Chapter 3 Process Description and Control

BY WILLIAM STALLINGS

What is process?

- A computer platform consists of hardware resources like processor, memory, I/O modules, etc.
- Applications are developed to perform tasks by processing input and generating output.
- Writing applications directly for hardware is inefficient due to the need for common routines, multiprogramming support, and resource isolation.
- The OS acts as an intermediary layer between applications and hardware, offering a consistent interface.
- The OS provides a uniform abstraction of resources and manages their sharing and protection among applications.

OS Management of Application Execution

- **Resource Sharing**: Resources are shared among multiple applications.
- **Processor Management**: The physical processor is switched among multiple applications to give the appearance of simultaneous progress.
- **Efficient Utilization**: Both the processor and I/O devices are utilized efficiently.

Processes

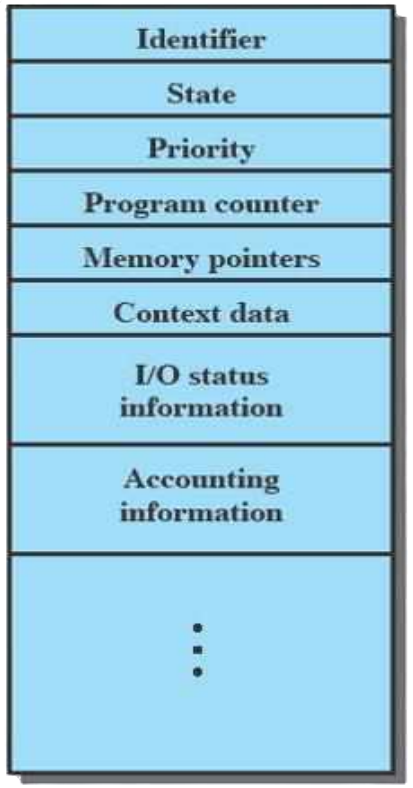
- A program in execution
- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system resources

Basic Elements

- **Identifier**: Unique process identifier.
- **State**: Running state indication.
- **Priority**: Relative priority level.
- **Program Counter**: Address of next instruction.
- **Memory Pointers**: Pointers to program code, data, and shared memory.
- **Context Data**: Processor register data during execution.
- **I/O Status**: Outstanding requests, assigned devices, and files in use.
- **Accounting Info**: Processor and clock time usage, time limits, and account numbers.

Process control block

- Contains the process elements
- It is possible to interrupt a running process and later resume execution as if the interruption had not occurred
- Created and managed by the operating system
- Key tool that allows support for multiple processes



Simplified Process Control Block

Process State

- A process's behavior is characterized by the sequence of instructions it executes, known as a trace.
- The processor's behavior is characterized by how these process traces are interleaved.
- In a simple example without virtual memory, processes are fully loaded into main memory.
- A dispatcher program switches the processor between processes.
- Traces of processes A, B, and C are shown, with process B encountering an I/O operation after the fourth instruction.
- From the processor's perspective, the traces are interleaved based on the execution of instructions.

- Figure 3.4 illustrates the interleaved traces for the first 52 instruction cycles, with shaded areas representing dispatcher code execution.
- The dispatcher's functionality remains consistent across instances.
- The operating system limits a process's execution to a maximum of six instruction cycles before interruption.

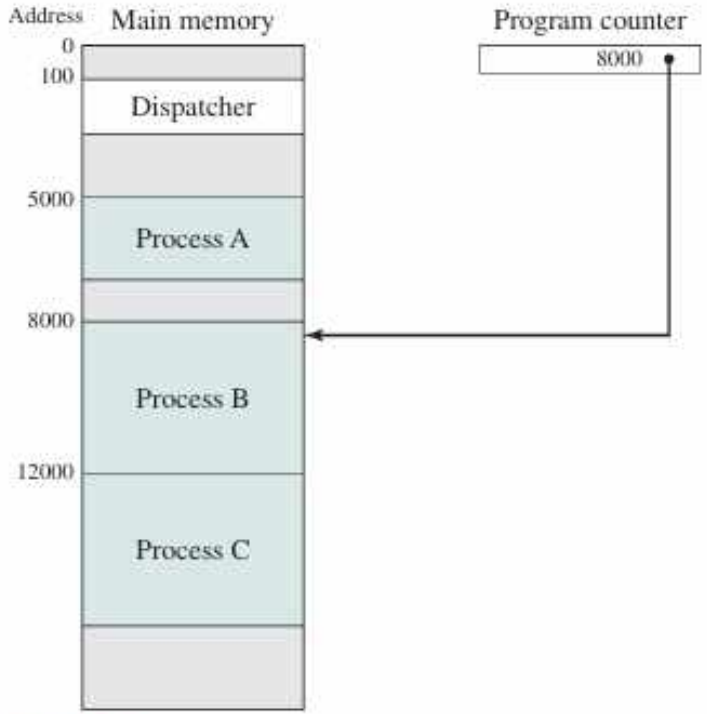


Figure 3.2 Snapshot of Example Execution (Figure 3.4) at Instruction Cycle 13

Traces of Processes of Figure 3.2

(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

1	5000	27	12004
2	5001	28	12005
3	5002	----------	Time-out
4	5003	29	100
5	5004	30	101
6	5005	31	102
----------	Time-out	32	103
7	100	33	104
8	101	34	105
9	102	35	5006
10	103	36	5007
11	104	37	5008
12	105	38	5009
13	8000	39	5010
14	8001	40	5011
15	8002	----------	Time-out
16	8003	41	100
----------	I/O request	42	101
17	100	43	102
18	101	44	103
19	102	45	104
20	103	46	105
21	104	47	12006
22	105	48	12007
23	12000	49	12008
24	12001	50	12009
25	12002	51	12010
26	12003	52	12011
		----------	Time-out

100 = Starting address of dispatcher program

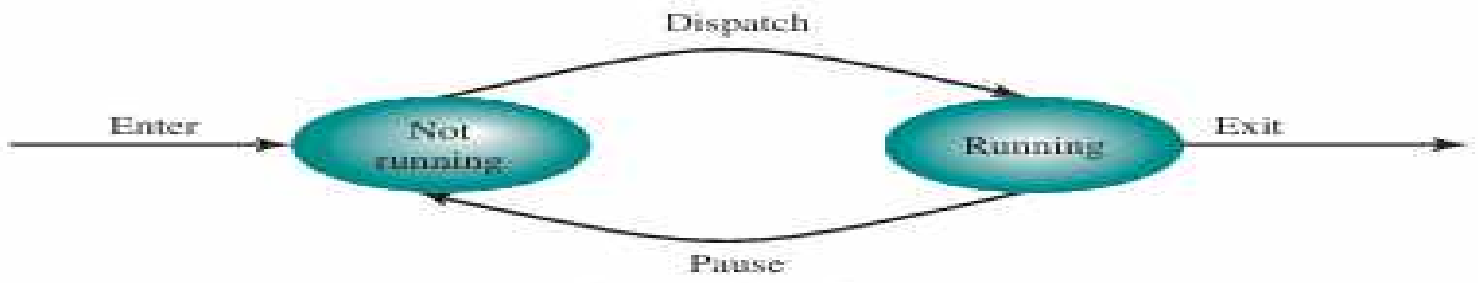
Shaded areas indicate execution of dispatcher process; first and third columns count instruction cycles; second and fourth columns show address of instruction being executed

Two-State Process Model

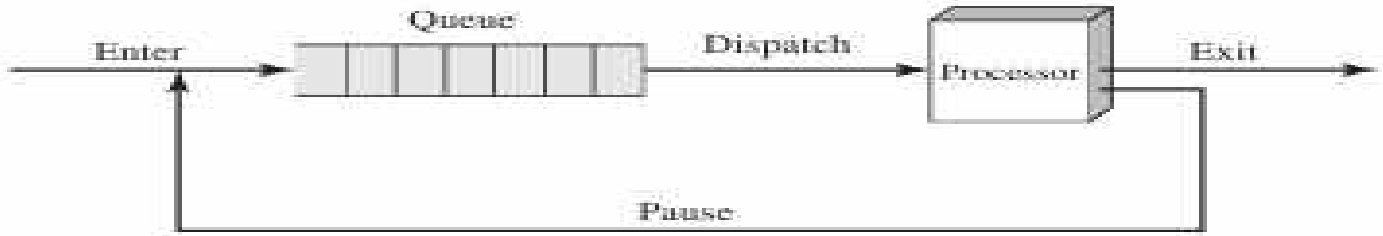
- The operating system's main responsibility is controlling process execution, including determining execution interleaving and resource allocation.
- The simplest model for OS process control involves processes being either "Running" or "Not Running."
- When a process is created, it enters the system in the "Not Running" state, waiting for execution.
- Periodically, the OS interrupts the currently running process and selects another process to run using a dispatcher.
- The process transitioning from "Running" to "Not Running" allows another process to transition to the "Running" state.
- Design elements of the OS include representing each process with a process control block (PCB), containing information like current state and memory location.

Processes not currently running are kept in a queue, awaiting their turn for execution.

- A suggested structure for this queue is a single queue where each entry is a pointer to the PCB of a specific process.



(a) State transition diagram



(b) Queueing diagram

Five-State Process Model

- If all processes were always ready to execute, the queueing discipline a first-in-first-out (FIFO) list, would be effective.
- The processor would operate in round-robin fashion, allocating a certain amount of time to each process in turn before returning it to the queue, unless it becomes blocked.
- However, in reality, some processes in the "Not Running" state are ready to execute, while others are blocked, waiting for I/O operations to complete.
- Using a single queue, the dispatcher cannot simply select the process at the oldest end of the queue because it might be blocked.
- Instead, the dispatcher needs to scan the list to find the process that is not blocked and has been waiting in the queue the longest.
- This adjustment is necessary to ensure efficient utilization of CPU time and to handle situations where processes may become blocked due to I/O operations.

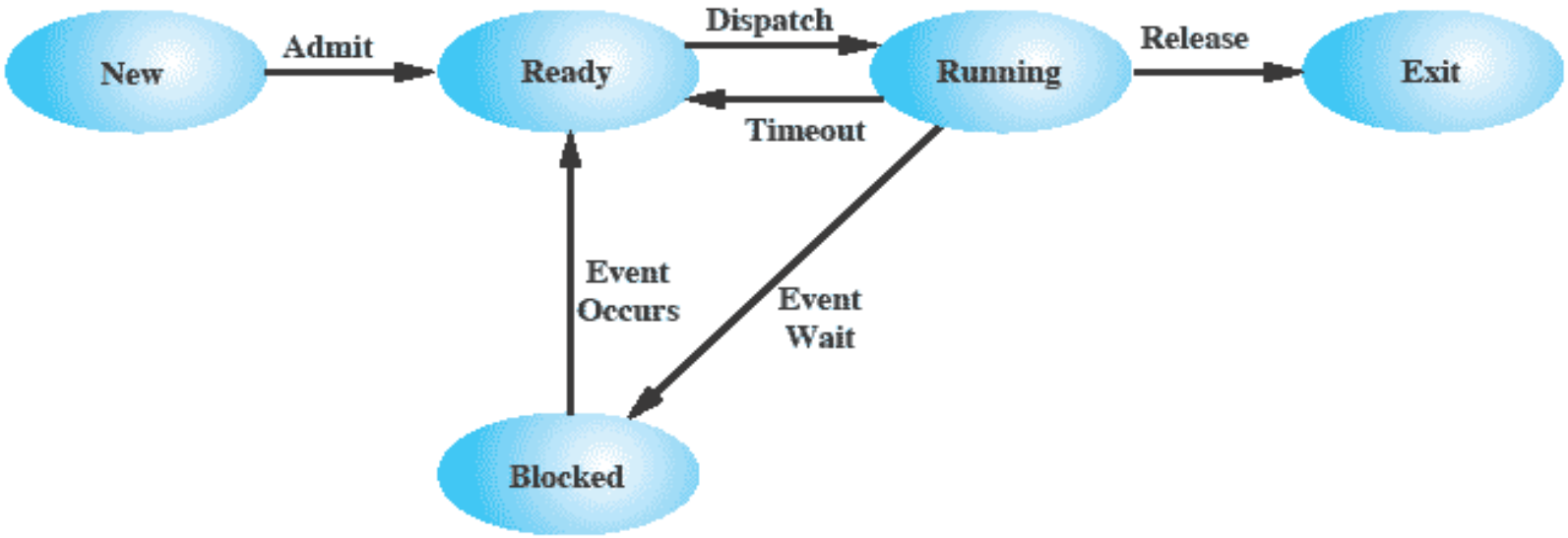


Figure 3.6 Five-State Process Model

Running: Currently executing process.
Ready: Prepared to execute.
Blocked/Waiting: Waiting for event like I/O completion.
New: Newly created, not yet admitted to executable pool.
Exit: Released from executable pool, either halted or aborted.

Possible transitions

- **Null to New**: A new process is created to execute a program, triggered by various reasons.
- **New to Ready**: The OS moves a process from the New state to the Ready state when it's ready to handle more processes, ensuring performance isn't degraded by too many active processes.
- **Ready to Running**: The OS selects a process from the Ready state to execute, done by the scheduler or dispatcher.
- **Running to Exit**: The currently running process is terminated if it completes or aborts.
- **Running to Ready**: Reasons include reaching maximum execution time, priority changes, or voluntary release of the processor.
- **Running to Blocked**: Process is put in the Blocked state if it requests something it must wait for, like I/O operations or resources.
- **Blocked to Ready**: Process moves to Ready state when the event it was waiting for occurs.
- **Blocked/Ready to Exit**: Process can be terminated by the parent or system.
- **Blocked to Exit**: Termination of a blocked process, often associated with parent termination.

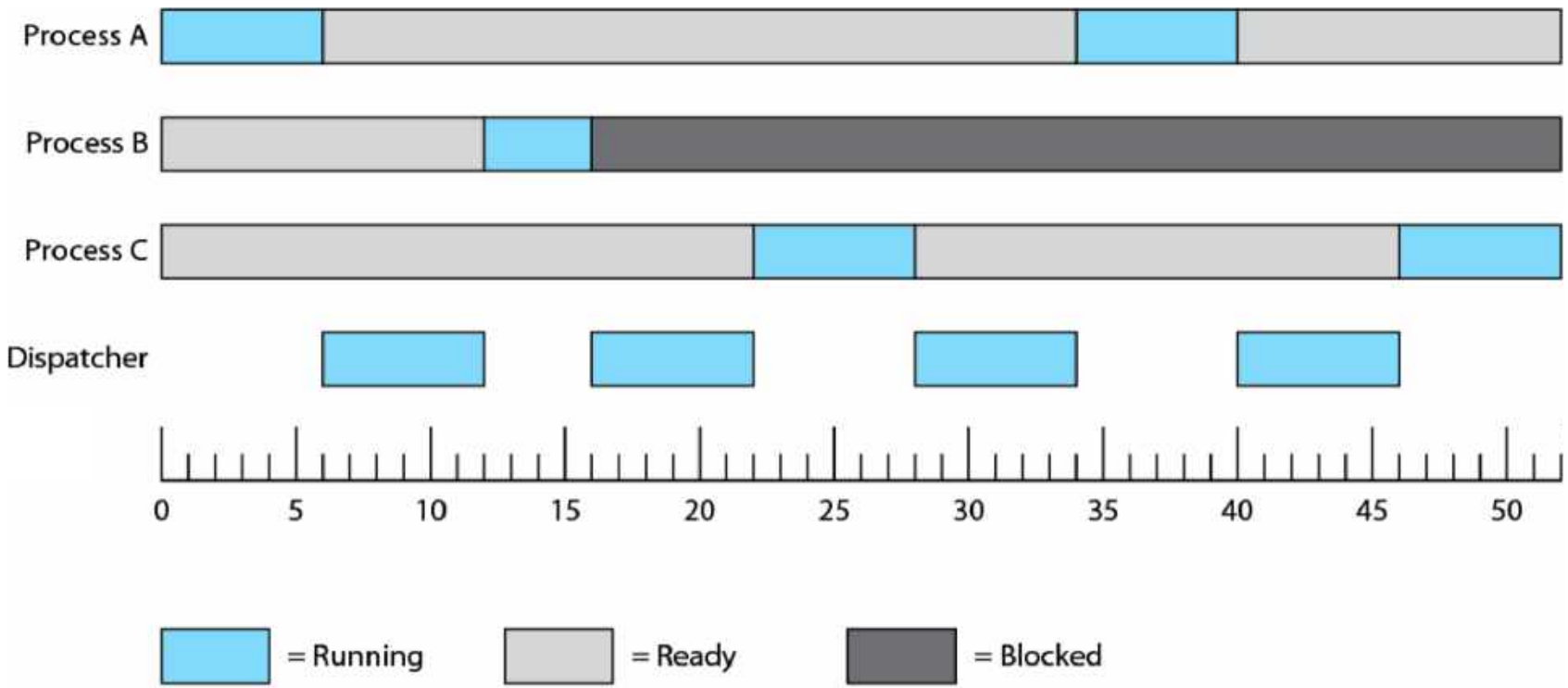
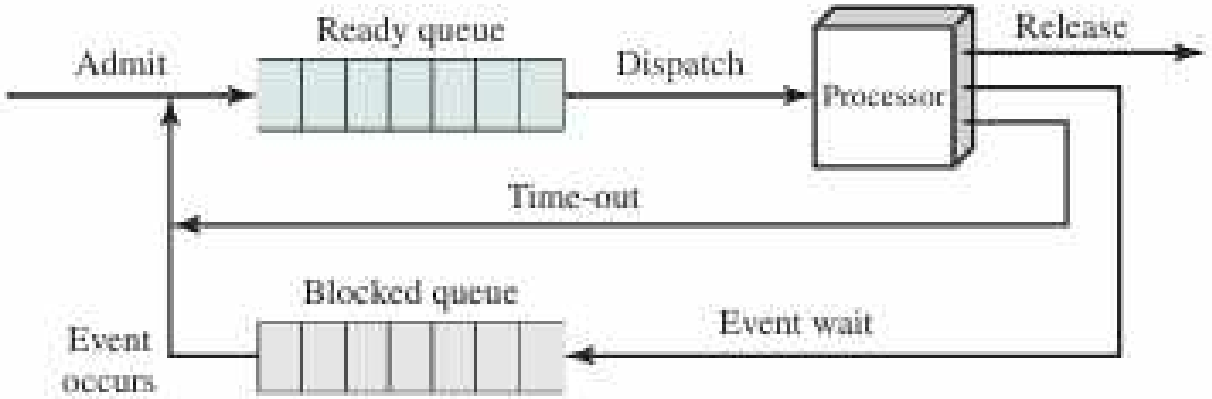
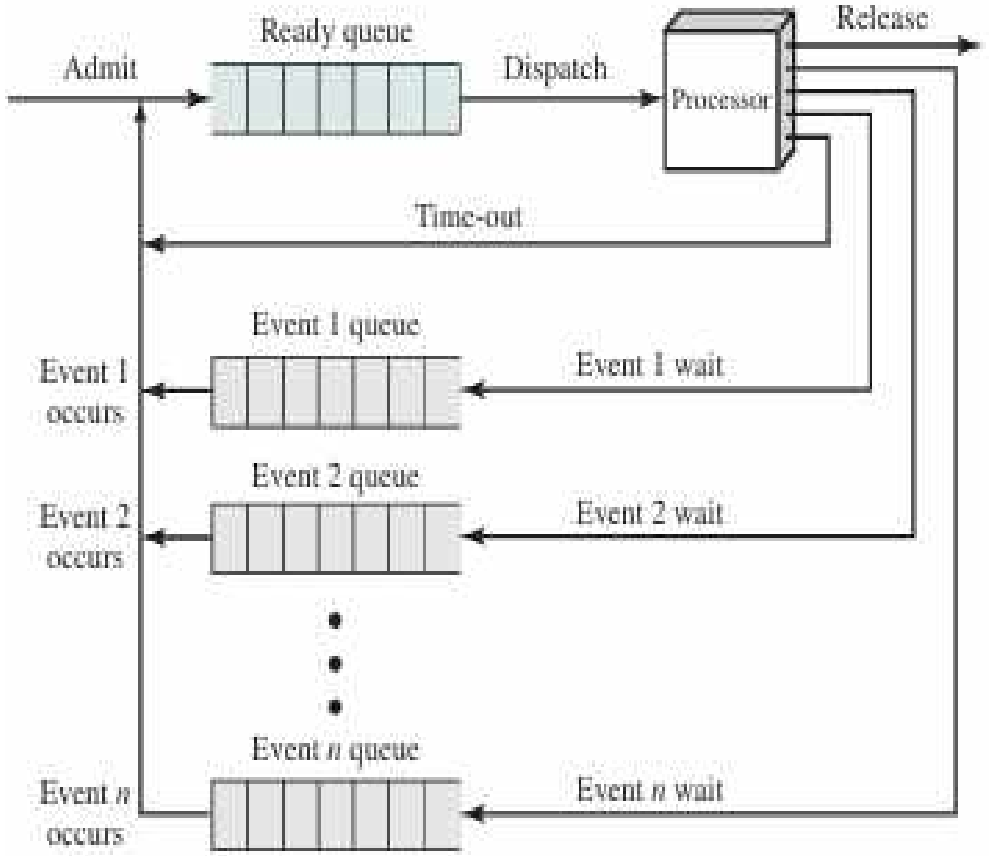


Figure 3.7 Process States for Trace of Figure 3.4



(a) Single blocked queue

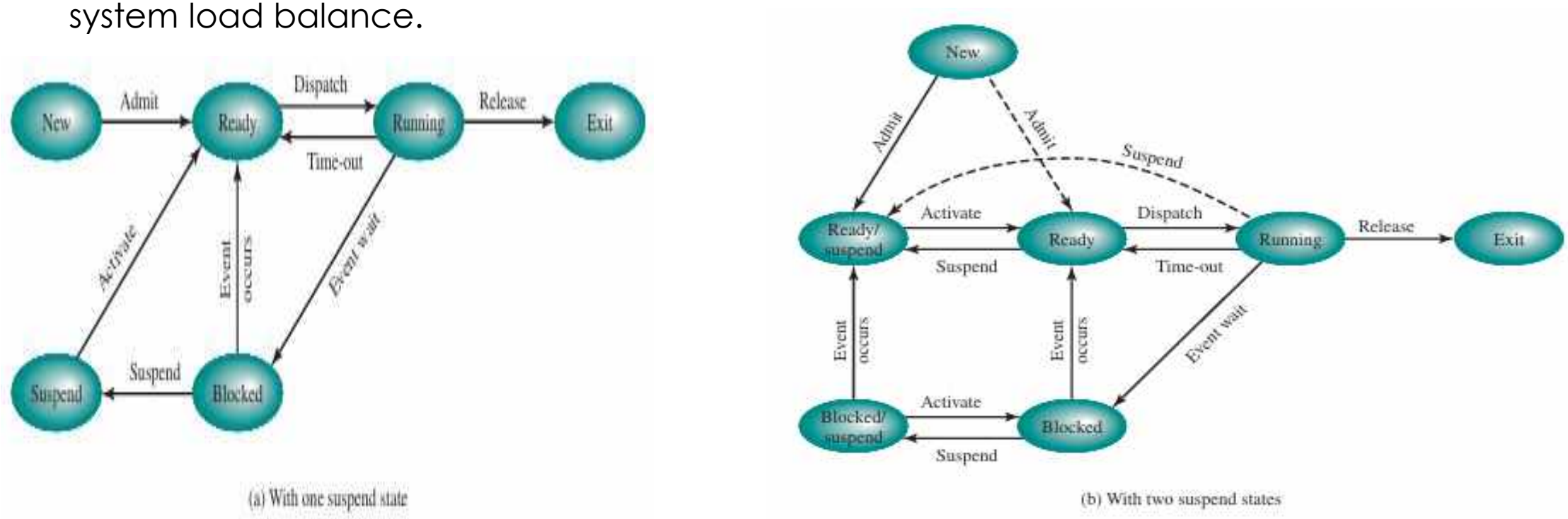


(b) Multiple blocked queues

Suspended process

- **Principal Process States**: Ready, Running, Blocked, are fundamental for modeling process behavior in operating systems.
- **Limitation of Basic Model**: Without virtual memory, all processes must be fully loaded into main memory, potentially leading to inefficiencies.
- **Challenge with Uniprogramming**: Processor idle time due to slow I/O activities persists, even with multiprogramming.
- **Memory Expansion Limitations**: Expanding main memory incurs costs and may not address increased program memory demands.
  - **Swapping Solution**
  - involves moving part of all of a process from main memory to disk
  - when none of the processes in main memory is in the Ready state, the OS swaps one of the blocked processes out on to disk into a suspend queue

Suspend State Addition: Introducing a Suspend state to the process model facilitates swapping processes out to disk when main memory is full.
Swapping Mechanism: Swapping involves transferring blocked processes to disk and bringing in new or previously suspended processes to main memory.
Preference for Suspended Processes: When selecting processes to bring into main memory, priority is given to previously suspended processes to maintain system load balance.



Process State Transition Diagram with Suspend States

Independent Concepts:

- Process waiting on an event (blocked or not) and whether it's swapped out of main memory (suspended or not) are separate.
- **Four States Needed**: To accommodate this combination, four states are required:
  - **Ready:** The process is in main memory and available for execution.
  - **Blocked:** Process is in main memory and awaiting an event.
  - **Blocked/Suspend:** Process is in secondary memory and awaiting an event.
  - **Ready/Suspend:** Process is in secondary memory but available for execution once loaded into main memory.

Important new transitions

- **Blocked to Blocked/Suspend**: If no ready processes are available, at least one blocked process is swapped out to create space for an unblocked process. This transition can occur even if ready processes exist, especially if the OS deems that the currently running process or a ready process it wants to dispatch requires more main memory to ensure performance.
- **Blocked/Suspend to Ready/Suspend**: A process in Blocked/Suspend moves to Ready/Suspend when the awaited event occurs, requiring accessible state information for suspended processes.
- **Ready/Suspend to Ready**: When no ready processes are in main memory, the OS brings one in for execution. A higher-priority Ready/Suspend process may take precedence over minimizing swapping.
- **Ready to Ready/Suspend**: Typically, the OS prefers to suspend a blocked process over a ready one to free up main memory for execution. However, if needed for memory management or priority reasons, a ready process might be suspended, especially if it's of lower priority or if the blocked process is expected to become ready soon.

other transitions

- **New to Ready/Suspend and New to Ready**: A new process can be added to either the Ready or Ready/Suspend queue upon creation. The OS creates a process control block and allocates an address space for the process. The timing of these tasks can vary, affecting system performance and resource utilization.
- **Blocked/Suspend to Blocked**: Although seemingly counterintuitive, bringing a blocked process into main memory may be justified in certain scenarios, such as when a terminated process frees up memory and a blocked process with higher priority is expected to unblock soon.
- **Running to Ready/Suspend**: A running process is typically moved to the Ready state when its time allocation expires. However, if preempted due to a higher-priority unblocked process, it may be moved directly to the Ready/Suspend queue to free up main memory.
- **Any State to Exit**: A process can terminate while in any state, either due to completion, a fatal fault condition, or termination by its creator or parent process. This transition allows for orderly process termination.

OTHER USES OF SUSPENSION

- The process is not immediately available for execution.
- The process may or may not be waiting on an event. If it is, this blocked condi tion is independent of the suspend condition, and occurrence of the blocking event does not enable the process to be executed immediately.
- The process was placed in a suspended state by an agent: either itself, a parent process, or the OS, for the purpose of preventing its execution.
- 4. The process may not be removed from this state until the agent explicitly orders the removal.

Reasons for Process Suspension

Swapping	The OS needs to release sufficient main memory to bring in a process that is ready to execute.
Other OS reason	The OS may suspend a background or utility process or a process that is suspected of causing a problem.
Interactive user request	A user may wish to suspend execution of a program for purposes of debugging or in connection with the use of a resource.
Timing	A process may be executed periodically (e.g., an accounting or system monitoring process) and may be suspended while waiting for the next time interval.
Parent process request	A parent process may wish to suspend execution of a descendent to examine or modify the suspended process, or to coordinate the activity of various descendants.

Process Description

- **Memory Tables**: Memory tables track both main (real) and secondary (virtual) memory. The OS reserves a portion of main memory for its own use, while the rest is available for processes. Key information in memory tables includes:
  - Allocation of main memory to processes.
  - Allocation of secondary memory (e.g., disk space) to processes when using virtual memory or swapping mechanisms.

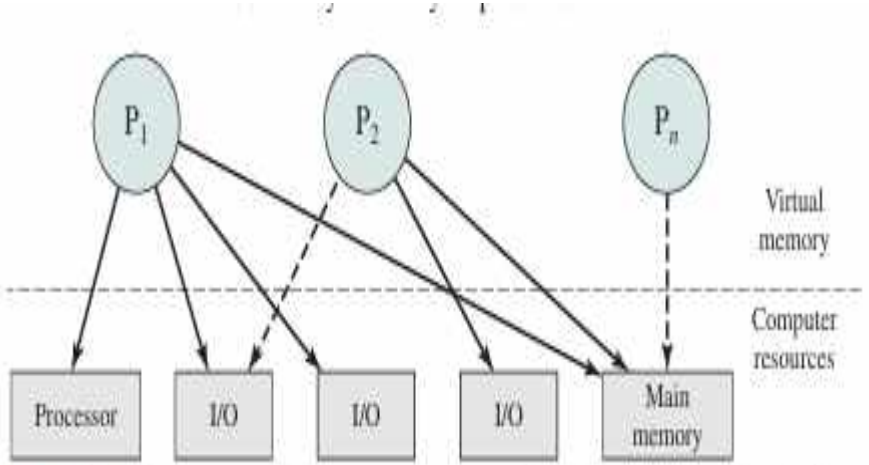


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)

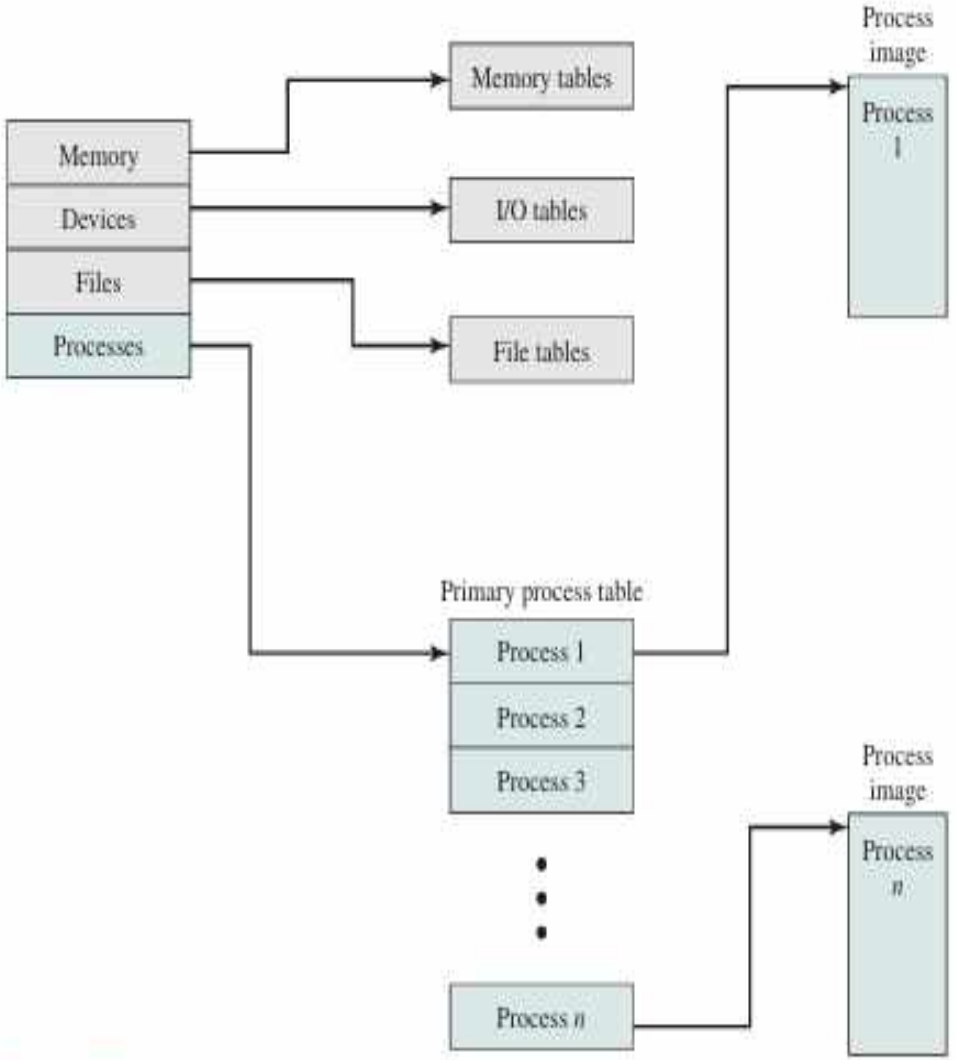


Figure 3.11 General Structure of Operating System Control Tables

I/O tables

- I/O tables manage input/output operations and devices. They contain information such as:
- Status of I/O devices (busy, available, etc.).
- Pending I/O requests and associated processes.
- Configuration and properties of I/O devices.

File Tables: File tables store information about files in the system, including:

- File names, locations, and attributes.
- Permissions and access control settings.
- File ownership and file type.

Process tables

- Process tables maintain data related to processes running in the system. This includes:
- Process identifiers (PIDs) and parent-child relationships.
- Process states, such as running, ready, blocked, or terminated.
- Process control blocks (PCBs) containing detailed information about each process, including register values, memory allocation, and scheduling parameters.

Process Control Structures

- To effectively manage and control a process, the operating system (OS) must possess certain essential information about the process. Here's a breakdown:
- **Process Location**: The OS needs to know where the process is located in memory. The physical manifestation of a process includes:
  - Programs or executable code to be executed.
  - Data locations for local and global variables, as well as defined constants.
  - A stack used for managing procedure calls and parameter passing between procedures.
  - A collection of attributes used for process control, typically stored in a process control block (PCB).

Process image

- The collection of program code, data, stack, and attributes is referred to as the process image. The location of this process image in memory depends on the memory management scheme employed by the OS.
- Typical Elements of a Process Image:

User Data
The modifiable part of the user space. May include program data, a user stack area, and programs that may be modified.

User Program
The program to be executed.

Stack
Each process has one or more last-in-first-out (LIFO) stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls.

Process Control Block
Data needed by the OS to control the process (see Table 3.5).

Process Attributes:

- The attributes associated with a process are crucial for its management. These attributes, stored in the PCB, include:
- Process ID (PID) for identification.
- Process state (e.g., running, ready, blocked, terminated).
- CPU registers and program counter values.
- Memory allocation details (e.g., base address, size).
- Information about open files and I/O operations.
- Process priority and scheduling parameters.

Process control block

- A Process Control Block (PCB), also known as a Task Control Block (TCB) or Task Struct in some operating systems, is a data structure used by operating systems to manage information about a running process or task.
- The Process Control Block (PCB) information can be categorized into three general categories:

Process Identification:

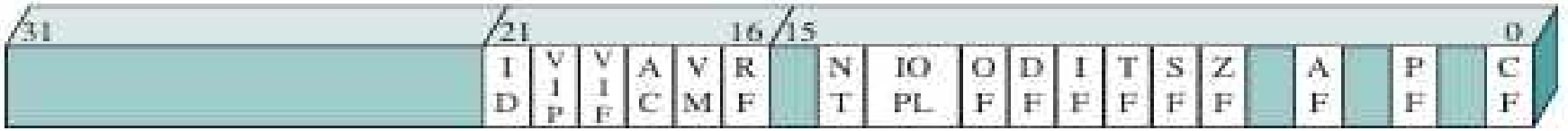
- Process ID (PID): A unique identifier assigned to the process by the operating system.
- Parent Process ID (PPID): Identifies the parent process that created the current process.
- User ID (UID): Identifies the user associated with the process.
- Group ID (GID): Identifies the group associated with the process.

Processor State Information:

- Program Counter (PC): Stores the address of the next instruction to be executed by the process.
- CPU Registers: Contains the values of CPU registers associated with the process, such as general-purpose registers, stack pointer, and program status word.
- Execution Context: Additional information needed to restore the process's execution state upon context switching, such as floating-point registers and system call context.

Process Control Information:

- **Process State:** Indicates the current state of the process (e.g., running, ready, blocked, terminated).
- **Memory Management Information:** Details about the memory allocated to the process, such as base address, size, and limits.
- **I/O Status Information:** Tracks the status of I/O operations initiated by the process, including pending I/O requests and open files.
- **Priority and Scheduling Information:** Specifies the priority of the process and scheduling parameters used by the operating system to determine the order of process execution.
- **Accounting Information**: Records resource usage statistics, such as CPU time, execution history, and resource utilization.

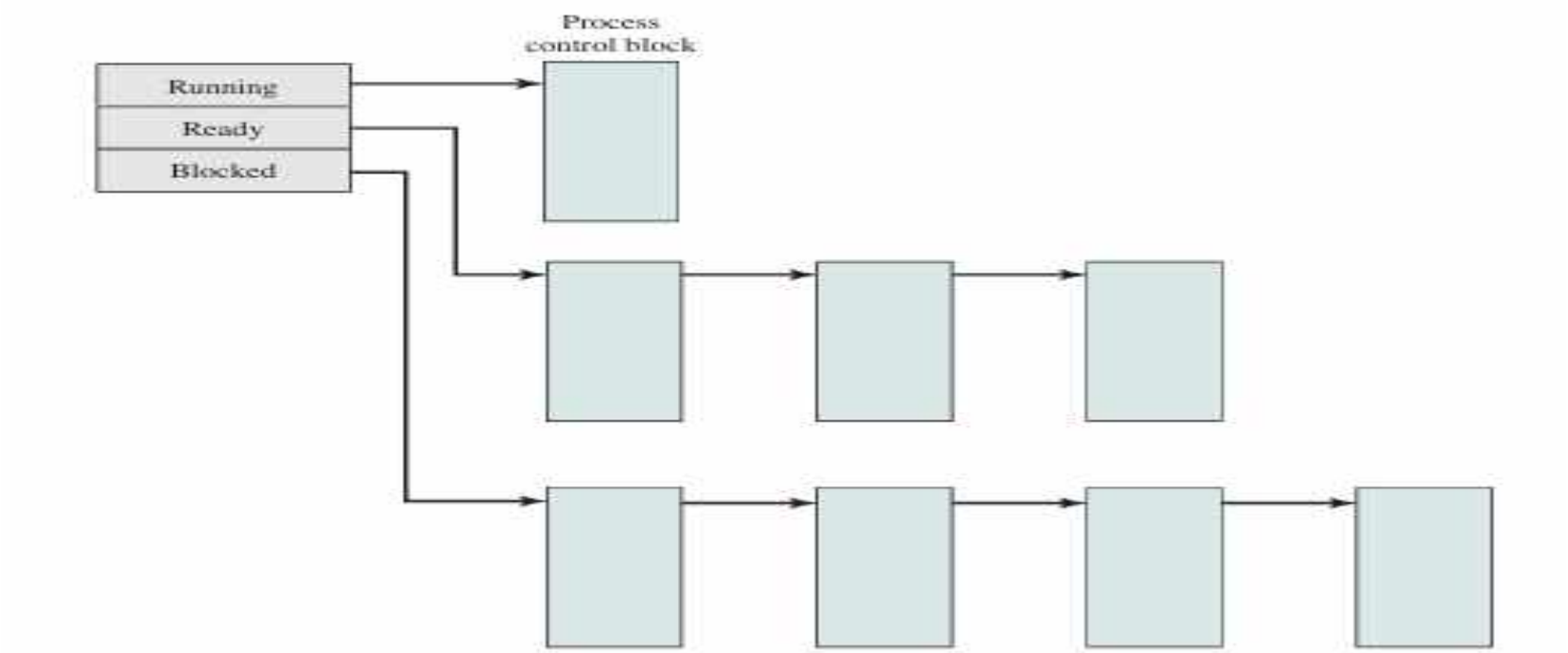


ID = Identification flag
VIP = Virtual interrupt pending
VIF = Virtual interrupt flag
AC = Alignment check
VM = Virtual 8086 mode
RF = Resume flag
NT = Nested task flag
IOPL = I/O privilege level
OF = Overflow flag
DF = Direction flag
IF = Interrupt enable flag
TF = Trap flag
SF = Sign flag
ZF = Zero flag
AF = Auxiliary carry flag
PF = Parity flag
CF = Carry flag

Figure 3.12 x86 EFLAGS Register

THE ROLE OF THE PROCESS CONTROL BLOCK

- **Process Control Block (PCB)**: Central data structure in an OS containing all information about a process necessary for the OS. Virtually every OS module interacts with PCBs, including those for scheduling, resource allocation, interrupts, and performance analysis.
- **Importance of PCBs**: The set of PCBs defines the state of the OS, highlighting their critical role in system management.
- **Design Issue**: Accessing information in PCBs isn't difficult due to unique process IDs. However, protection against bugs and design changes is crucial.
- **Protection Challenges**:
  - A bug in a routine, like an interrupt handler, could damage PCBs, disrupting process management.
  - Structural or semantic changes to PCBs could impact multiple OS modules.
- **Solution**: Addressed by enforcing all OS routines to go through a handler routine dedicated to protecting PCBs. This handler becomes the sole authority for reading and writing PCBs.
- **Trade-offs**: This approach balances performance concerns with the need for system integrity and correctness assurance.



Process List Structures

Process Control

- Modes of execution, often referred to as privilege levels, play a crucial role in ensuring the security and integrity of the operating system and user programs. Here's a breakdown of the key concepts discussed:
- **Two Modes of Execution**:
  - **User Mode**: Also known as less-privileged mode, where user programs typically execute. Access to certain instructions and memory regions is restricted.
  - **Kernel Mode**: Also known as system mode, control mode, or supervisor mode. In this mode, the operating system has complete control over the processor and system resources. Access to privileged instructions and critical system resources is allowed.

Purpose of Two Modes

- **Protection:** The use of two modes helps protect the operating system and key system data structures, such as process control blocks, from unauthorized access or interference by user programs.
- **Security:** Kernel mode ensures that only trusted system components have access to critical resources and can execute privileged instructions, enhancing system security and stability.

Transition between Modes

- **Program Status Word (PSW)**: Typically, a bit in the PSW indicates the current mode of execution. This bit is changed in response to certain events, such as system calls or interrupts.
- **Privilege Levels**: Processors often use privilege levels, represented by numeric values (e.g., levels 0 to 3), to differentiate between user and kernel modes. Higher privilege levels have greater access to privileged instructions and system resources.
- **Mode Change**: When a user program makes a system call or when an interrupt occurs, the mode is changed to kernel mode to execute the corresponding operating system routine. After completing the operation, the mode is restored to user mode before returning control to the user program.

Example with Itanium Processor:

- The Intel Itanium processor uses a 2-bit Current Privilege Level (CPL) field in the Processor Status Register (PSR) to indicate the privilege level.
- Level 0 is the most privileged level (kernel mode), while level 3 is the least privileged level (user mode).
- When an interrupt occurs, the CPL is automatically set to level 0, indicating kernel mode execution.
- After handling the interrupt, the PSR is restored to its original state, transitioning back to user mode.
- Similarly, system calls involve transitioning from user mode to kernel mode to execute the requested service and then returning to user mode after completion.

Typical Functions of an Operating System Kernel

Process Management

- Process creation and termination
- Process scheduling and dispatching
- Process switching
- Process synchronization and support for interprocess communication
- 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

Process Creation

- **Assign a Process Identifier:**
  - Assign a unique identifier to the new process.
  - Add a new entry to the primary process table.
- **Allocate Process Space:**
  - Allocate space for the process image.
  - Determine space needed for user address space and stack.
  - Allocate space for a process control block.
- **Initialize Process Control Block (PCB):**
  - Initialize process identification and processor state information.
  - Set process control information based on default values or requested attributes.
- **Set Linkages:**
  - Place the new process in the appropriate scheduling queue (e.g., Ready or Ready/Suspend).
- **Create or Expand Data Structures:**
  - Create or expand data structures such as accounting files for billing and performance assessment.