 Bytes in each page

Page size & frame size should be same

MM. Size = 16B

Frame size = 2B.

No. of frames = 16/2 = 8 frames

8B

Let assume, page 0 is in fram 3.
& pge 1 is in fram 5

When CPU need any data of process, how do it know that where it is in main memory?

Date

- A mapping function is used to convert the address of that required data by CPU, that is actually page's data into frame's address of main memory.
- MMU does the mapping function. It uses page table.
- Page table contains frame numbers that where that page is actually present in the main memory

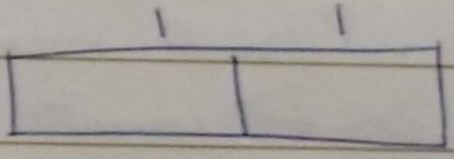
Page table of a Process P_1

Page no	Frame no
P0	F3
P1	F5

P_1

Each process will have its own page table.

CPU works with logical address, logical address is made up of page number & page offset



no. of pages ~~size~~ 2 / 2 number 0, 1
0, 1 can be rep. in 1 bit.

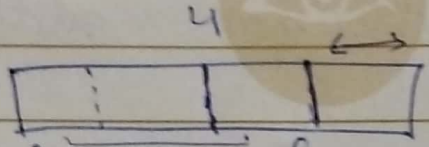
Page no => 0, 1 => 1 Bit

Page size contain 2 Bytes mem 0, 1

Date

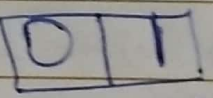
Physical address is for main memory.
Logical addresses are converted into physical addresses

Physical address.

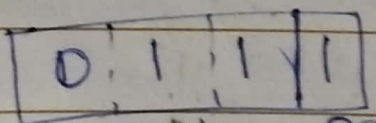


Our MM is of 16 Bytes, 16 can be represented in binary in 4 bits.
Physical address is of 4 bits.
Frame size is 2 Bytes, 2 can be represented in binary in 1 bit
frame offset 1 bit.
3 bit for frame no. 1 bit for offset.

as in our example CPU asks for ~~Byte~~ 1 Byte its logical address =>



In Page table, Page 0 has Frame 3.
The mapping function converts it into physical address ->



return 7 Byte from MM.

Date

MY SCHOOL

Frames.

- Frames are physical concept.
- In RAM.
- Small chunks are called frames in RAM.
- We can't see pages but we can see frames.

Mapping

- Converting virtual or logical address of pages by CPU into physical address of frames in MM.
- Performed by MMU.
- If an error occurs at any physical address, it retrieves back to CPU

Paging Protection:

- Paging process is protected by adding valid/invalid bit
- Consider -> 14bit address space.

$$2^{14} = 16383 \text{ bytes.}$$

- address limit is set to 10468.

Date

- If five process P_0, P_1, P_2, P_3, P_4 are defined within this address space (i.e. their address will be less then 10468) then it is valid bit.
- P5 has started before 10468, it also is considered
- Remaining processes are invalid.
- Pages are internally fragmented in this way.
- The bit is in Page table.

Advantage & Disadvantages of Paging.

⭐ ① No external fragmentation.

☺ ② Simple memory management algo.

③ Easy swapping (equal sized pages & frames)

☹ ① Internal fragmentation.

② Page table consumes memory.

The wasted space in memory for each process is due to internal fragmentation consisting of only a fraction of the last page of a process.

Date

MY SCHOOL

Fragmentation:

As process are loaded & removed from memory, the available memory space is broken into little pieces. hence, it is not utilized or wasted. This is fragmentation in memory.

External fragmentation:

When there is enough memory space to satisfy a request but the available spaces are not contiguous; storage is fragmented into a large number of small holes.

Internal fragmentation:

If memory is divided into fixed patitions & process are loaded in the partitions; the process sizes are smaller then the partition, there is wasted space internal to the partition due to the fact that the block of data loaded is smaller then the partition.