PROCESS

The notion of *process* is central to the understanding of today's multi-user computer systems. Several of the more popular definitions are presented, but no perfect definition of process has as yet appeared in literature.
The definitions and concepts introduced here serve as a basis for the discussions of concurrent processes and process scheduling.

DEFINITIONS OF PROCESS

The designers of the Multics system first used the term process in the 1960s. Since that time, process, used somewhat interchangeably with task, has been given many definitions. Some of these are as follows:

1. A program in execution
2. An asynchronous activity
3. The "locus of control" of a procedure in execution
4. That which is manifested by the existence of a "process control block" in the operating system.
5. That entity to which processors are assigned
6. The "dispatchable" unit

Many other definitions have been given. There is no universally agreed upon definition but the "program in execution" concept seems to be most frequently referenced.

PROCESS BEHAVIOR

The first step in designing an operating system to control processes is to describe the behavior that the user would like the processes to exhibit.

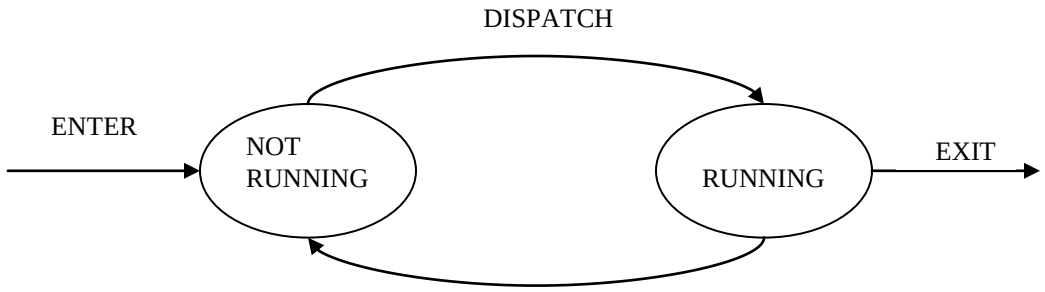
The simplest possible model can be constructed by observing that, at any time, a process is either being executed by a processor or not. Thus, a process may be in one of two states: *Running* and *Not-running*.

When the operating system creates a new process, it enters that process into the system in the *not-running* state. Thus, the process exists, is known to the operating system, and is waiting for an opportunity to execute. From time to time, current process will be interrupted and the dispatcher portion of the operating system will select a new process to run. The former process moves from the running state to the not-running state, and one of the other processes moves to the running state.

Each process must be represented in some way so that the OS can keep track of it. That is, there must be some information relating to each process, including current state and location in memory. Those processes that are not running need to be kept in some sort of queue, waiting for their turn to execute. The 2nd figure below suggests a structure. There is a single queue of processes. Each entry in the queue is a Pointer to a particular process; the queue consists of linked list of data blocks in which each block represents one process.

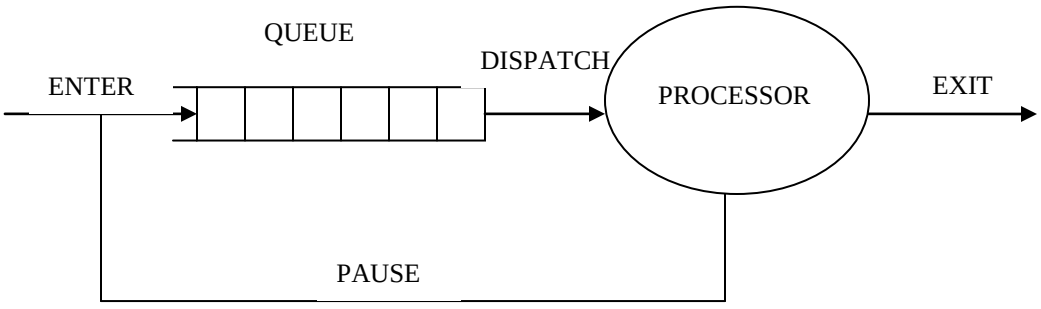
The behavior of the Dispatcher can be described in terms of the queuing diagram. When a process is interrupted, it is transferred to the queue of waiting processes, if the process has completed or aborted, the process is discarded (exits the system). In either case, the dispatcher then selects a process from the queue to execute.

PROCESS STATES



STATE TRANSITION

TWO STATE PROCESS MODEL



QUEUEING DIAGRAM

PROCESS CREATION

NEW BATCH JOB:

The operating system is provided with a batch Job control stream, when O/S is ready to take on new work it will read the next sequence of job control commands.

INTERACTIVE LOGON:

A user at a terminal logs on to the system.

CREATED BY OS TO PROVIDE A SEVICE:

The OS can create a process to perform a function on behalf of a user program, without the user having to wait (Printing).

SPAWNED BY EXISTING PROCESS:

For purposes of modularity or to exploit parallelism a user program can dictate the creation of a number of processes.

PROCESS TERMINATION

1. **NORMAL COMPLETION*:*** The process executes an OS service call to indicate that it has completed running.

2. **TIME LIMIT EXCEDED*:*** The process has run longer than the specified total time.

3. **MEMORY UNAVAILABLE*:*** The process requires more memory than the system can provided.

4. **BOUNDS VIOLATION*:*** The process tries to use M.L that it is not allowed to access.

5. **PROTECTION ERROR*:*** The process attempts to use a resource or a file that it is not allowed to use. Such as prohibited computation, such as writing to a read-only file.

6. **ARITHMETIC ERROR**: The process tries a prohibited computation such as divide by zero or store number larger than the hardware can accommodate.

7. **TIME OVERRUN*:*** The process has waited longer than a specified maximum for a certain event to occur.

8. **I/O FAILURE*:*** An error in I/O
Example: inability to find a file.
Hard-disk error
Failure to read or write after a specified maximum number of tries (when a defective area is encountered on a tape, or invalid operation such as reading from printer).

9. **INVALID INSTRUCTION*:*** The process attempts to execute a non-existent instruction (often a result of branching into a data area and attempting to execute the data)

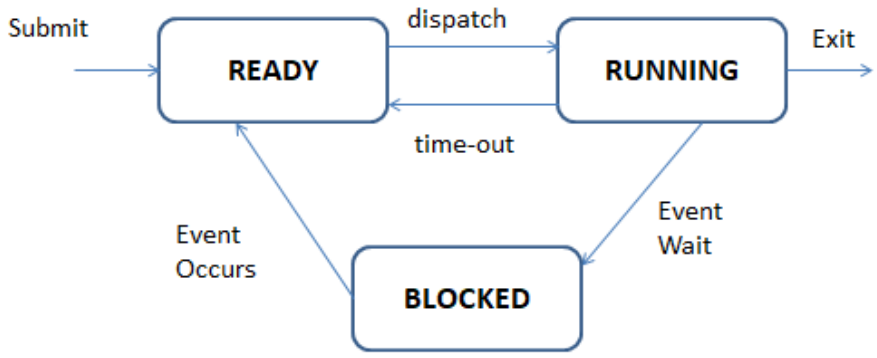
10. **PRIVILEGED INSTRUCTION*:*** The process attempt to use an instruction reserved for the operating system.

11. **DATA MISUSED*:*** A piece of data is not initialized.

12. **OPERATOR OR OS INTERVENTION*:*** The OS can terminate a process if a deadlock exists.

13. **PARENT TERMINATION*:*** When a parent terminates OS automatically terminates all the off springs or a parent process has the authority to terminate any of its off spring.

PROCESS STATE TRANSITION



THREE STATE PROCESS MODEL

Referring to the two state model, the not-running state can be split into two states: Ready and Blocked. The dispatcher would have to scan the list looking for the process that is not blocked and that has been in the queue the longest. The queue (Ready) is a first-in, first-out (FIFO) list, and the processor operates in round-robin fashion on the available processes. Each process in the queue is given a certain amount of time to execute and then returned to the queue unless blocked.

RUNNING: The process that is currently being executed. At most one process can be in this state at a time.
READY: Processes that are prepared to execute when given the opportunity.
BLOCKED: A process that cannot execute until some event occurs, such as the completion of an I/O operation.

QUEUING MODEL

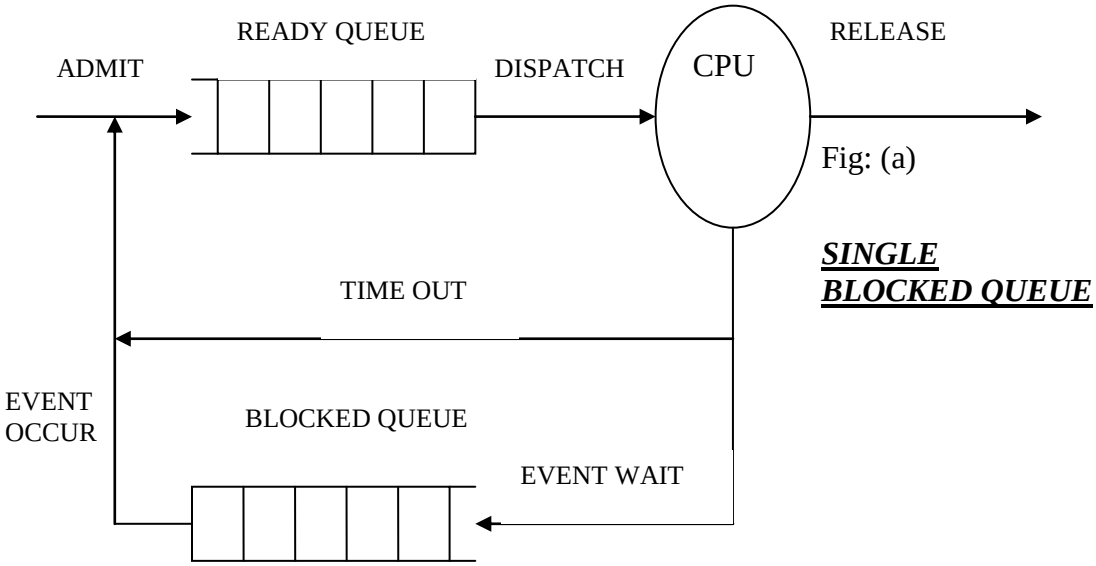


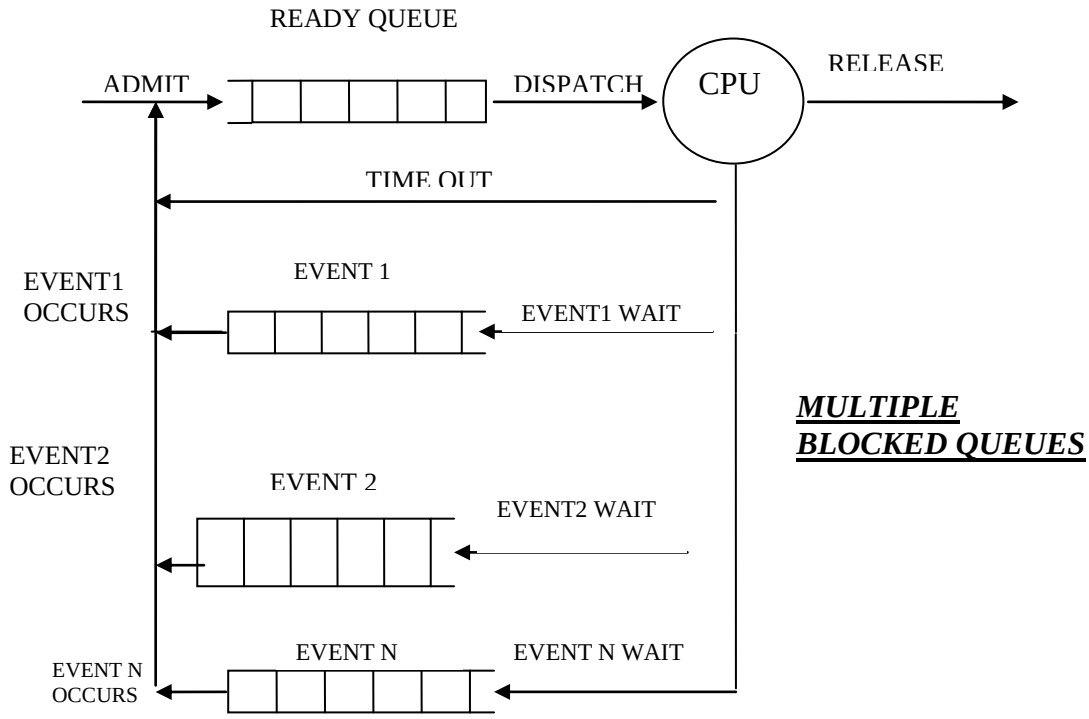
Fig: (a)

SINGLE BLOCKED QUEUE

The above figure suggests the way in which the queuing decipline might be implemented. There are now two queues: a Ready queue and a Blocked queue. As each process is admitted to the system, it is placed in the Ready queue. When it is time for OS to choose another process to run, it selects one from the Ready queue. When a running process is removed from execution, it is either terminated or placed in the Ready or Blocked queue, depending on the circumtances. Finally when an event occurs, all the processes in the Blocked queue that are waiting on that event are moved to the Ready queue.

MULTIPLE BLOCK QUEUES

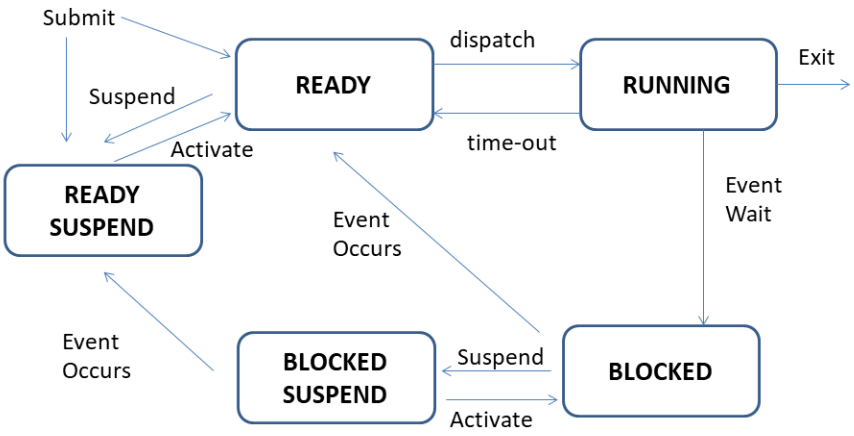
The activity just discussed means that, when an event occurs, the operating system must scan the entire Blocked queue, searching for those processes waiting on that event. In a large operating system, there could be hundreds or even thousands of processes in that queue. Therefore, it would be more efficient to have number of queues, one for each event. Then, when the event occurs one of the Blocked process from the appropriate queue is moved to the Ready state.



MULTIPLE BLOCKED QUEUES

PROCESS STATE TRANSITION

FIVE STATE PROCESS MODEL



The three states that have been described provide a systematic way of modeling the behavior of processes. Many operating systems are constructed using only these three states.

There is a good justification for adding additional states to the model. To see the benefit of these new states, consider a system that does not employ **virtual memory**.

Each process to be executed must be loaded fully into main memory. Thus (in the models discussed in the previous figures) all the processes in all the queues must reside in main memory.

It is very clear that I/O activities are much slower than computation. Thus, even with multiprogramming, a processor could be idle most of the time, waiting for processes to be available (Ready) for execution.

Looking at the degree of multiprogramming where hundreds of processes may be in the Ready state, and hundreds of them may be in the Blocked state. This is the point in time where the system becomes over loaded as far as the resource allocation is concerned. It becomes difficult for the operating system to allow all the process (Ready and/or Blocked) to monopolize the system resources and pay the heavy price in shape of a system deadlock.

What to do? Main memory could be expanded; the appetite of programs for memory has grown as fast as the cost of memory has dropped. So, larger memory results in larger processes, not more processes.

Solution: SWAPPING.

Swapping involves moving part or all of a process from main memory to disk.

Swapping, is an I/O operation, and there is the potential for making the problem worse, not better. But because disk I/O is generally the fastest I/O on a system (e.g., compared to tape or printer I/O), swapping will usually enhance performance.

With the use of swapping, one additional state must be added to our process behavior model, the **Suspend state**. When all the processes in main memory are in the Blocked state, the operating system swaps one of the blocked processes out onto disk into a Suspend queue. This is a queue of existing processes that have been temporarily kicked out of main memory, or suspended. The operating system then brings in another process from the Suspend queue, or it honors a new-process request. Execution then continues with the newly arrived process.

When the operating system has performed a swapping-out operation, it has two choices for selecting a process to bring into main memory: It can admit a newly created process, or can bring in a suspended process. It would appear that the preference should be to bring in a previously suspended process to provide it with service rather than increasing the total load on the system.

All the processes that have been suspended were in the Blocked state at the time of suspension. It would not do any good to bring a Blocked process back into main memory, since it is still not ready for execution (each process in the Suspend state was originally blocked on a particular event). When its event occurs, the process is not blocked and is potentially available for execution.

Therefore, we need to rethink this aspect of the design. There are two independent concepts here: whether a process is waiting on an event (blocked or not), and whether a process has been swapped out of main memory (suspended or not). To accommodate this two-by-two combination, we need the following states:

Ready: The process is in main memory and available for execution.
Blocked: The process is in main memory and awaiting an event.

Blocked suspend: The process is in secondary memory and awaiting an event.
Ready suspend: The process is in secondary memory but is available for execution as soon as it is loaded into main memory.

The conclusion is that in the interest of performance and efficiency, the operating system may suspend and swap few of the processes from both the active states that are the Ready state and Blocked state.

REASONS FOR PROCESS SUSPENSION

1. ***SWAPPING:*** The OS needs to release sufficient main memory to bring in a process that is ready to execute.

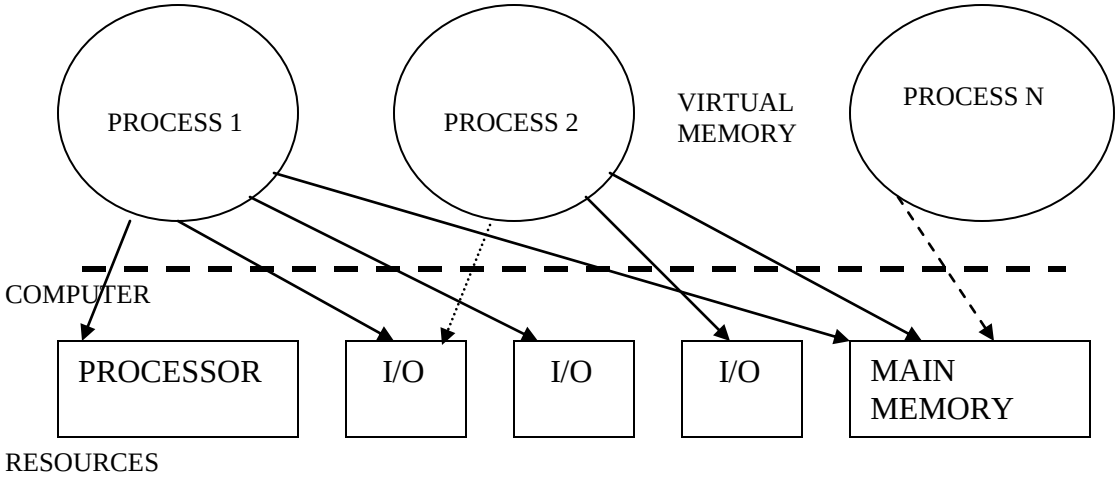
2. ***OTHER OS REASON:*** The OS may wish to suspend a background or utility process or a process that is suspected of causing a problem

3. ***ITERACTIVE USER REQUEST:*** A user may wish to suspend execution of a program for purpose of debugging or in connection with the use of a resource.

4. ***TIMING:*** A process may be executed periodically and may be suspended while waiting for the next time interval.

5. ***PARENT PROCESS REQUEST:*** A parent process may wish to suspend execution of a descendant to examine or modify the suspended process or to coordinate the activity of various descendants.

PROCESS DESCRIPTION



Process and Resources

The operating system is the controller of events. It schedules, dispatches processes for execution by the processor, that allocates resources to processes, and that responds to requests by user programs for basic services. The operating system is the unit that manages the use of system resources by processes.

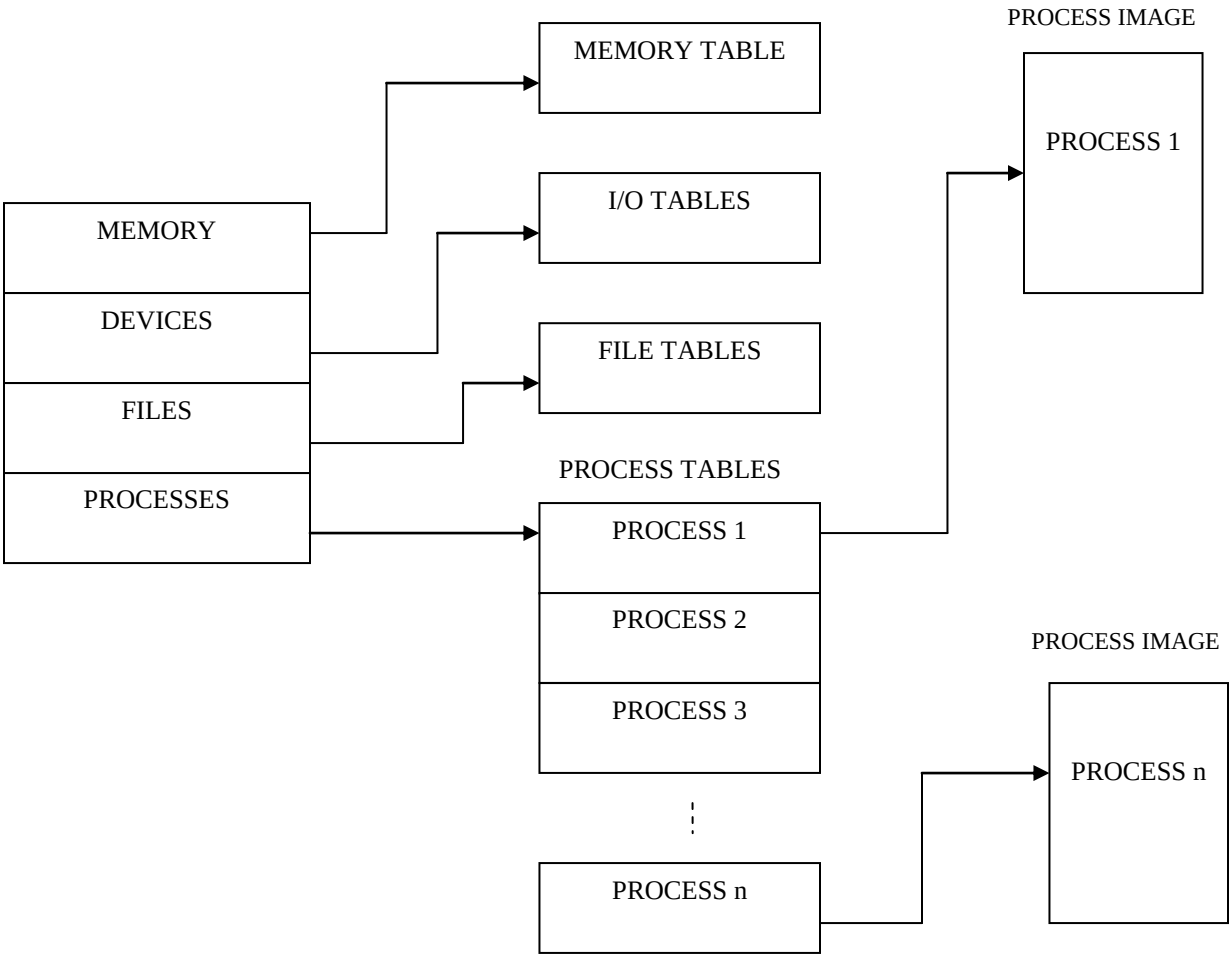
This concept is shown in the above figure. In a multiprogramming environment, there are a number of processes (P_1, P_2, ..., P_N) that have been created and exist in virtual memory. During the course of its execution, each process needs to have access to certain system resources, including the processor, I/O devices, and main memory. In the figure, process P_1 is **Running**; at least part of the process is in main memory, and it has control of two I/O devices. Process P_2 is also in main memory, but it is **Blocked,** waiting for an I/O device allocated to P_l. Process P_N has been swapped out and is therefore suspended

What does the operating system need to be able to control processes and manage resources for them?

Operating-System Control Structures

If the operating system is to manage processes and resources, then it must have information about the current status of each process and resource. The approach to providing this information is straightforward: The operating system constructs and maintains tables of information about each entity that it is managing. A general idea of the scope of this effort is indicated in the figure below, which shows four different types of tables maintained by the operating system: **memory**, **I/O (device)**, **file**, and **process**. Although the details will differ from one operating system to another, fundamentally, all operating systems maintain information in these four categories.

STRUCTURES OF CONTROL TABLES



Memory tables are used to keep track of both main (real) and secondary (virtual) memory. Some of main memory is reserved for use by the operating system; the remainder is available for use by processes. Processes are maintained on secondary memory by using some sort of virtual memory or simple swapping mechanism. The memory tables must include the following information:

- The allocation of main memory to processes
- The allocation of secondary memory to processes
- Any protection attributes of segments of main or virtual memory, such as which processes may access certain shared memory regions
- Any information needed to manage virtual memory

I/O tables are used by the operating system to manage the I/O devices and channels of the computer system. At any given time, an I/O device may be available or assigned to a particular process. If an I/O operation is in progress, the operating system needs to know the status of the I/O operation and the location in main memory being used as the source or destination of the I/O transfer.

The operating system may also maintain **file tables**, which provide information about the existence of files, their location on secondary memory, their current status, and other attributes. Much, if not all, of this information may be maintained and used by a *file-management system*, in which case the operating system has little or no knowledge of files. In other operating systems, much of the detail of file management is managed by the operating system itself.

Finally, the operating system must maintain **process tables** to manage processes.

ELEMENT OF A PROCESS IMAGE

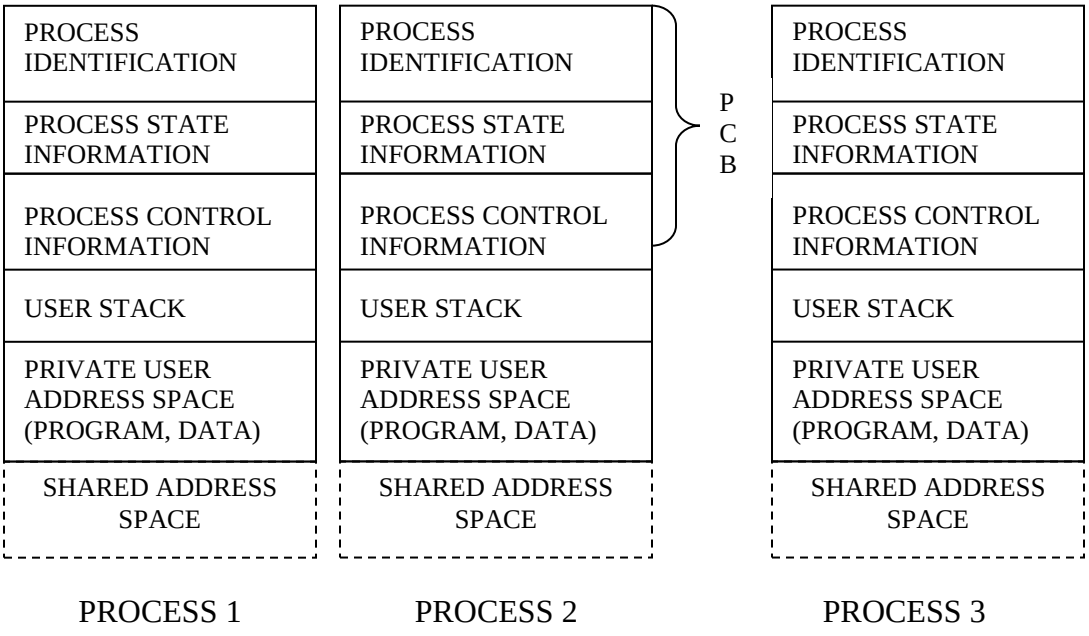
1. ***USER DATA:*** The modifiable part of user space. May include program data a user stack area and program that may be modified.

2. ***USER PROGRAM:*** The program to be executed.

3. ***USER STACK:*** Each process has one or more last-in, first-out (LIFO) system stack associated with it. A stack is used to store parameters and calling addresses for procedure and system cells.

4. ***PROCESS CONTROL BLOCK:*** Information needed by the operating system to control the process.

PROCESS IMAGES IN VIRTUAL MEMORY



ELEMENTS OF A PROCESS CONTROL BLOCK

PROCESS IDENTIFICATION

IDENTIFIERS: Numeric identifiers that may be stored with the process Control block include:

- Identifier of this process,
- Identifier of the process that created this process (present process),
- User identifier

PROCESSOR STATE INFORMATION

USER VISIBLE REGISTERS: A user-visible register that may be referenced by means of the machine language that the process executes. Typically there are from 8 to 32 of these registers, although some RISC implementations have more than 100.

CONTROL AND STATUS REGISTERS: These are a variety of processor registers that are employed to control the operation of the processor. These include:

- **P.C:** contains the address of the next instruction to be fetched {six, zero, carry equal over flow}.
- **CONDITION CODES:** Result of most recent arithmetic or logical operation.
- **STATUS INFOR:** Interrupt enable disable flags execution mode.

STACK POINTERS: Each process has one or more last-in-first out (LIFO) system stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls. The stack pointer points to the top of the stack.

PROCESS CONTROL INFORMATION

SCHEDULIING & STATE INFORMATION: This information is needed by the OS to perform its scheduling function. Typical items of information are the following defines the readiness of the process to be scheduled for execution.

PROCESS STATE PRIORITY: One or more fields may be used to be describe the scheduling priority of the process in some systems, several values are required (e.g. default, current, highest allowable)

SCHEDULIN RELATED INFORMATION: This will depend on the scheduling algorithm used examples are the amount of time that the process has waiting and the amount of time that the process executed the last time it was running.

EVENT: The identity of event that process is waiting before it can be resumed.

DATA STRUCTURING: A process may be linked to other process in a queue, ring or some other structure for example; all process in a waiting state for a particular priority level may be linked in a queue. A process may exhibit a parent-child relation with another process. The process control block may contain pointers to other processes to support these structures.

INTER PROGRESS COMMUNICATION: Various flags, signals and messages may be associated with communication between two independent processes some or all of this information may be mentioned in the process control block.

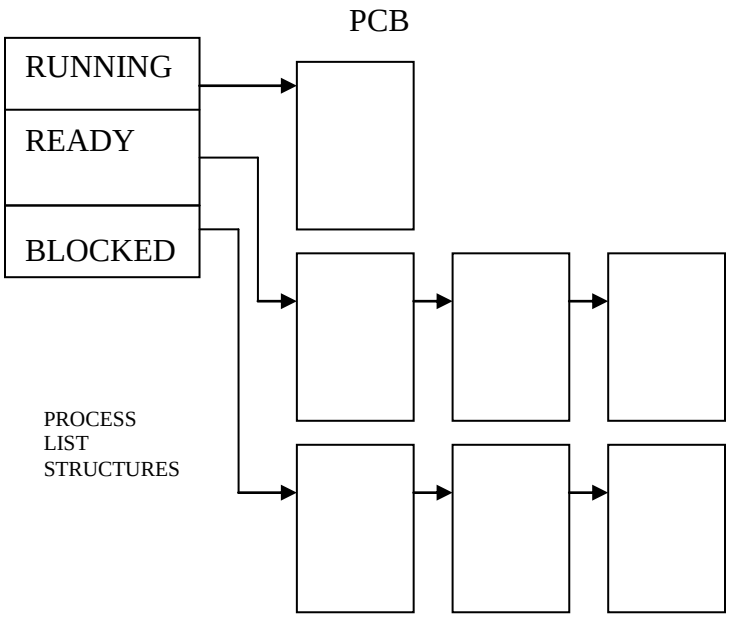
PROCESS PRIVILEGES: Processes are granted privileges in terms of the memory that may be accused and the types of instructions that may be executed in addition, privileges may apply to the use of system utilities and services.

MEMORY MANAGEMENT: This section may include pointers to segment/Page tables that describe the virtual memory assigned to this process.

RESOURCE OWNERSHIP: Resources controlled by the use of process or other resources may also be included, this information may be need by the scheduler.

PROCES LIST SRUCTURE

Error!



The process control block may contain structuring information include pointers that allow the linking of PCB. The queues that were discussed could be implemented as linked lists of PCBs.

Let us consider a single CPU system for simplicity only one process may be running at one time but several may be “ready” and several may be “blocked”. Therefore ready list of ready processes is established, and a blocked list of blocked processes. The ready list is maintained in priority so that the next process to receive the CPU is the first process on the list. The blocked list is unordered. Processes do not become unblocked in priority order; rather they unblock in the order in which the events they are waiting occur.

FUNCTIONS OF AN OPERATING SYSTEM KERNEL

PROCESS MANAGEMENT

- Process Creation and Termination
- Process Scheduling and Dispatching
- Process Switching
- Process Synchronization and support for IPC
- Management of Process Control Blocks

MEMORY MANAGEMENT

- Allocation of Address Space to Processes
- Swapping
- Page and Segment management

I/O MANAGEMENT

- Buffer Management
- Allocation of I/O Channels and Devices to Processes

SUPPORT FUNCTIONS

- Interrupt handling
- Accounting
- Monitoring