3...

Process Management

Objectives...

- To learn about concept of the Process.
- To get knowledge about Process Scheduling.
- To study about operations on Process such as Process Creation and Termination.
- To know about Interprocess Communication.

3.1 WHAT IS PROCESS? [S-22]

- A process is a program in execution. As the program executes the process changes state.
- The state of a process is defined by its current activity.
- Process execution is an alternating sequence of CPU and I/O bursts, beginning and ending with a CPU burst. Thus, each process may be in one of the following states: **New, Active, Waiting** or **Halted**.
- In Process model, the operating system is organized into a number of sequential processes, or just processes for short.
- A process is just an executing program, including the current values of the program counter, register and variables.
- Conceptually, each process had its own virtual CPU.
- In reality, of course, the real CPU switches back and forth from process to process; thus it is much a collection of processes running in parallel. This rapid switching back and forth is called Multiprogramming.
- In Fig. 3.1 (a), we see a computer multiprogramming four programs in the memory. In Fig. 3.1 (b), we can see how this has been abstracted into four processes, each with its own flow of control (i.e. its own program counter), and each one running independent of the other ones. In third Fig. 3.1 (c), we can see that viewed over a long enough time interval, all the processes have made progress, but at any given instant only one process is actually running.

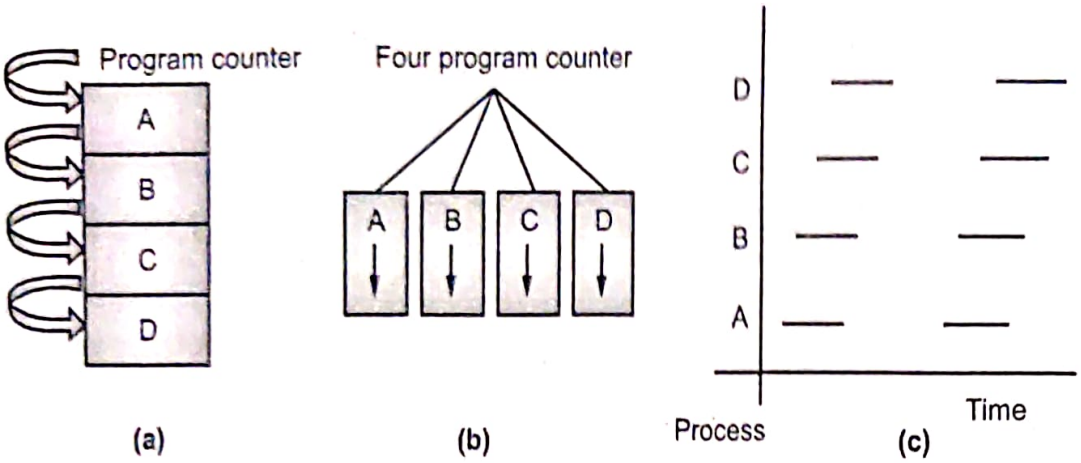

Fig. 3.1: The Process Model

- With the CPU switching back and forth among the processes, the rate at which a process performs its computation will not be uniform, and probably not even reproducible if the same processes are run again.
- Thus, processes are an activity of some kind. It has a program, input, output and a state a single processor may be shared among several processes, with some scheduling algorithm being used to determine when to stop work on one process and service a different one.

3.1.1 Process States [S-18]

- A process is a program in execution which includes the current activity and this state is represented by the program counter and the contents of the processor's register.
- There is a process stack for storage of temporary data.
- There may be a situation when two processes may be associated with the same program but they are considered for two separate execution processes.
- A user can have several programs running and all these programs may be of a similar nature but they must have different processes.
- As a process executes, it changes state. The state of a process is defined in part by the current activity of that process.
- We have two types of process state models which are :
  1. **Two State Model:** This represents a simple model by observing that a process is either being executed by a processor or not. It can be two states Running or Not running.

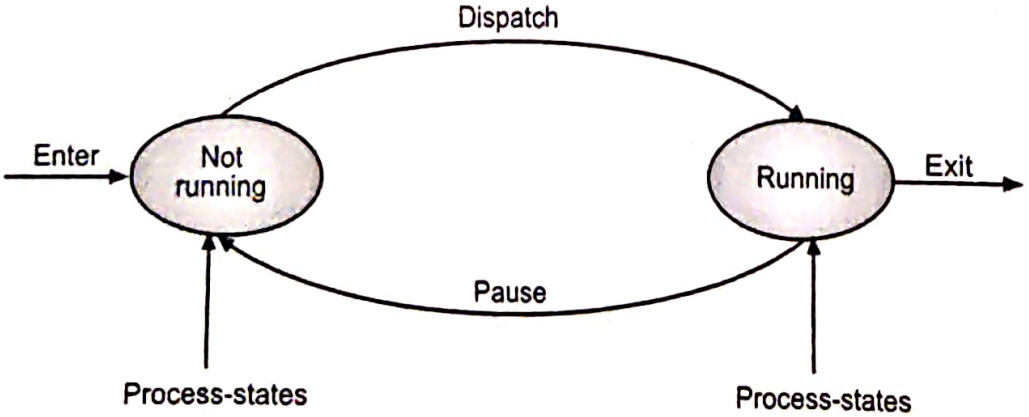

Fig. 3.2: State Transition diagram of Two-State Process Model

(a) **Running state:**

- When a new process is created, it enters into the system as in the running state.
- The process is waiting for an opportunity to operate, at times the currently running process will be interrupted and the dispatcher of the operating system will select a new process to run, one of the processes is now in the **running** state.

(b) **Not running State:** Processes that are not running must be kept in some queue, waiting their turn to be executed. Each entry in the queue is a pointer to a particular process.

2. **Five State Model :** In this, there are five stages and each process may be in one of the following states :

(a) **New** : The new process is created when a specific program calls from secondary memory/ hard disk to primary memory/ RAM

(b) **Running** : The process is being executed.

(c) **Waiting** : The process is waiting for some event to occur such as an I/O completion.

(d) **Ready** : The process should be loaded into the primary memory, which is ready for execution.

(e) **Terminated** : The process has finished execution.

- These names are arbitrary and they vary from operating systems to operating systems. The important thing is only one process can be running in any processor at any time. Many processes may be ready and waiting state.
- The state diagram corresponding to these states is shown in Fig. 3.3.

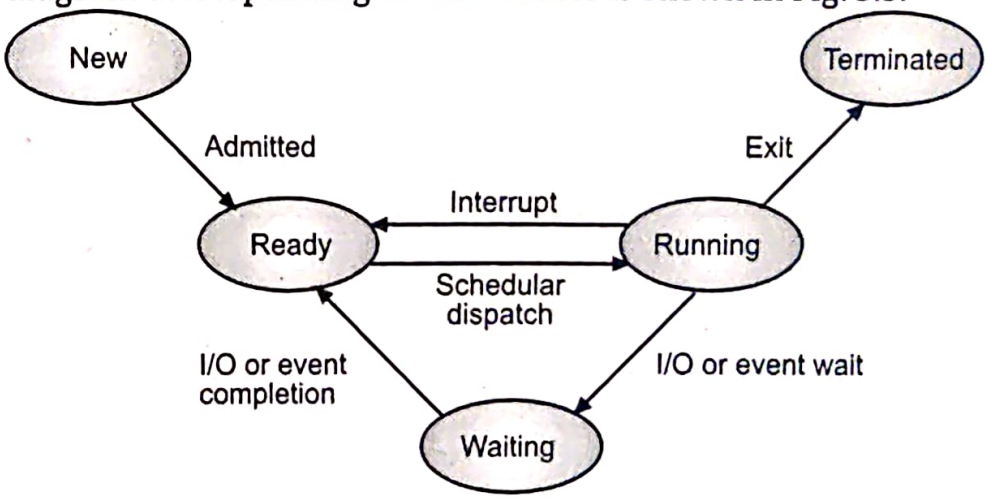

Fig. 3.3: Process States

- **New → Ready:** The operating system creates a process and prepares the process to be executed, and then the operating system moves the process into "Ready" queue.
- **Ready → Running:** When it is time to select a process to run, the operating system selects one of the jobs from the ready queue and moves the process from ready state to running state.
- **Running → Terminated:** When the execution of a process has completed, then the operating system terminates that process from running state. Sometimes, operating

system terminates the process some other reasons also include time limit exceeded, memory unavailable violation, protection error, I/O failure, data misuse and so on.

- **Running → Ready:** When the time slot of the processor expired or if processor received any interrupt signal, then the operating system shifted running process to ready state. For example, process P1 is executed by processor; in the meantime process P2 generates an interrupt signal to the processor. Then, the processor compares the priorities of process P1 and P2, if P1 > P2 then the processor continues the process P1 otherwise, the processor switched to process P2, and the process P1 moved to ready state.
- **Running → Waiting:** A process is put into waiting state, if the process need an event to occur or an I/O device requires. The operating system does not provide the I/O or event immediately then the process moved to waiting state by the operating system.
- **Waiting → Ready:** A process in the blocked state is moved to ready state when the event for which it has been waiting to occur. For example, a process is in running state need an I/O device, then the process moved to block or waiting state. When the I/O device provided by the operating system, the process moved to ready state from waiting or blocked state.

3.1.2 Process Control Block (PCB) [W-18, 22; S-19, 22]

- Each process is represented in the operating system by a **Process Control Block** (PCB) also called as **Task Control Block**.
- The operating system groups all information that it needs about a particular process into a data structure called a **PCB** or **Process Descriptor**.
- When a process is created, the operating system creates a corresponding PCB and releases whenever, the process terminates.
- The information stored in a PCB includes:
  - Process name (ID): Each process is given an integer identifier, termed as Process identifier, or PID.
  - Priority.

Structure of the Process Control Block:

- A PCB is shown in Fig. 3.4. It contains many data items associated with a specific process.

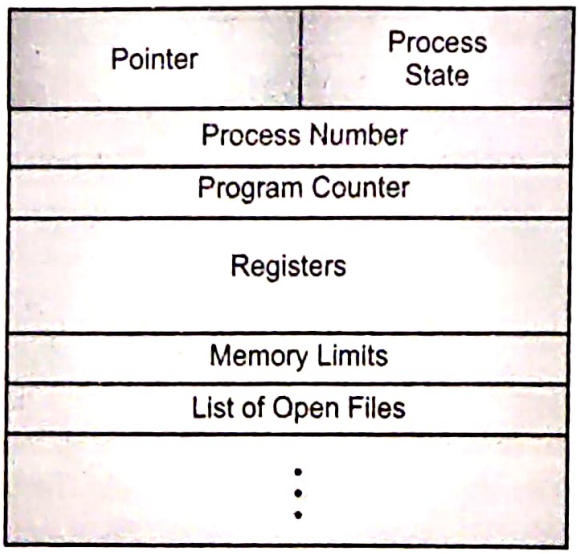

Fig. 3.4: Process Control Block (PCB)

- The following are the data items in PCB:
  - **Process State:** The state may be New, Ready, Running, and Waiting, Halted and so on.
  - **Program Counter:** The counter indicates the address of the next instruction to be executed for this process.
  - **CPU Registers:** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers and general purpose registers, plus any condition - code information.
  - **CPU Scheduling Information:** The information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
  - **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.
  - **Accounting Information :** This information includes the amount of CPU and real time used, time limits, account number, job or process numbers and so on.
  - **I/O Status Information:** The information includes the list of I/O devices allocated to this process, a list of open files and so on.

Example: The PCB in the Linux operating system contains C structure `task_struct`. It contains all necessary information about process like state of the process, its Scheduling and memory management related information, how many lists of open files, and pointers to the process's parent and children.

The structure has some of the following field members:

```
pid_t id;                    //process identifier
long state;                  //state of the process
long int time_slice          //scheduling information
struct task_struct *parent   //pointer to parent process
struct task_struct *parent   //pointer to child process
struct file_struct *files    // list of open files
struct mm_struct *mm;        //address space for the processed
```

Threads:

- Thread is the segment of a process. A process can have multiple threads.
- The process model discussed so far has implied that a process is a program that performs a single thread of execution of execution.
- For example, when a process is running a word-processor program, a single thread of instructions is being executed. This single thread of control allows the process to perform only one task at one time. For this reason, the user cannot simultaneously type in characters and run the spell checker within the same process.
- Number of modern operating systems have extended the process concept to allow a process to have multiple threads of execution and thus to perform more than one task at a time.
- On a system that supports threads, the PCB is expanded to include information for each thread. Other changes throughout the system are also needed to support threads.

3.2 PROCESS SCHEDULING [W-18, S-19]

- In operating system always a process should be executing to maximize CPU utilization. The main objective of time sharing system is to switch CPU between different processes such that user can interact with each program while running.
- The process scheduler selects the available process (from the set of available processes) for execution purpose. If there are other processes then they have to wait until the CPU is free and it can be rescheduled.

3.2.1 Scheduling Queues

- The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue.
- The Operating System maintains the following important process scheduling queues:
  1. **Job queue:** All processes when enters into the system are stored in the Job Queue.
  2. **Ready queue:** Processes in the Ready state are placed in the Ready Queue.
  3. **Device queues:** Processes waiting for a device to become available are places in device queues. There are unique device queues available for each I/O device.
- The most common representation of process scheduling is queueing diagram, in the following diagram the rectangular boxes represents queue, there are two types of queue ready queue, and set of device queue. The circle denotes resources that serves queue, arrow indicates flow of process in the system.
- When a new process is arrives it is kept in ready queue, it has to wait until it gets selected for execution, or is dispatched.
- When the process is selected for execution, one of following states can occur:
  - The process could issue an I/O request and then it is placed in I/O queue.
  - The process can create new sub process and waits for its subprocess termination.
  - The process can be removed forcefully, from the CPU, when the interrupt occurs and can be taken back in ready queue.

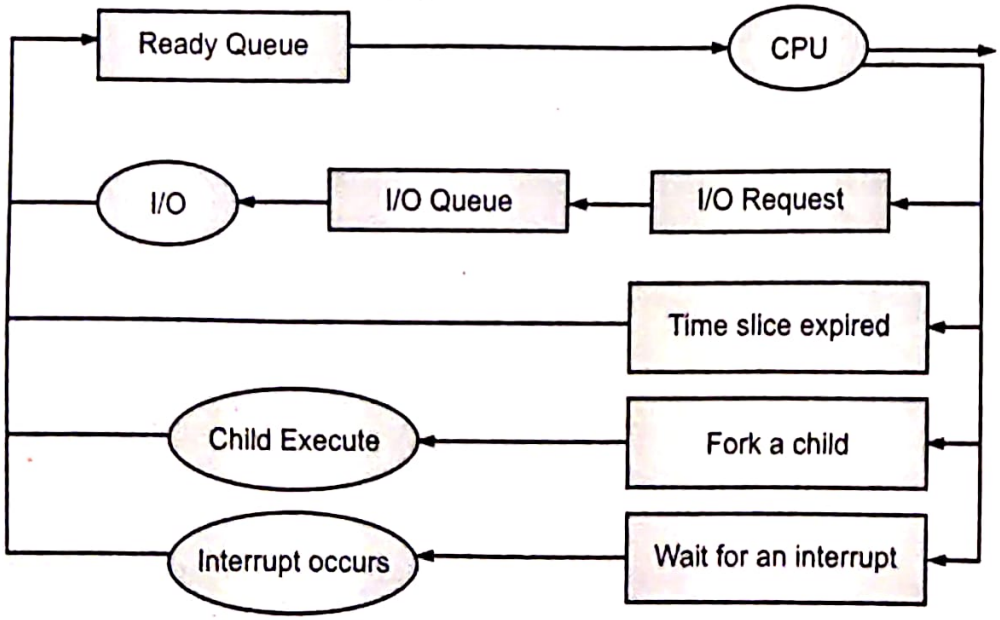

Fig. 3.5: Queuing diagram for Process Scheduling

3.2.2 Schedulers [S-18, 19; W-22]

- Schedulers are special system software which handles the process scheduling in various ways.
- Their main task is to select the jobs to be submitted into the system and to decide which process to run.
- Schedulers are of three types:
  1. Long-Term Scheduler
  2. Short-Term Scheduler
  3. Medium-Term Scheduler

1. Long-Term Scheduler [S-22]

- The Long term scheduler or job scheduler selects the process from the pool and loads it into the main memory.
- Long term scheduler runs less frequently. It decides which program must get into the job queue. From the job queue, job processor selects process and loads into memory for execution. The main objective of scheduler is to maintain the degree of multiprogramming.

2. Short-Term Scheduler

- The Short Term Scheduler or CPU Scheduler selects among the processes that are ready to execute and allocate CPU to one of them. Short term scheduler runs very frequently.
- Primary aim of this is to enhance CPU performance and increase process execution rate.

3. Medium-Term Scheduler [S-22; W-22]

- Medium term scheduler takes care of swapped out processes. If the running process needs some I/O time for completion then it needs to change its state from running to waiting.
- Medium term scheduler is used for this purpose. It removes the running process and makes room for new process. Such processes are called swapped out processes and this procedure is called as Swapping.
- The medium term scheduler is used for suspending and resuming the process.

3.2.3 Context Switch [S-23; W-18]

- It is a process that involves switching of the CPU from one process to another. In this, execution of the process that is present in the running state is suspended by the kernel and another process that is present in ready state is executed by the CPU.
- Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a Context Switch.
- Context switch can happen due to the following reasons:
  1. When high priority process comes in the ready state, in this case running process is stopped and higher priority process should be given to CPU for execution.
  2. When an interrupt occurs then running process should be stopped and CPU should handle the interrupt which is occurred.
  3. When transaction between user mode and kernel mode is required then context switch is performed.

- While performing context switch following steps occurred:
  1. Save the context of the process that is currently running by the CPU, update the PCB and other important details.
  2. Move the PCB of above process into ready queue, I/O queue etc.
  3. Select new process for execution.
  4. Update the PCB of the selected process. It includes updating the process state to running.
  5. Update memory management data structure as required.
  6. Restore the context of the process that was previously running when it is loaded again on the processor.
- Context switch is used for multitasking that is multi programming with time sharing, here the context switch occurs so fast that user feels that the CPU is executing more than one task at same time.
- Context-switch time is pure overhead, because the system does no useful work while switching. Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions.
- Context-switch times are highly dependent on hardware support. For instance, some processors provide multiple sets of registers.
- A context switch simply includes changing the pointers to the current register set. Of course, if active processes exceed register sets, the system resorts to copying the register data to and from memory.
- Also, the more complex the operating system, the more work must be done during a context switch.

3.3 OPERATIONS ON PROCESSES

- The processes in the system can execute concurrently and they must be created and deleted dynamically.
- Operations on processes are Process Creation and Termination.

3.3.1 Process Creation

- When a new process is to be added to those currently being managed, the operating system builds the data structures that are used to manage the process, and allocates address space in main memory to the process. This is the creation of a new process.

Common events lead to a Process Creation:

- In a batch environment, a process is created in response to the submission of a job. In an interactive environment, a process is created when a new user attempts to log on. In both cases, the operating system is responsible for the creation of the new process.
- When operating system decides to create a new process, it can proceed as follows:
  1. Assign a unique identifier to the new process.
  2. Allocate space for the process.
  3. Initialize the process control block.
  4. Set the appropriate linkages.
  5. Create or expand other data structures.

- Operating system created all processes in a way that was transparent to the user or application program, and this is still commonly found with many contemporary operating systems. When the operating system creates a process at the explicit request of another process, the action is referred to as process spawning.
- When one process spawns another, the former is referred to as the parent process, and the spawned process is referred to as the child process. The parent may have to partition its resources among its children, or it may be able to share some resources among several of its children.
- A sub-process may be able to obtain its resources directly from the operating system. When a process creates a new process, two possibilities exist in terms of execution :
  - The parent continues to execute concurrently with its children.
  - The parent waits until some or all of its children have terminated.

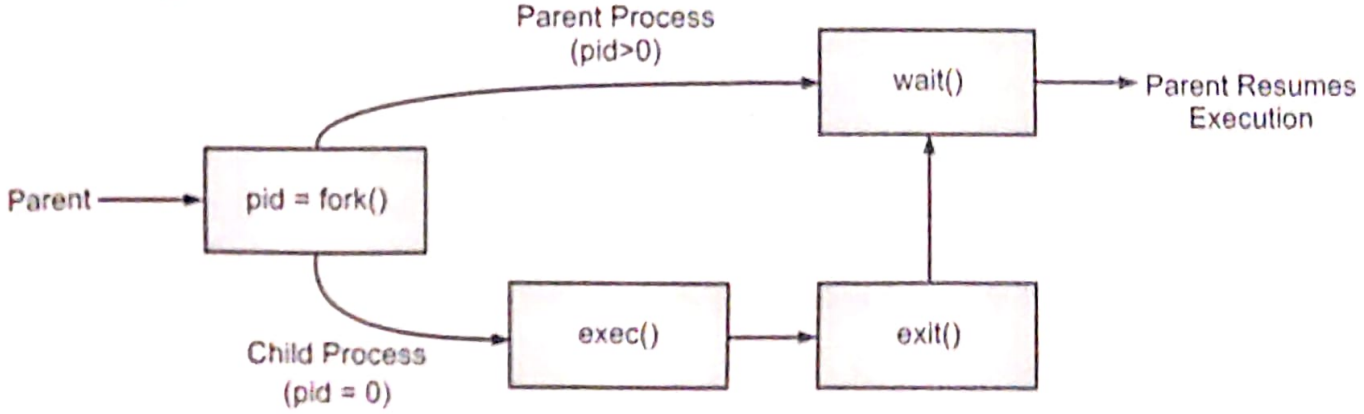

Fig. 3.6: Process Creation using fork()

Example of Process Creation on UNIX system:

- A process creates a child process in UNIX through a system call *fork*.
- fork system call creates a child process and sets up its execution environment, then it allocates an entry in the process table i.e. a PCB for the newly created process and marks its state as ready.
- Fork also returns the id of the child process to its creator also called the parent process.
- The child process shares the address space and file pointers of the parent process, hence data and files can be directly shared.
- A child process can in turn create its own child processes, thus leading to the creation of a process tree.
- The UNIX operating system keeps track of the parent-child relationships throughout the lives of the parent and child processes.
- The child process in UNIX environment called its context is a copy of the parent's environment. Hence, the child executes the same code as the parent. At creation, the program counter of the child process is set at the instruction at which the fork call returns.
- The only difference between the parent and the child processes is that in the parent process fork returns with the process id of the child process, while in the child process it returns with a '0'.

3.3.2 Process Termination

- When processes terminate, it returns data to its parent process. Resources like memory, files and I/O are de-allocated by the operating system. A child process may be terminated if its parent process requests for its termination.
- Process will terminate usually following reasons:
  1. **Due to Normal exit:** When compiler has compiled the program, the compiler executes a system call to inform the operating system that task has finished. This call is exit in UNIX and Exit Process in windows.
  2. **Error Exit:** Another reason for termination is that process finds fatal error, suppose user types the command *sample.c* and no such files present, the compiler simply exits.
  3. **Fatal Error:** The third reason for termination is an error caused by the process, often due to a program bug. Examples include executing an illegal instruction, referencing nonexistent memory, or dividing by zero. In some systems (e.g., UNIX), a process can tell the operating system that it wishes to handle certain errors itself, in which case the process is signaled (interrupted) instead of terminated when one of the errors occurs.
  4. **Killed by another Process:** The fourth reason a process might terminate is that the process executes a system call telling the operating system to kill some other process. In UNIX this call is kill. The corresponding Win32 function is *Terminate Process*.

Example:

- Any process p_i can terminate itself through the exit system call,

```
exit (status_code);
```

  Where, the value of `status_code` is saved in the kernel for access by the parent of p_i. If the parent is waiting for the termination of p_i, a signal is sent to it. The child processes of p_i are made the children of a kernel process.
- Waiting for process termination : A process p_i can wait for the completion of a child process through the system call,

```
wait(add(abc));
```

  Where `abc` is a variable. When a child of p_i terminates the wait call returns after storing the termination status of the terminated child process into `abc`. The wait call returns with '–1' if p_i has no children.

```
main ()
{
int saved_status;
for (i = 0; i < 3; i++)
{
   if fork()== 0)
   {
      /* code for child processes */
         exit ();
   }
}
while(wait(&saved_status) != -1)
      /* All child processes terminated? */
}
```

3.4 INTERPROCESS COMMUNICATION

- There are independent processes and co-operating processes in the operating system.
  1. **Independent Process:** The processes which are independent that can not affect or be affected by other processes executed in system. Or we can say that the process which does not share data with other process is independent process.
  2. **Co-operating Process:** Co-operating process shares data with other process and its execution can get affected by other process.
- There are many reasons for providing environment that allows process cooperation.
  1. **Information Sharing:** Shared data can be required by several processes. Like shared file can be used by two or more processes, so we must allow concurrent access to such information.
  2. **Computation speedup:** To execute task faster we should break up into subtasks, so each subtask will execute parallel with other. This can be achieved only if computer has multiple processing elements.
  3. **Modularity:** Dividing the system functions into separate processes or threads.
  4. **Convenience:** Individual user may work on multiple tasks at a same time. Like, user may be editing, compiling, printing in parallel.

3.4.1 Shared Memory System

- In the shared memory system, the cooperating processes communicate with each other by establishing the shared memory region, in its address space.
- It is the fundamental model of interprocess communication. It allows fastest interprocess communication.

Working:

- In Shared Memory system, the cooperating processes communicate, to exchange the data or the information with each other. For this, the cooperating processes establish a shared region in their memory. The processes share data by reading and writing the data in the shared segment of the processes.
- Consider a situation that there are two cooperating processes P1 and P2. Both the processes have their different address space. Now P1 wants to share some data with P2. So both processes will have to perform following steps.

  Step 1 : As Process P1 has some data to share with P2. Process P1 has to take initiative and establish a shared memory region in its address space and store its data to be shared in its shared memory region.

  Step 2 : Now, P2 requires information which is in shared segment of P1. So P2 has to attach itself to shared address space of P1. Now, Process P2 can read out the data from there.

  Step 3 : Now, Processes P2 and P1 can exchange information by reading and writing data in shared segment of the process.

3.4.2 Message Passing Systems

- Operating system provides the means for cooperating processes to communicate with each other via message passing facility.
- In this method, processes communicate with each other without using any kind of shared memory.
- This communication could involve a process letting another process to know that some event has occurred or transferring of data from one process to another. This communication model is message passing model.
- This model allows multiple processes to read and write data to message queue without connected to each other. Messages are stored in the queue until their recipient retrieves them. Message queues are very useful for Interprocess Communication.

Working:

- Following diagram demonstrates the Message passing model of the process communication.

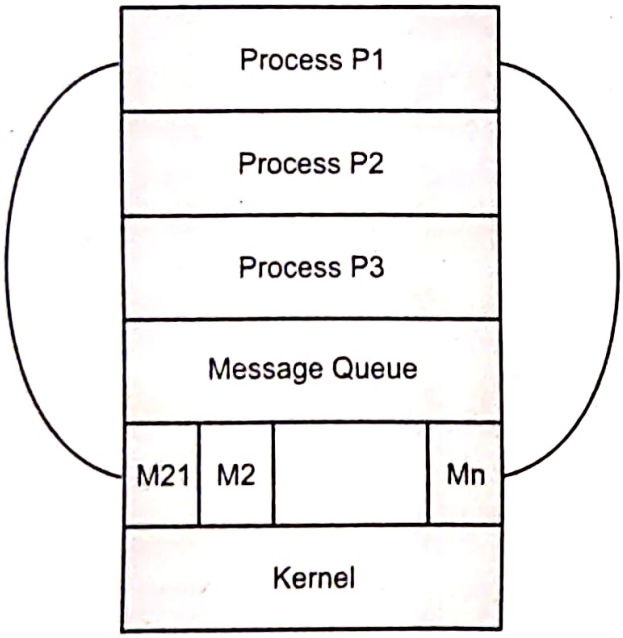

Fig. 3.7: Message Passing Model

- Message passing facility provides at least two operations: send (message) and receive (message).
- The messages send by the process can be fixed size or variable size. If fixed size messages are sent, the system level implementation is straight forward.
- Variable size messages require more complex system level implementation, but the programming task becomes simple.
- A standard message can have two parts: header and body.
- Header part is used for storing Message Type, Destination Id, Source Id, Length of Message, and Control Information.
- If the Processes P and Q want to communicate, they must send and receive messages from each other; communication link must exist between them.
- There are several methods for logically implementing a link and send() and receive() operations.

Methods of implementing communication link:

- A link has some capacity that determines the number of messages that can reside in it temporarily for which every link has a queue associated with it which can be of zero capacity, bounded capacity, or unbounded capacity.
- In zero capacity, the sender waits until the receiver informs the sender that it has received the message.
- In non-zero capacity cases, a process does not know whether a message has been received or not after the send operation. For this, the sender must communicate with the receiver explicitly. Implementation of the link depends on the situation; it can be either a direct communication link or an in-directed communication link.

1. **Direct and Indirect Communication:**

- Direct Communication links are implemented when the processes use a specific process identifier for the communication, but it is hard to identify the sender ahead of time. For example: the print server.
- In-direct Communication is done via a shared mailbox (port), which consists of a queue of messages. The sender keeps the message in mailbox and the receiver picks them up.

2. **Synchronous and Asynchronous Communication:**

- Communication between processes take place through calls to send() and receive() primitives. There are different design options for implementing each primitive message passing may be either Blocking or Non-blocking also called as Synchronous and Asynchronous.
  - **Synchronous:** A synchronous operation blocks process till the operation completes.
  - **Asynchronous:** This operation is non-blocking and only initiates the operation. The caller could discover completion by some other mechanism notion of synchronous operations requires an understanding of what it means for an operation to complete.

Summary

- Process is instance of computer program that is being executed by one or more threads. Process has its different states like, New, Ready, Running, Waiting and Terminated.
- **Process Models:** Two state model and five state model.
- Process Control Block is a data structure that contains information of the process related to it. The process control block is also known as a task control block, entry of the process table.
- Entries in the PCB are Process state, program counter, CPU Registers, CPU – Scheduling Information, Memory Management Information, Accounting information, I/O Status information.
- We have to schedule processes to CPU for multiprogramming purpose. The main objective is to increase degree of multiprogramming. Schedulers play this role to schedule processes. Short Term Scheduler, Medium Term Scheduler and Long Term Scheduler are used for this purpose.

- Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a Context Switch.
- There are different operations performed on processes like Creation of process and Termination of process.
- Interprocess communication is a set of programming interface which allow a programmer to coordinate activities among various program processes which can run concurrently in an operating system. This allows a specific program to handle many user requests at the same time.
- Interprocess communication (IPC) is performed by Shared Memory Systems and Message Passing Systems.
- Message passing is a mechanism for a process to communicate and synchronize.
- In Shared Memory System, a particular region of memory is shared between cooperating processes.

Check Your Understanding

1. A process is more than program code, which is sometimes known as _____.
   (a) text section (b) contents of processor registers
   (c) stack (d) data section
2. When process gets executes it changes its state, the process may be in one of the following states.
   1. New 2. running
   3. waiting 4. ready
   5. terminated
   (a) 1, 2 (b) 1,2,4
   (c) 1,2,3 (d) 1,2,3,4,5
3. Which of the following stage defines "the process is waiting for some other event to occur"?
   (a) ready (b) running
   (c) waiting (d) terminated
4. In operating systems each process is represented by_____.
   (a) print dialog box (b) process control block
   (c) program control box (d) process command box
5. The address of next instruction to be executed for the current process is stored in _____.
   (a) CPU registers (b) program registers
   (c) stack pointers (d) program counter
6. The objective of multiprogramming is _____.
   1. to have running some processes all the time
   2. to execute single process at a time
   3. to minimize the CPU utilization
   4. to maximize the CPU utilization
   (a) 1,2 (b) 1,3
   (c) 1,4 (d) none of above

7. The process which is ready to execute and residing in main memory is kept in _____.
   (a) device queue (b) processor queue
   (c) ready queue (d) job queue
8. Which among following is true for short term scheduler?
   1. Select from among the processes that are ready to execute and allocate CPU to them.
   2. Select process from mass storage device and loads them into memory for execution.
   3. Short term scheduler must select new process for CPU.
   4. It executes less frequently.
   (a) 1 only (b) 2 and 3 only
   (c) 1, 2, 4 only (d) 1, 3
9. Copying process from memory to disk to allow space for other process is called _____.
   (a) deadlock (b) swapping
   (c) shifting (d) copying
10. PCB does not contain following _____.
   (a) code (b) stack
   (c) data (d) bootstrap program
11. Which of the following is used as synchronization tool?
   (a) thread (b) pipe
   (c) semaphore (d) socket
12. Which of the following is not a scheduler type?
   (a) short term scheduler (b) medium term scheduler
   (c) block scheduler (d) long term scheduler

Answers

1. (a)	2. (d)	3. (c)	4. (b)	5. (d)	6. (c)	7. (c)	8. (d)	9. (b)	10. (d)
				11. (c)	12. (c)				

Practice Questions

Q.I Answer the following questions in short:

1. Define the process?
2. What is interprocess communication?
3. What is swap in and swap out?
4. Define scheduler.
5. List types of interprocess communication.
6. What is thread in Process Management?

Q.II Answer the following questions:

1. What is Process? Explain with its state?
2. What is process control block?
3. Explain the term process concept in detail.
4. Explain context switch between the processes.
5. Explain the process creation and process termination
6. What is scheduler? Explain scheduling queue?
7. Explain shared memory system concept?
8. Explain message passing systems?

Q.III Define terms:

1. Shared memory
2. Message passing
3. Scheduling queues
4. Process control block
5. Context switch

Previous Exam Questions

Summer 2018

1. Explain process states in detail. **[4 M]**

Ans. Refer to Section 3.1.2.

2. Explain medium term scheduler. **[4 M]**

Ans. Refer to Section 3.2.2

Winter 2018

1. What is Context switch? **[2 M]**

Ans. Refer to Section 3.2.3.

2. Explain Process Control Block (PCB) in detail with the help of diagram. **[4 M]**

Ans. Refer to Section 3.1.3.

3. Describe process state with suitable diagram **[4 M]**

Ans. Refer to Section 3.1.2.

4. Explain medium term scheduler in detail. **[4 M]**

Ans. Refer to Section 3.2.

Summer 2019

1. Which scheduler controls the degree of multiprogramming? How? **[2 M]**

Ans. Refer to Section 3.2.

2. Explain the process Control Block with a diagram. **[4 M]**

Ans. Refer to Section 3.1.3.

3. Write a short note on medium-term scheduler. **[4 M]**

Ans. Refer to Section 3.2.2.

■■■