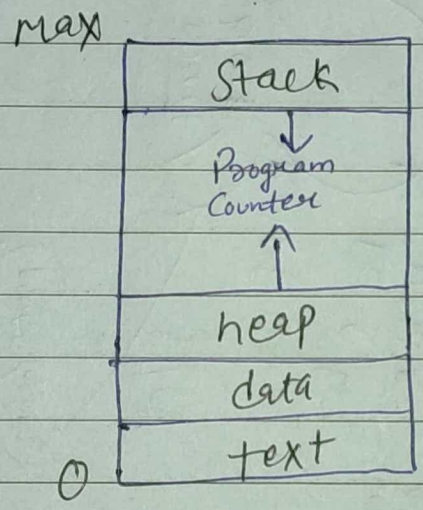
Page No.:
Date: 30/12/21

Unit-2
Process Management

Process - Program is passive entity stored on disk (executable file) while process is active entity.
Program become process when executable file loaded in memory (RAM).
One program can be several process.

Parts of Process -

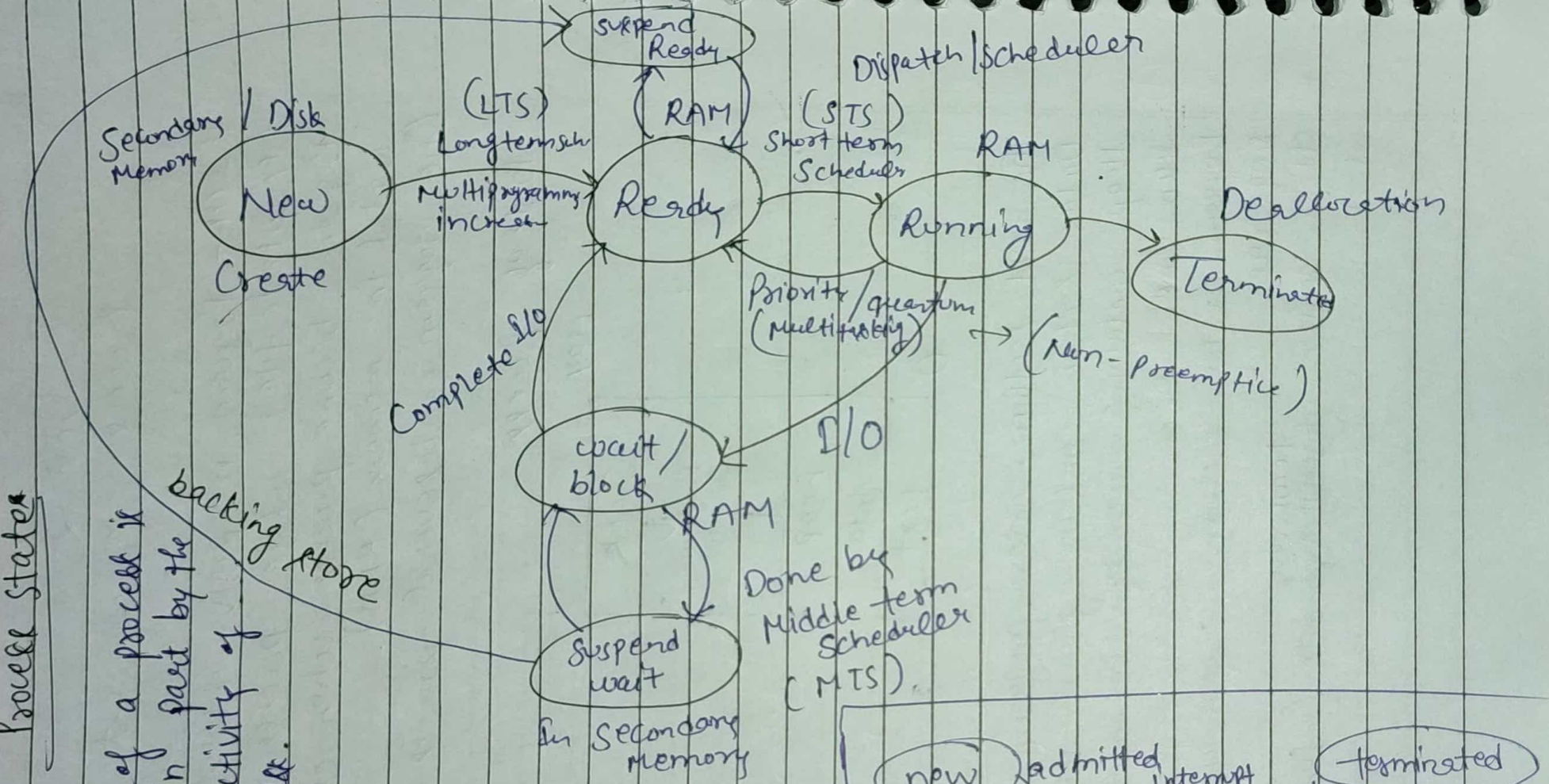


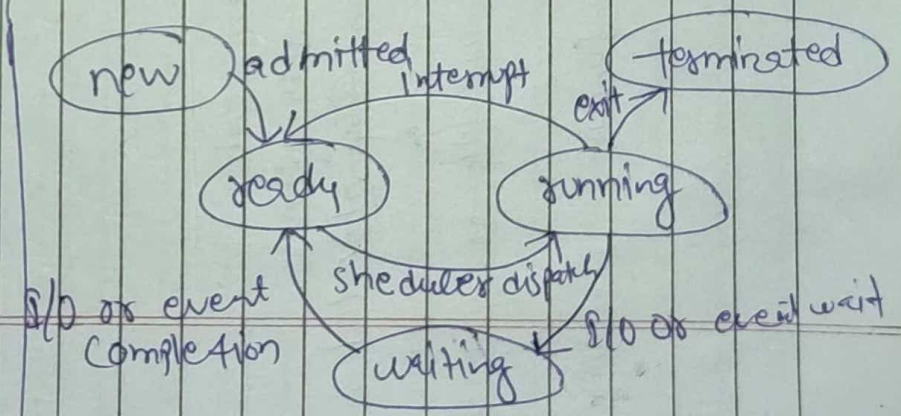
The text section contain program code.
Data section contain global variable
Heap containing memory dynamically allocated during run time.
Program counter includes current activity of process registers.
stack contain temporary data (function, local variable, etc).

Page No.:
Date: / /

Process States

The state of a process is defined in part by the current activity of that process.





Degree of Multiprogramming control with Long term scheduler.

Page No.:
Date: / /

As a process execute it changes state -

1. New - The process is being created.
2. Running - Instructions are being executed.
3. Waiting - The process is waiting for some event to occur.
4. Ready - The process is waiting to be assigned to a processor.
5. terminated - The process has finish execution.

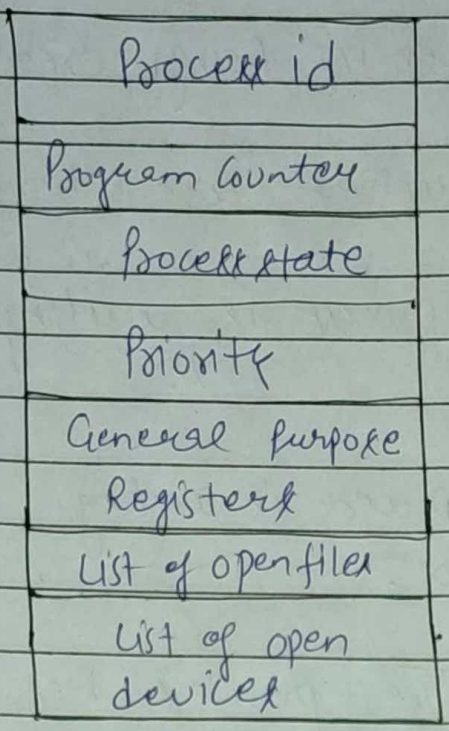
03/11/22

Process Control Block

The area or datastructure where process attribute are stored is known as PCB.
It is also known as task control block or process discriptor.
Each process contain its own PCB.
Process attribute include process id, program counter, general purpose register, procority, process state, process type etc.

Page No.:
Date: / /

PCB



(1) Processid - Processid contain the process number. It is decided when process is created.

(2) Program Counter - It is the location of next instruction that we have to execute.

(3) Process state - It is basically running, waiting etc all the state of process

(4) Priority - It is the no. that tells which process has to execute when.

(5) General purpose register - It contains all process centric registers.

Page No.:
Date: / /

Thread

Process has a single thread of execution. We can consider having multiple program counter per process.

It can have Multiple location that can execute at once.

	Process	Thread
①	System call involved in process.	① There is no system call involved.
②	OS treat diff. process differently.	② All user level thread treated as single task for OS.
③	Different process have different copies of data, file, code.	③ Thread share the same copy of code & data.
④	Context switching is slower in process.	④ Context switching is faster in thread.
⑤	Blocking a process will not block others.	⑤ Blocking a thread will block entire process.
⑥	Each process is independent.	⑥ Thread is interdependent.

Context switching order

~~Process > kernel > User~~

Process > Kernel > User

Page No.:
Date: / /

User level thread Vs Kernel level thread

User level thread	Kernel level thread
① It is managed by user level library.	① These are managed by OS system call.
② User level thread are typically fast.	② Kernel level thread are slower than user level thread
③ Context switching is faster.	③ Context switching is slower
④ If one user level thread perform blocking operation then entire process get blocked.	④ If one kernel level thread blocked no affect on other

Interprocess Communication - Processes within a system maybe independent or cooperating. Cooperating process can affect or be affected by other process including sharing data. Independent process cannot affect or be affected by the execution of another process.

Reasons for Cooperating process —

① Information sharing

② Computation speed ③ Modularity

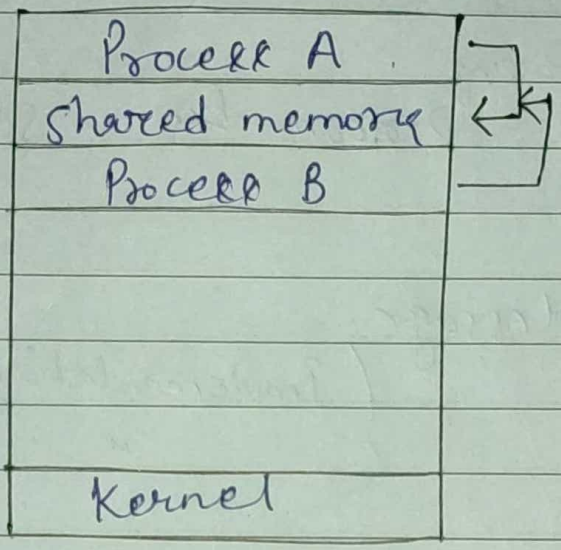
④ Convenience

Co-operating process required an interprocess communication mechanism that will allow them to exchange data & information.

There are two fundamental MODEL IPC

1. Shared memory
2. Message passing

1. Shared Memory -



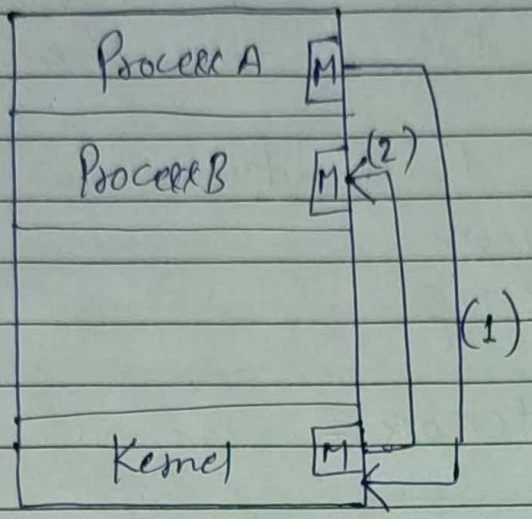
Interprocess communication using shared memory requires communicating process to establish a region of shared memory.

Typically a shared memory region resides in the address space of the process creating the shared memory segment.

Other process that wish to communicate using this shared memory segment must attach it to their address space.

Page No.:
Date: / /

(2) Message Passing -



Message Passing

~~Operation~~ perform by send / receive operation.

Types of Message -

(1) Fixed (Implementation easy)
(2) Variable (" difficult)

Message passing provide a mechanism to allow processes to communicate & to synchronise their action without sharing the same address space. & it is particularly use in a distributed environment where the communicating process may reside on different computer connected by network.

A message passing facility provide two operation

(1) Send
(2) Receive

Page No.:
Date: / /

Message send by a process can be of either fixed or variable size.
fixed size = System level implementation is straightforward or easy but programming task is difficult.

Variable size = System level implementation is complex but the programming task become simpler.

If process P & Q want to communicate, they need to establish a communication link b/w them. which exchange message via send/receive operation.
Implementation of communication link include -

(1) Physical
(a) Shared Memory
(b) Hardware Bus
(c) Network

(2) Logical
Direct or Indirect
Synchronous or Asynchronous
Automatic or explicit buffering

Page No.:
Date: 11/01/2022

Direct Communication

With direct communication each process that requires to communicate must explicitly name the recipient or sender of the communication.

Processes must name each other explicitly :
- 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.
- Between each pair there exists exactly one link
- The link may be unidirectional, but is usually bi-directional

Indirect Communication

- Messages are directed and recieved from mailboxes
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox.
- Properties of communication link
  - Link established only if processes share a

Common mailbox.

- A link may be associated with many processes
- Each pair of processes may share several communication links.
- Link may be unidirectional or bi-directional

- Operations
  - create a new mailbox (port)
  - send and receive messages through mailbox
  - destroy a mailbox

- Primitives are defined as:
  send (A, message) - send a message to mailbox A
  receive (A, message) - receive a message from mailbox A

Synchronization

- Process Synchronization is a way to coordinate processes that use shared data.
- It occurs in an operating system among cooperating processes.
- The need for synchronization originates when processes need to execute concurrently.
- The main purpose of synchronization is the sharing of resources without interference using mutual exclusion.
- Message passing may be either blocking or

non-blocking.

- Blocking is considered synchronous
  - Blocking send - the sender is blocked until the message is received
  - Blocking receive - the receiver is blocked until a message is available.

- Non-blocking is considered asynchronous
  - Non-blocking send - the sender sends the message and continue
  - Non-blocking receive - the receiver receives:
    - A valid message, or
    - Null message

- Different combinations possible
  If both send and receive are blocking, we have a rendezvous

- Producer-consumer becomes trivial

```
message next_produced;
while (true) {
    send (next_produced);
}

message next_consumed;
while (true) {
    receive (next_consumed);
}
```

Buffering

- A Buffer is a temporary storage area. Generally, buffer stores the data when the data transferred b/w two devices or one device & one application.
- Queue of messages attached to the link.

Buffering is done for 3 reasons:-

1. If the producer device is a high speed, the consumer device is the low speed at that time buffering is required.
2. If two devices have different data transfer size, then buffering required.
3. Support copy semantic for an application I/O

- Implemented in one of three ways -
  1. Zero capacity - no messages are queued on a link. Sender must wait for receiver.
  2. Bounded capacity - finite length of n messages Sender must wait if link full
  3. Unbounded capacity - infinite length Sender never waits

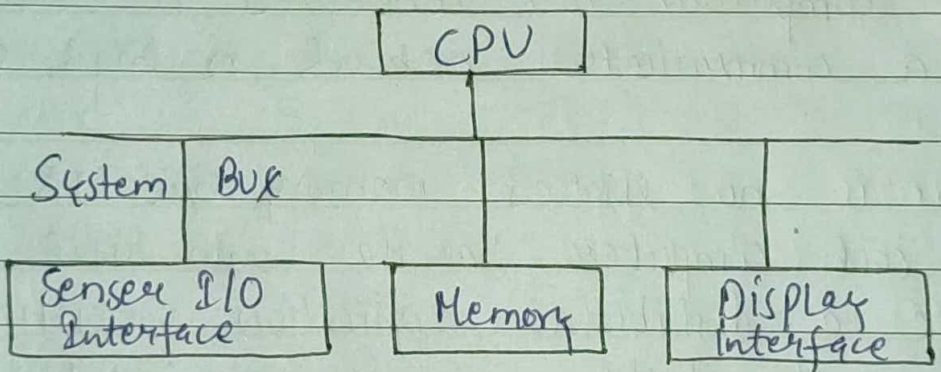
Concurrent Process

- Concurrent processing is a computing model in which multiple processors execute instructions

simultaneously for better performance.
Concurrent means, which occurs when something else happens.
The tasks are broken into sub-types, which are then assigned to different processors to perform simultaneously, sequentially instead, as they would have to be performed by one processor.
Concurrent processing is sometimes synonymous with parallel processing.

Multiprogramming Environment:

In multiprogramming environment, ther are multiple tasks shared by one processor. while a virtual concert can be achieved by the operating system, if the processor is allocated for each individual task, so that the virtual concept is visible if each task has a dedicated processor.

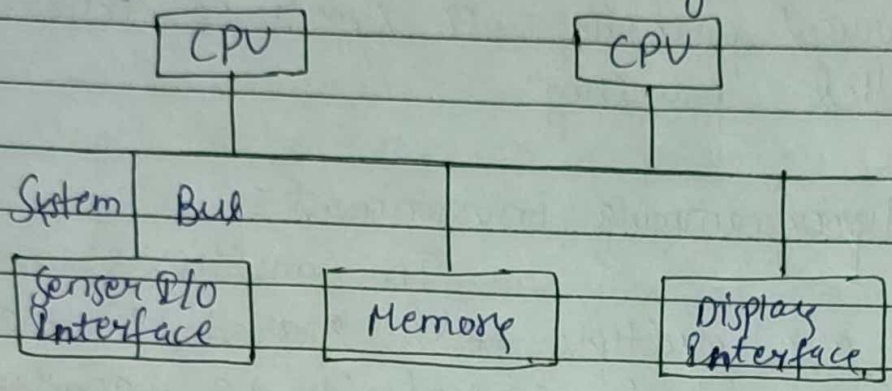


Multiprogramming (single CPU) Environment

Multiprocessing Environment - In multiprocessing environment two or more processors are used with shared memory.

- Only on virtual address space is used, which is common for all processors.
- All tasks reside in shared memory.

In this environment, concurrency is supported in the form of concurrently executing processors. The tasks executed on different processors are performed with each other through shared memory.

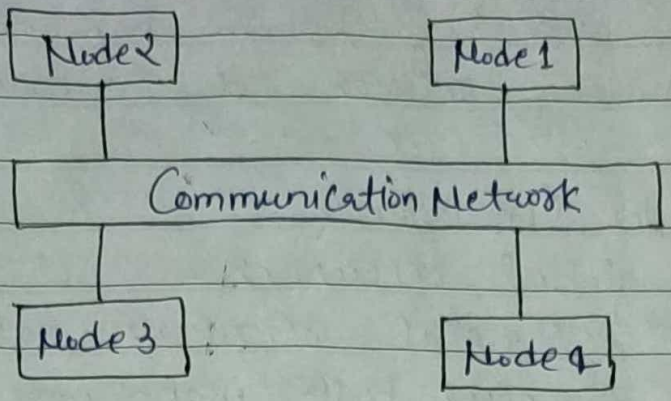


Multiprocessing Environment

Distributed Processing Environment: In a distributed processing environment, two or more computers are connected to each other by a communication network or high speed bus.

There is no shared memory b/w the processors and each computer has its own local memory. Hence a distributed application consisting of concurrent tasks, which are distributed over network communication via messages.

→



Distributed processing Environment

Concurrency :- Concurrency is the execution of the multiple instruction sequences at the same time.
It happens in the operating system when there are several process threads running in parallel.
The running process threads always communicate with each other through shared memory or message passing.
Concurrency results in sharing of resources result in problems like deadlocks & resources starvation.
It helps in techniques like coordinating execution of processes, memory allocation and execution scheduling for maximizing throughput.

Principles of Concurrency :
Both interleaved and overlapped processes can be viewed as example of concurrent processes.
The relative speed of execution cannot be predicted.
It depends on the following :

- The activities of other processes

- The way operating system handles interrupts
- The scheduling policies of the operating system

Problems in Concurrency :

- Sharing global resources -
Sharing of global resources safely is difficult. If two processes both make use of a global variable & both perform read and write on that variable, then the occur in which various read and write are executed is critical

- Optimal allocation of resources -
It is difficult for ~~locate~~ the operating system to manage the allocation of resources optimally.

- Locating programming errors -
It is very difficult to locate a programming errors because reports are usually not reproducible.

- Locking the channel -
It may be inefficient for the operating system to simply lock the channel and prevents its use by other processes.

Advantages of Concurrency -

- Running of multiple applications -
It enable to run multiple applications at the same time

- Better resource utilization -
It enables that the resources that are unused by one application can be used for other applications.

- Better average response time -
without concurrency, each application has to be run to completion before the next one can be run.

- Better performance -
when one application uses only the processor and another application uses only the disk drive then the time to run both applications concurrently to completion will be shorter than the time to run each application consecutively.

Drawbacks of Concurrency:
- It is required to protect multiple applications from one another.
- It is required to coordinate multiple applications through additional mechanisms.
- Additional performance overheads and complexities in operating systems are required for switching among applications.
- Sometimes running too many applications concurrently leads to severely degraded performance.

Page No.:
Date: / /

Issues of Concurrency -

- **Non-atomic -**
Operations that are non-atomic but interruptible by multiple processes can cause problems.

- **Race Condition -**
A race condition occurs of the outcome depends on which of several processes gets to a point first.

OR

A race condition is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device, operations must be done in the proper sequence to be done correctly.

- **Blocking -**
Processes can block waiting for resources. A process could be blocked for long period of time waiting for input from a terminal. If the process is required to periodically update some data, this would be very undesirable.

- **Starvation -**
It occurs when a process does not obtain service to progress.

OR

Starvation is the problem that occurs when high priority processes keep executing &

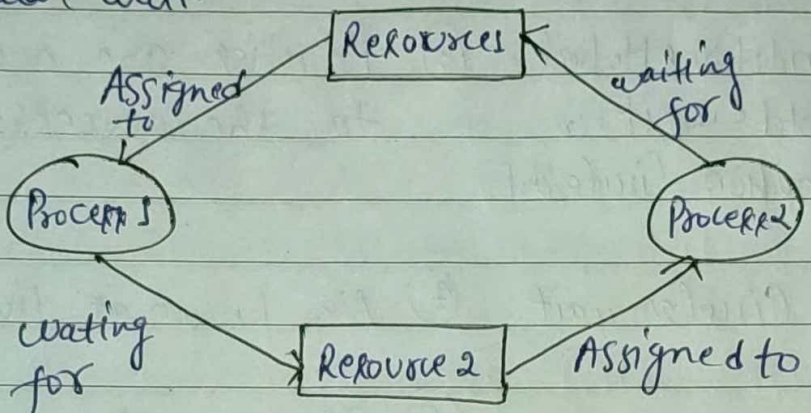
low priority processes get blocked for indefinite time

- Deadlock - It occurs when two processes are blocked and hence neither can proceed to execute.

OR

Deadlock occurs when each process holds a resource and wait for other resource held by any other process.

Necessary conditions for deadlock to occur are Mutual exclusion, Hold & wait, No Preemption, Circular wait.



Here, Process 1 is holding Resource 1 & waiting for R2 which is acquired by P2, & P2 is waiting for resource 1. Hence both P1 & P2 are in deadlock.

Page No.:
Date : / /

Diff b/w Deadlock & Starvation -

	Deadlock	Starvation
①	All processes keep waiting for each other to complete & none get executed	① High priority processes keep executing and low priority processes are blocked
②	Resources are blocked by the processes.	② Resources are continuously utilized by high priority processes
③	Necessary conditions Mutual Exclusion, Hold & wait, No preemption, Circular wait	③ Priorities are assigned to the processes.
④	Also known as Circular wait	④ Also known as livelock
⑤	It can be prevented by avoiding the necessary conditions for deadlock.	⑤ It can be prevented by Aging.

Race Condition

- Counter++ could be implemented as

```
register = counter
register = register + 1
counter = register
```

Page No.:
Date: / /

- counter-- could be implemented as
  register 2 = counter
  register 2 = register 2 - 1
  counter = register 2

- Consider this example with count = 5 initially:

S0: producer execute register 1 = counter {register 1 = 5}
S1: " " " = register 1 + 1 {" = 6}
S2: Consumer execute register 2 = counter {register 2 = 5}
S3: " " " = register 2 - 1 {" = 4}
S4: producer execute counter = register 1 {counter = 6}
S5: consumer execute counter = register 2 {counter = 4}

Page No.:
Date: / /

Critical Section Problem

Critical section - It is part of the program where shared resources are accessed by various processes (cooperating process).
It is a place where shared variables, resources are placed.

- Consider system of n processes {$P_0, P_1, \ldots, P_n$}
- 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

General structure of process P_i

```
do {
        entry section
            critical section
        exit section
            remainder section
} while (true);
```

Page No.:
Date: / /

Algorithm for process P_i

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

Solution to Critical-Section Problem

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
   - Assume that each process executes at a nonzero speed
   - No assumption concerning relative speed of the n processes

Critical-Section Handling in OS

Two approaches depending on if kernel is preemptive or non-preemptive

- Preemptive - allows preemption of process when running in kernel mode
- Non-preemptive - runs untill exists kernel mode, blocks, voluntarily yields CPU
- Essentially free of race conditions in kernal mode

~~Peterson~~

Peterson's Solution

- Good algorithmic description of solving the problem
- Two process solution
- Assume that the load and store machine-language instructions are atomic; that is cannot be interrupted
- 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 P_i is ready!

Page No.:
Date: / /

Algorithm for Process P_i:

```
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
   Pi enters CS only if:
   either flag[j] = false or turn = i
2. Progress requirement is satisfied
3. Bounded-waiting requirement is met

Synchronization Hardware

- Many systems provide hardware support for implementing the critical section code.
- All solutions below based on idea of locking
  → Protecting critical regions via locks
- Uniprocessors - could disable interrupts
  - Currently running code would execute without preemption
  - Generally too inefficient on multiprocessor systems
    → Operating systems using this not broadly scalable

- Modern machines provide special atomic hardware instructions
  → Atomic = non-interruptible
  → Either test memory word & set value
  → Or swap contents of two memory words

Solution to Critical-section Problem using locks -

```
do {
    acquire lock
        Critical section
    release lock
        remainder section
} while (TRUE);
```

test & set instruction

Definition:

```
boolean testandset (boolean *target)
{
    boolean rv = *target;
    *target = TRUE;
    return rv;
}
```

1. Executed atomically
2. Returns the original value of passed parameter
3. Set the new value of passed parameter to "TRUE".

Page No.:
Date: / /

Solution using test_and_set()

- Shared Boolean variable lock, initialized to FALSE
- Solution:

```
do {
    while (test_and_set (&lock));
        /* do nothing */
        /* critical section */
    lock = false;
        /* remainder section */
} while (true);
```