Such systems are multiprocessor systems also known as tightly coupled systems. It deals with multiple computer resources that can include a single computer with multiple processors.

Q2. What is System call ? Discuss six major categories of system call with suitable example. Explain various steps to execute a system call.

A. System calls provide an interface to the services made available by an operating system. These calls are generally available as routines written in C and C++. A computer program makes a system call when it makes a request to the operating system's kernel. System call provides the services of the operating system to the user programs via Application Program Interface.

System calls can be grouped into six major categories :

(i) Process Control - A running program needs to be able to halt its execution either normally or abnormally. If the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message is generated. A process or job executing one program may want to load and execute another program. Allocation of memory and freeing of memory is done by process control

Example : In Windows, CreateProcess(), ExitProcess() WaitForSingleObject(). In Unix, fork(), exit(), wait().

(ii) File Management – Some common system calls are creation and deletion of files, read, write, reposition or closing of files. Also there is a need to determine the file attributes by get and set file attribute. Some operating systems provide many more calls, such as calls for file move and copy.
Example :- In Windows, CreateFile(), ReadFile(), WriteFile(), CloseHandle().
In Unix, open(), read(), write(), close().

(iii) Device Management – Process usually require several resources to execute, if these resources are available, they will be granted and control is returned to the user process. Otherwise, the process will have to wait until sufficient resources are available. These resources are thought of as devices. User programs request the device, and when finished they release the device.
Example: In Windows, SetConsoleMode(), ReadConsole(), WriteConsole().
In Unix, read(), write()

(iv) Information Maintenance – Many system calls exist simply for the purpose of transferring information between the user program and the operating system. Many systems provide system calls to dump memory which is useful for debugging. Also some provide a time profile of a program to indicate the amount of time that the program executes at a particular location.

Example: In Windows, GetCurrentProcessID(), SetTimer(), Sleep. In Unix, alarm(), sleep(), getpid().

(v) Communication - There are two models of interprocess communication, the message-passing model and the shared memory model.
- Message-passing uses a common mailbox to pass messages between processes.
- Shared memory uses a some certain system calls to create and gain access to create and regions of memory owned by other processes. The two processes exchange information by reading and writing in the shared data.

Example: CreateFileMapping(), CreatePipe(), in Windows.
In Unix, pipe(), mmap().

(vi) Protection - Protection provides a mechanism for controlling access to the resources provided by a computer system. System calls providing protection include set permission and get permission, which manipulate the permission settings of resources such as files and disks. The allow user and deny user system calls specify whether particular users can - or cannot - be allowed access to certain resources.

Various steps to execute a system Call:
The system call constitutes of three parameters, file name, pointer to the buffer and number of bytes to read.

Step 1 : When the system call is executed, the parameters are pushed onto the stack and later on saved in process register.

Step 2 : The corresponding library procedure is then executed.

Step 3 : There is a particular code for every system call by which kernel identifies which system call handler needs to be executed.

Step 4 : TRAP instruction is executed that switches from user mode to kernel mode.

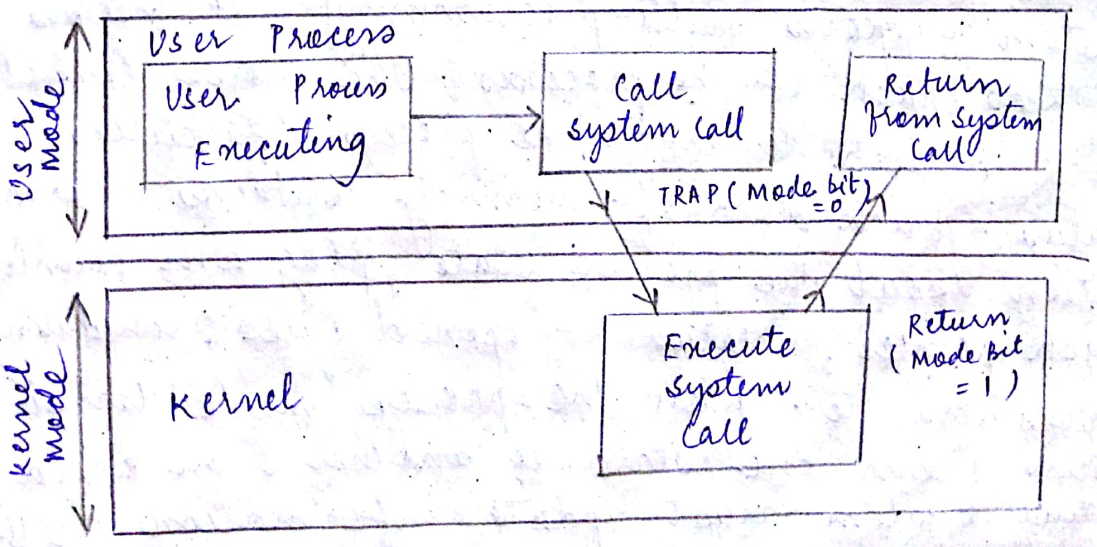
Step 5 : The hardware saves the current contents of CPU registers.

Step 6 : Then finally, system call handler executes

Step 7 : After the completion of the task by system call handler, the control is returned back to the user-space library procedure.

Step 8 : Then the control is transferred to the user program from the read procedure.

In this way the job of read system call is completed.



Q3. What is Process Control Block ? Explain with suitable diagram. Draw the state diagram of a process from its creation to termination, including all transitions, and briefly elaborate every state and every transition.

A.

Process Identification Number (PID)
Process State
Program Counter
Registers
Memory Information
Resource Related Information
Scheduling Related Information
Input - Output Related Information
Other files
⋮

Process Control Block (PCB)

Each process is represented in the operating system by a process control block (PCB) - also called a task control block. As the operating system supports multi-programming, it needs to keep track of the processes. The Process Control Block is used to track the process's execution status. Each block of memory contains information about the process state, program counter, stack pointer, status of opened files, scheduling algorithms, etc. When the process made transitions from one state to another, the operating system must update information in the

Process Control Block:

- Process state - The state may be new, ready, running, waiting, halted, and so on.
- Program counter - The counter indicates the address of the next instruction to be executed for this process
- CPU Registers - They include accumulators, index registers, stack pointers, and general-purpose registers. Along with the program counter this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.
- Memory-management information - This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system.
- CPU-scheduling information - This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- Input-Output related information - This information includes the list of input-output devices allocated to the process, a list of open files, and so on.
- Resource related information - This information includes the management of resources like memory, processor and all the input-output devices.

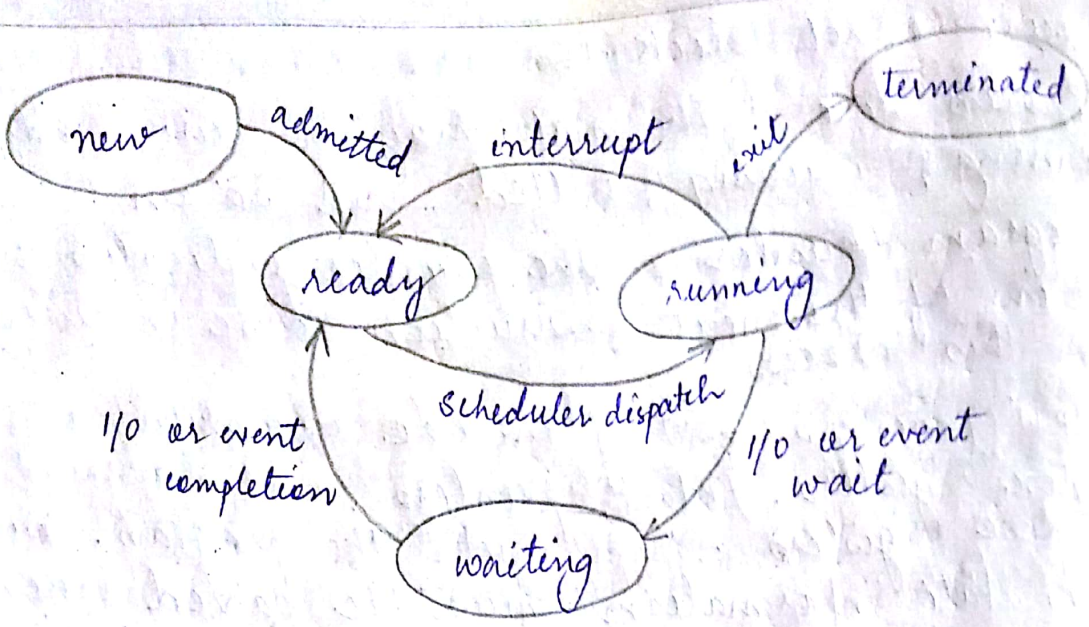


Diagram of process state

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

- New – This is the initial or start state when a process is first started / created.
- Ready – The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after start/ new state or while running it by but interrupted by the scheduler to assign CPU to some other process.
- Running – Once the process has been assigned to a processor by the operating system scheduler, the process state is set to running and the processor executes its instructions.
- Waiting – The process is waiting for some event to occur (such as an I/O completion or reception of a signal). It waits for a resource such as waiting for user input,

or waiting for a file to become available.

- Terminated or Exit – Once the process finishes its execution, or it is terminated by the operation system, it is moved to the terminated state where it waits to be removed from main memory.

Q4. What is Inter Process Communication? Discuss the two fundamental models of Inter Process Communication.

A. A process can be of two type:

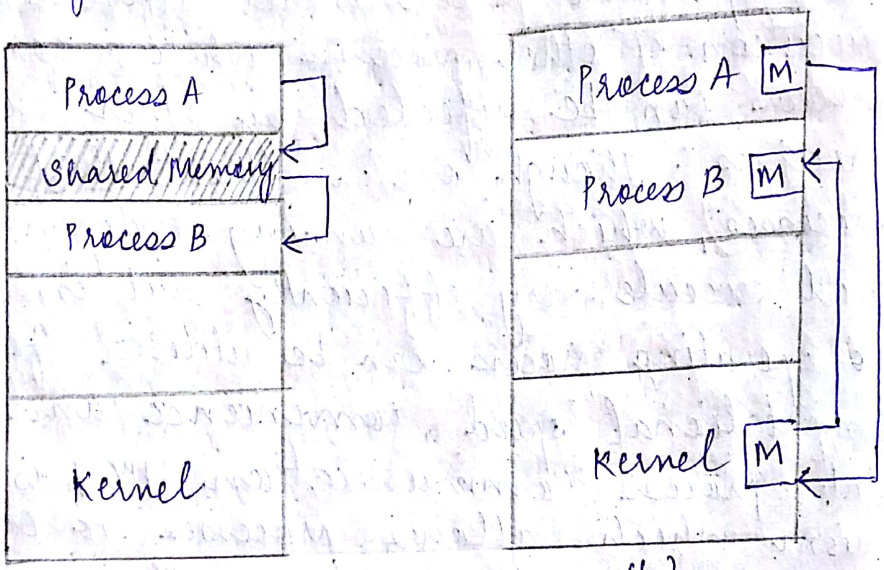
- Independent Process
- Co-operating Process

An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently but in practical, co-operating process can be utilised for increasing computational speed, convenience and modularity. Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions. The communication between these processes can be seen in two ways:

1. Shared Memory Model
2. Message Passing Model.

1. Shared Memory Model – It completely depends on how programmer will implement it. Suppose Process 1 and Process 2 are

executing simultaneously and they share some resources or use some information from other process, process 1 generate information about certain computations or resources ~~or use some~~ information from ~~other~~ process being used and keeps it as a record in shared memory. When process 2 need to use the shared information, it will check in the record stored in shared memory and take note of the generated information by process 1 to act accordingly. Processes can use shared memory for extracting information as a record from other process as well as for delivering any specific information to other process.



(a) Shared Memory
(b) Message Passing
Communication Models.

For example Producer - Consumer Problem, there are two processes P_1 and P_2 where P_1 is the producer and P_2 is the consumer. P_1 can only produce and P_2 can only consume. P_1 has to wait upon P_2, when the buffer is full. In every condition, consumer P_2 has to wait for the producer P_1 when the buffer is empty.

Buffer

```
#define Buffer_size
typedef struct
{
  - - -
  - - -
} item;
item buffer [Buffer_size];
int in = 0;
int out = 0;
```

Producer

```
item nextproduced;
while (true)
{
while (((in+1) % Buffer_size) == out);
buffer [in] = nextproduced;
  in = (in+1) % Buffer_size;
}.
```

Consumer

```
item nextconsumed;
while (true)
{
while (in == out);
nextconsumed = buffer [out];
out = (out+1) % Buffer_size;
/* consume the next item in nextconsumed
}
```

2. Message Passing Model - Mesage passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space and is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network. It provides at least two

operations : send (message) and receive (message)
Messages sent by a process can be of either fixed or variable size. There are three methods for implementing a link and the send() / receive() operations :

⊙ Direct or Indirect communication

Under direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication. In this scheme, the send() and receive() primitives are defined as :

- send (P, message) – send a message to process P
- receive (Q, message) – Receive a message from process Q.

A link is established automatically between every pair of processes that want to communicate. A link is associated with exactly two processes. Between each pair of processes, there exists exactly one link.

Under indirect communication, the messages are send to and received from mailboxes, or ports. A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed.

- send (A, message) – send a message to mailbox A.
- receive (A, message) – Receive a message from mailbox A.

A link is established between a pair of processes only if both members of the pair have a shared mailbox.

A link may be associated with more than two processes.

Between each pair of processes, there may be a number of different links, with each link corresponding to one mailbox.

④ Synchronous or Asynchronous communication
Message passing may be either blocking or nonblocking — also known as synchronous and asynchronous.

- Blocking send - The sending process is blocked until the message ~~and resumes~~ is received by the receiving process or by the mailbox.
- Non blocking send - The sending process sends the message and resumes operation.
- Blocking receive - The receiver blocks until a message is available.
- Non blocking receive - The receiver retrieves either a valid message or a null.

⑤ Buffering (Automatic and Explicit Buffering)
Whether communication is direct or indirect messages exchanged by communicating processes reside in a temporary queue which can be implemented by 3 ways:

- Zero capacity - It has a maximum length of zero, thus, the link cannot have any messages waiting in it.
- Bounded capacity - The queue has finite length n, thus, at most n messages can reside in it. If the queue is not full when a new message is sent, the message is placed in the queue.
- Unbounded capacity - The queue's length is potentially infinite, thus any number of messages can wait in it. The sender never blocks.

Bounded and Unbounded capacity cases are referred to as automatic buffering.