Module-2
Process & IPC

Process Concept

An operating system executes a variety of programs:

- Batch system – jobs
- Time-shared systems – user programs or tasks

Process – a program in execution; process execution must progress in sequential fashion

Program is ***passive*** entity stored on disk (**executable file**),
process is ***active entity*** when executable file loaded into memory

Execution of program started via GUI mouse clicks, command line entry of its name, etc

Process Concept (Cont.)

When a program is loaded into main memory, it is divided into four main sections:

Text Segment (Code Segment):

Contains the program's compiled machine code (instructions).

Data Segment:

Stores global and static variables, which are allocated memory once and maintain their values throughout the program's life.

Heap Segment:

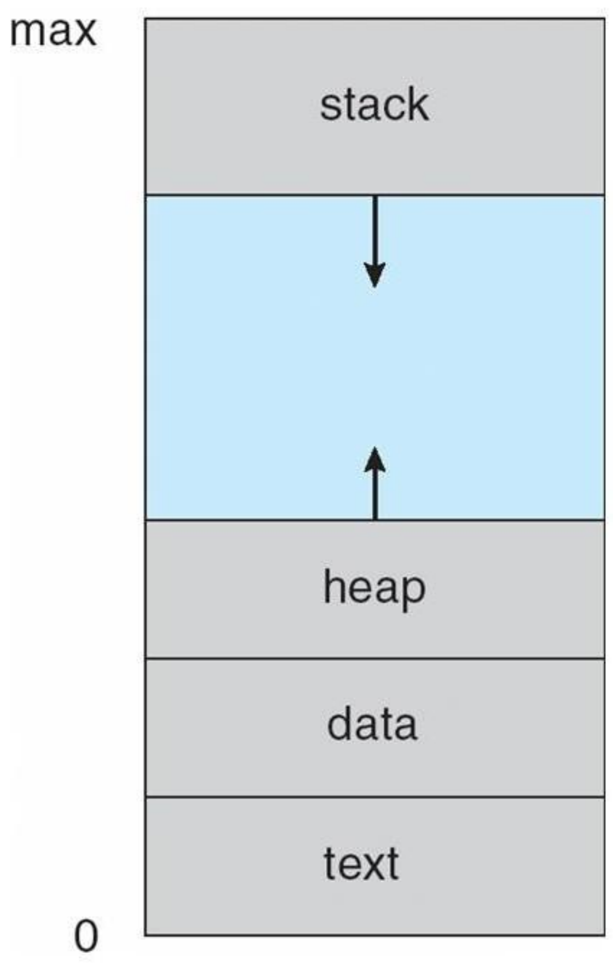
A region of memory used for dynamic memory allocation. Memory is allocated or deallocated during the program's execution using functions like malloc().

Stack Segment :

Used for temporary data, including function parameters, local variables, and return addresses. The stack grows downwards in memory, while the heap typically grows upwards.

CONTD..

Process in Memory



```
#include <stdio.h>

int k = 10;   // global variable
static q=15
int main() {
   int a=7;
   static int c=9;
   float b = 5.5;
  int *p;

   printf("Hello, World! \n");
   p=(int *) malloc(sizeof(int));
   return 0;
}
```
Fit this code in memory

```
 +----------------------------+
 |        STACK               |  ← Grows downward
 |  int a;                    |
 |  float b = 5.5;            |
 |  int *p; (holds heap addr) |
 +----------------------------+
 |        HEAP                |  ← Grows upward
 |  malloc(sizeof(int)) block |
 +----------------------------+
|   DATA Segment              |
 |   int k = 10;              |
 |   static int q = 15;       |
 |   static int c = 9;        |
 +----------------------------+
 |    TEXT Segment            |
 |   main(), printf()         |
 +----------------------------+
```

Process States

In an operating system, a process can be in one of several states, representing its current activity or status. These states dictate how the OS manages and allocates resources to the process. The main process states are New, Ready, Running, Waiting (or Blocked), and Terminated.

New: The process is newly created and has not yet been admitted to the system for execution.

Ready: The process is prepared to execute and is waiting for the CPU to become available.

Running: The process is currently being executed by the CPU.

Waiting (or Blocked): The process is temporarily halted, waiting for an event to occur, such as the completion of an I/O operation or the availability of a resource.

Terminated: The process has finished executing and is being removed from the system.

Suspend Ready State-

A process moves from ready state to suspend ready state if a process with higher priority has to be executed but the main memory is full.

Moving a process with lower priority from ready state to suspend ready state creates a room for higher priority process in the ready state.

The process remains in the suspend ready state until the main memory becomes available. When main memory becomes available, the process is brought back to the ready state.

Suspend Wait State

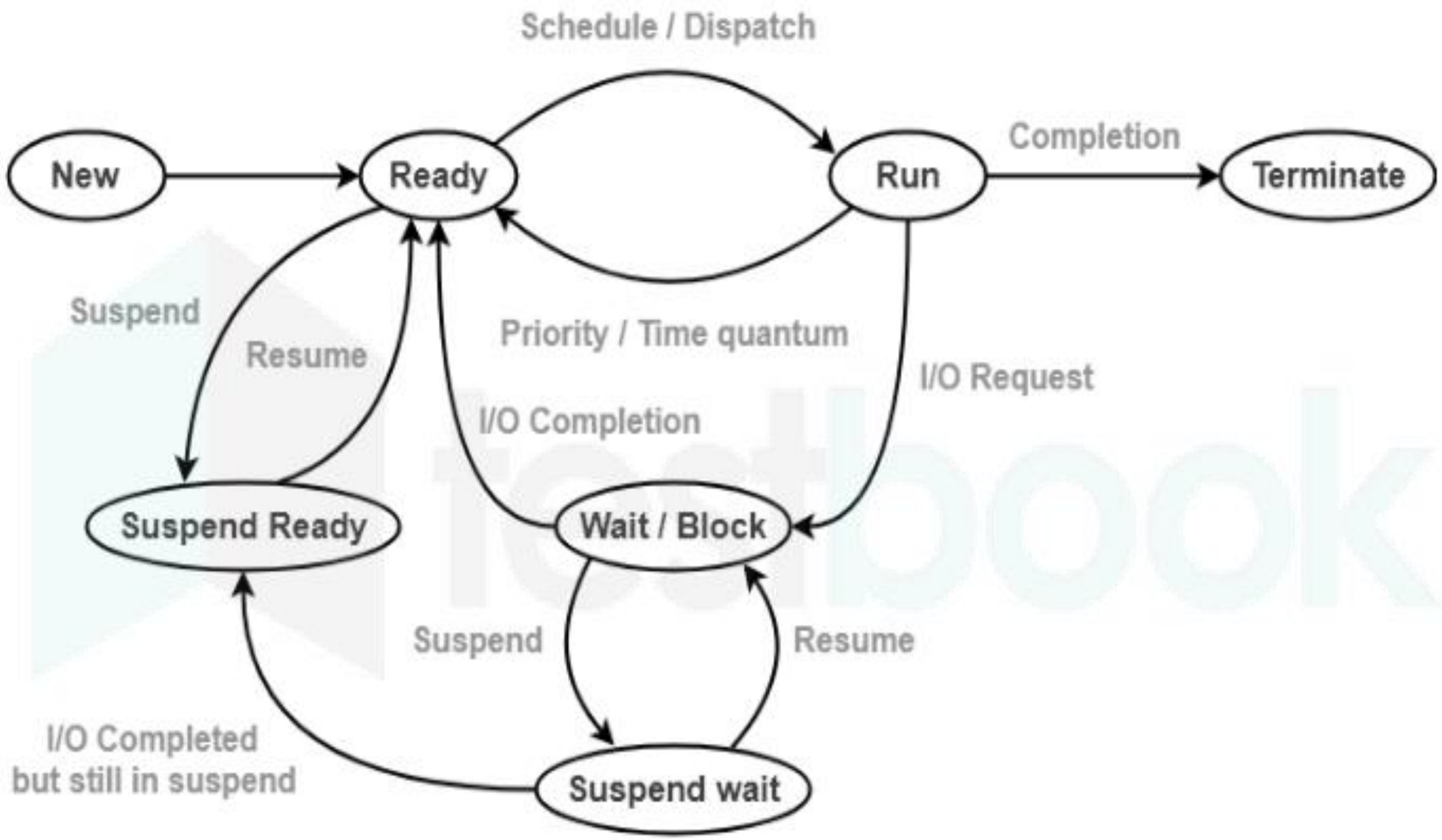
A process moves from wait state to suspend wait state if a process with higher priority has to be executed but the main memory is full.

Moving a process with lower priority from wait state to suspend wait state creates a room for higher priority process in the ready state.

After the resource becomes available, the process is moved to the suspend ready state.

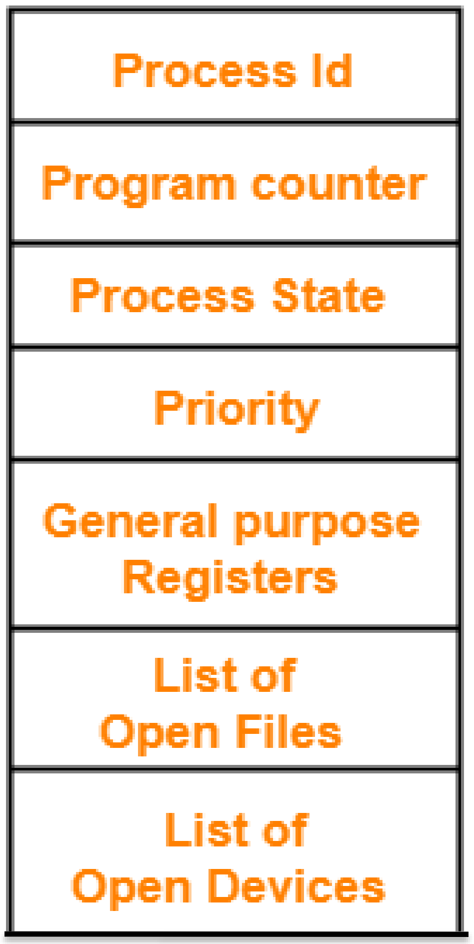
After main memory becomes available, the process is moved to the ready state.

Diagram of Process State



Process State Diagram

Process Control Block (PCB)



Process Control Block (PCB)

❖Process Control Block (PCB) is a data structure that stores information about a particular process.

❖This information is required by the CPU while executing the process.

❖Each process is identified by its own process control block (PCB).

❖It is also called as context of the process.

Process Control Block (PCB)

1. Process Id-

➢Process Id is a unique Id that identifies each process of the system uniquely.

➢A process Id is assigned to each process during its creation.

2. Program Counter-

➢Program counter specifies the address of the instruction to be executed next.

➢Before execution, program counter is initialized with the address of the first instruction of the program.

➢After executing an instruction, value of program counter is automatically incremented to point to the next instruction.

➢This process repeats till the end of the program.

3. Process State-

Each process goes through different states during its lifetime.

➢Process state specifies the current state of the process.

4. Priority-

➢Priority specifies how urgent is to execute the process.

➢Process with the highest priority is allocated the CPU first among all the processes.

5. General Purpose Registers-

➢General purpose registers are used to hold the data of process generated during its execution.

➢Each process has its own set of registers which are maintained by its PCB.

6. List of Open Files-

➢Each process requires some files which must be present in the main memory during its execution.

➢PCB maintains a list of files used by the process during its execution.

7. List of Open Devices-

PCB maintains a list of open devices used by the process during its execution.

Process Scheduling

Definition

Scheduling is a method that is used to distribute valuable computing resources, ususally process threads, data flows and applications that need them.

Scheduling is done to balance the load on the system and ensure equal distribution of resources and give some set of rules.

Operating System - Process Scheduling

Definition

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

CONTD..

Process 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.

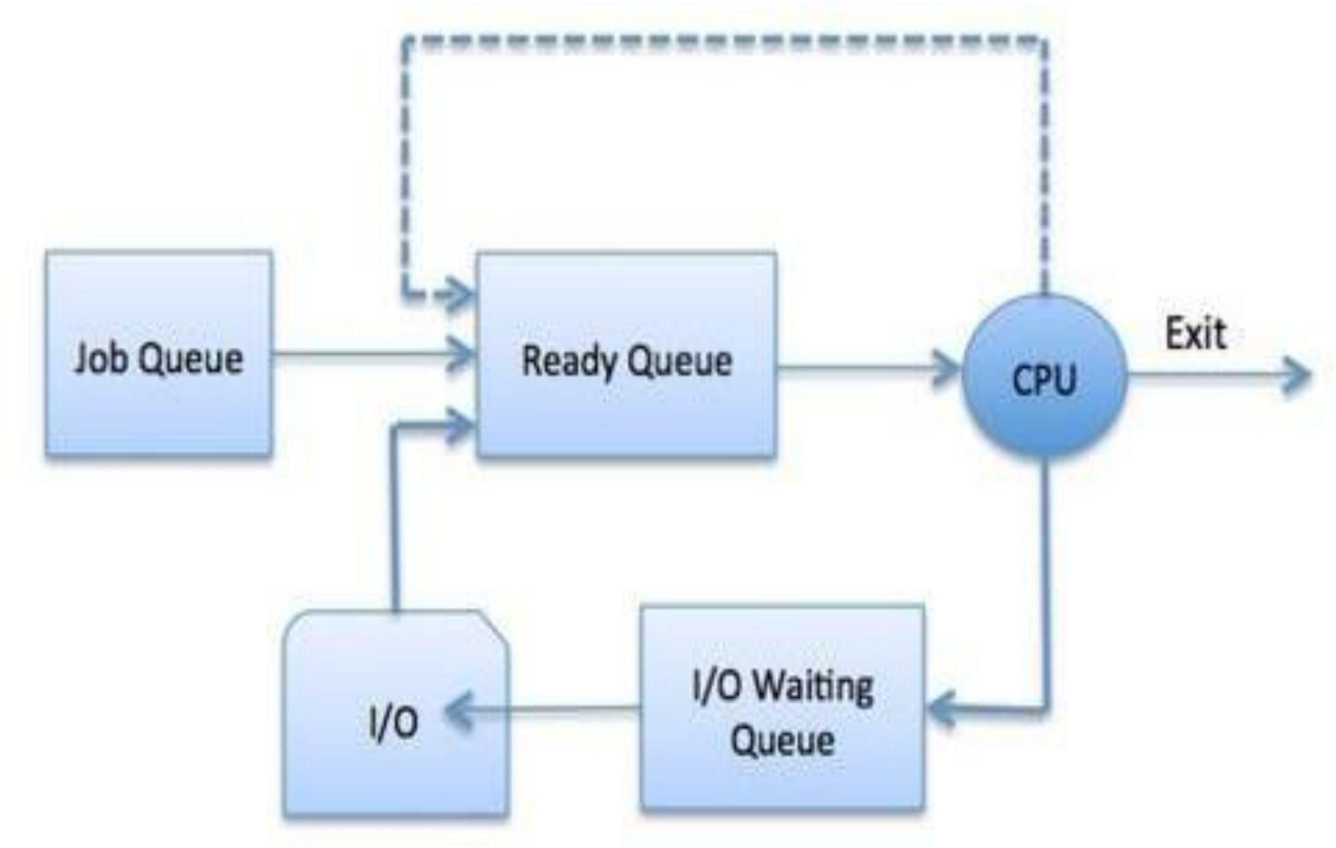
The Operating System maintains the following important process scheduling queues −

Job queue − This queue keeps all the processes in the system.

Ready queue − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.

Device queues − The processes which are blocked due to unavailability of an I/O device constitute this queue.

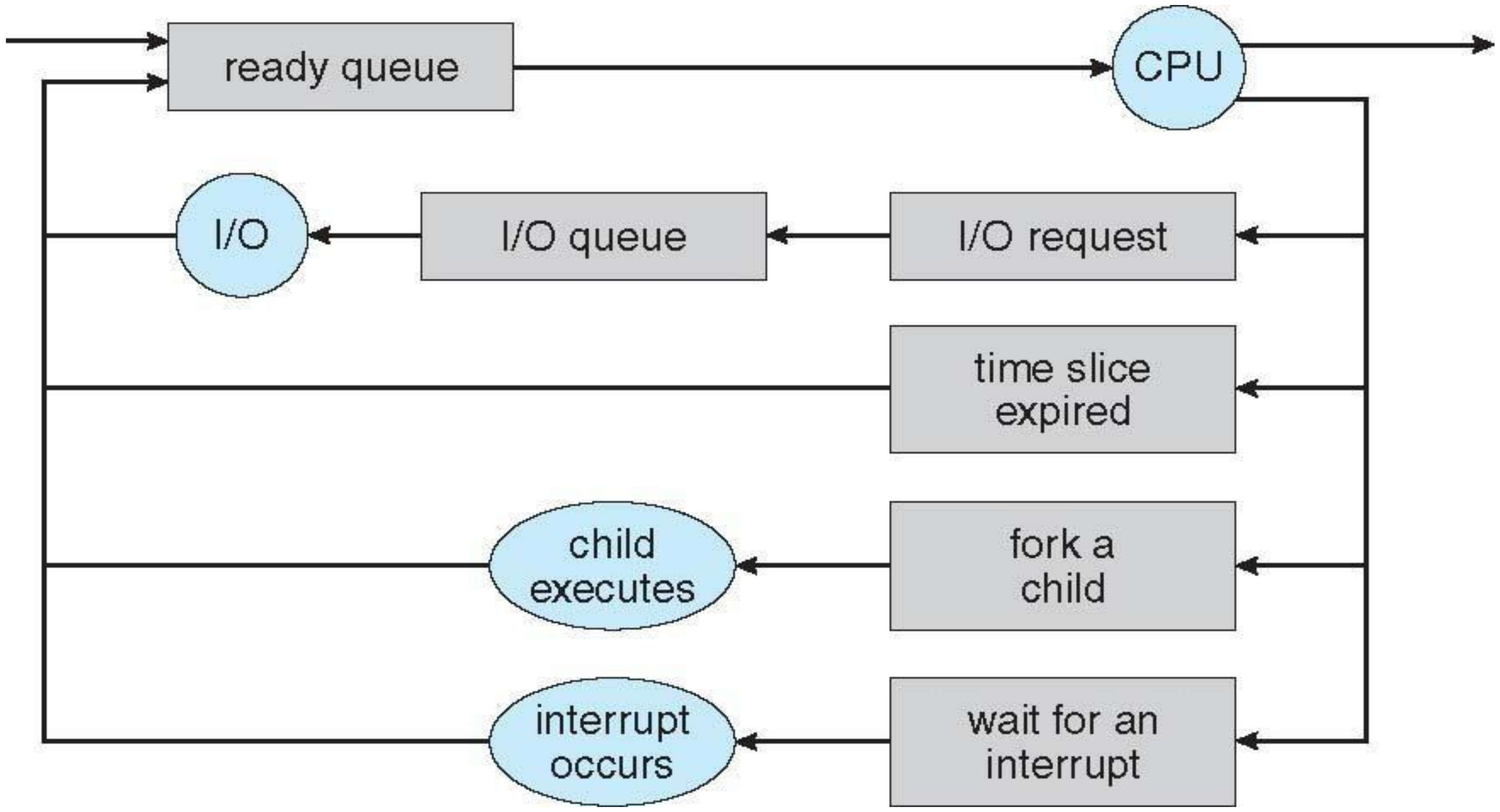
CONTD..



CONTD..

Representation of Process Scheduling

Queueing diagram represents queues, resources, flows



CONTD..

Schedulers

➢Schedulers are special system software which handle 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 –

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

CONTD..

Long Term Scheduler

A long-term scheduler is a scheduler that is responsible for bringing processes from the **JOB** queue **(or secondary memory)** into the **READY** queue **(or main memory)**.

A long-term scheduler determines which programs will enter into the **RAM for processing by the CPU**.

Long-term schedulers are also called **Job Schedulers**.

They are responsible for the degree of multiprogramming, **i.e., managing the total processes present in the READY queue**.

I/O-bound process – An I/O-bound process in an operating system is a process whose execution time is spent mostly on Input/Output (I/O) operations(like reading from disk, writing to a file, waiting for network data, or interacting with a user), rather than on CPU computations.
CPU-bound process – A CPU-bound process in an operating system is the opposite of an I/O-bound process. It is a process that spends most of its execution time performing computations (CPU time) rather than waiting for input/output operations.

CONTD..

Short Term Scheduler

- A short-term scheduler, or CPU scheduler, is an operating system component that selects a process from the ready queue and assigns it the CPU for execution.

- It operates very frequently, managing the allocation of the CPU to various processes and ensuring the system remains efficient and multitasking.

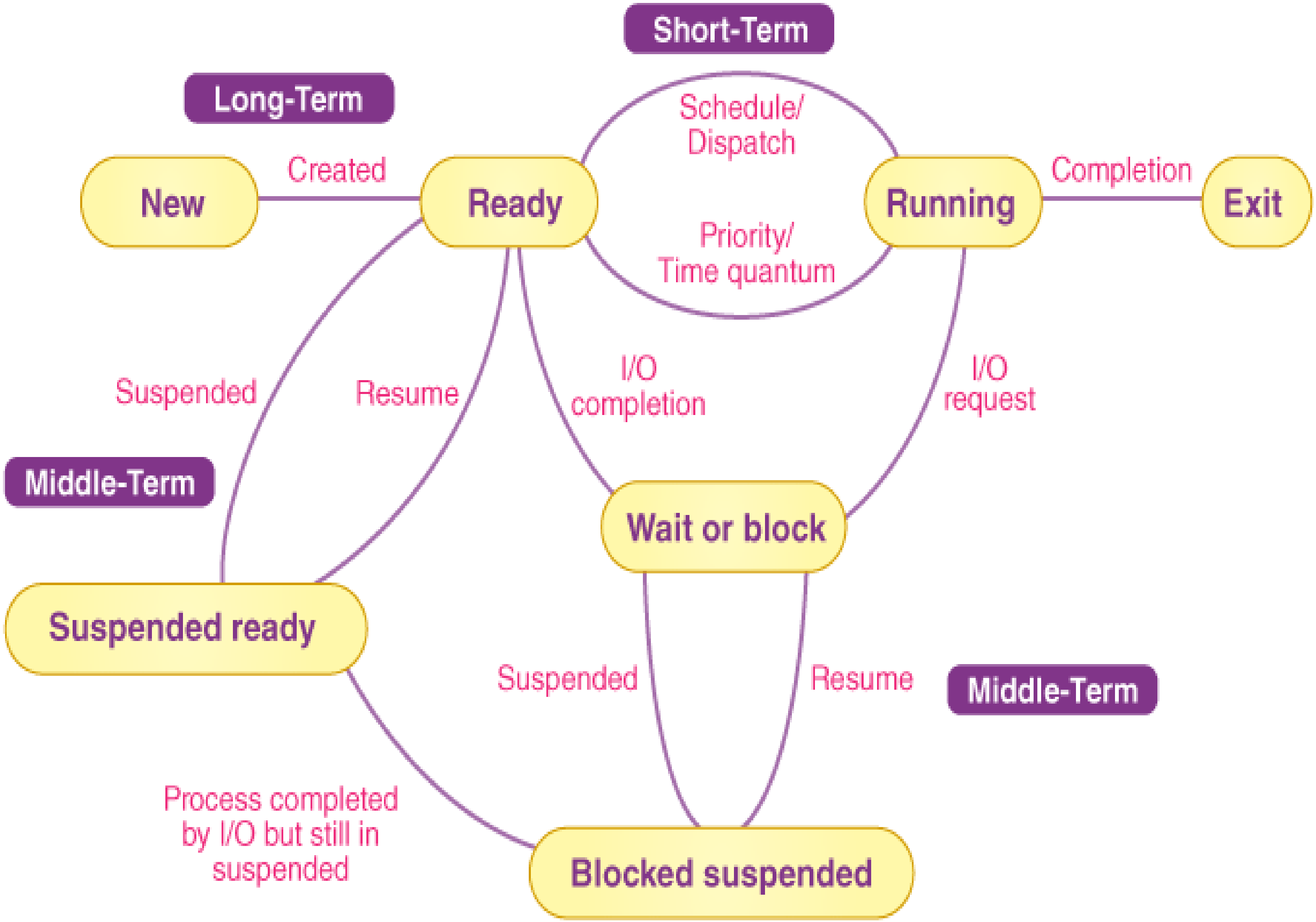
- It helps an operating system remain responsive by allocating the CPU to processes when they are ready to run.

CONTD..

Medium Term Scheduler

A medium-term scheduler is an operating system component responsible for swapping out and swapping in processes between main memory and secondary storage to manage memory and control the degree of multiprogramming.

- ➢A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion.
- ➢In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage.
- ➢This process is called swapping, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.
- ➢ It temporarily removes processes from memory to free up resources or because they are suspended (e.g., waiting for I/O), and brings them back in later when conditions are favorable.


Short-Term
Long-Term
Schedule/ Dispatch
New
Created
Ready
Running
Completion
Exit
Priority/ Time quantum
Suspended
Resume
I/O completion
I/O request
Middle-Term
Wait or block
Suspended ready
Suspended
Resume
Middle-Term
Process completed by I/O but still in suspended
Blocked suspended

Schedulers

Short-term scheduler (or **CPU scheduler**) – selects which process should be executed next and allocates CPU

Sometimes the only scheduler in a system

Short-term scheduler is invoked frequently (milliseconds) $\Rightarrow$ (must be fast)

Long-term scheduler (or **job scheduler**) – selects which processes should be brought into the ready queue

Long-term scheduler is invoked infrequently (seconds, minutes) $\Rightarrow$ (may be slow)

The long-term scheduler controls the **degree of multiprogramming**

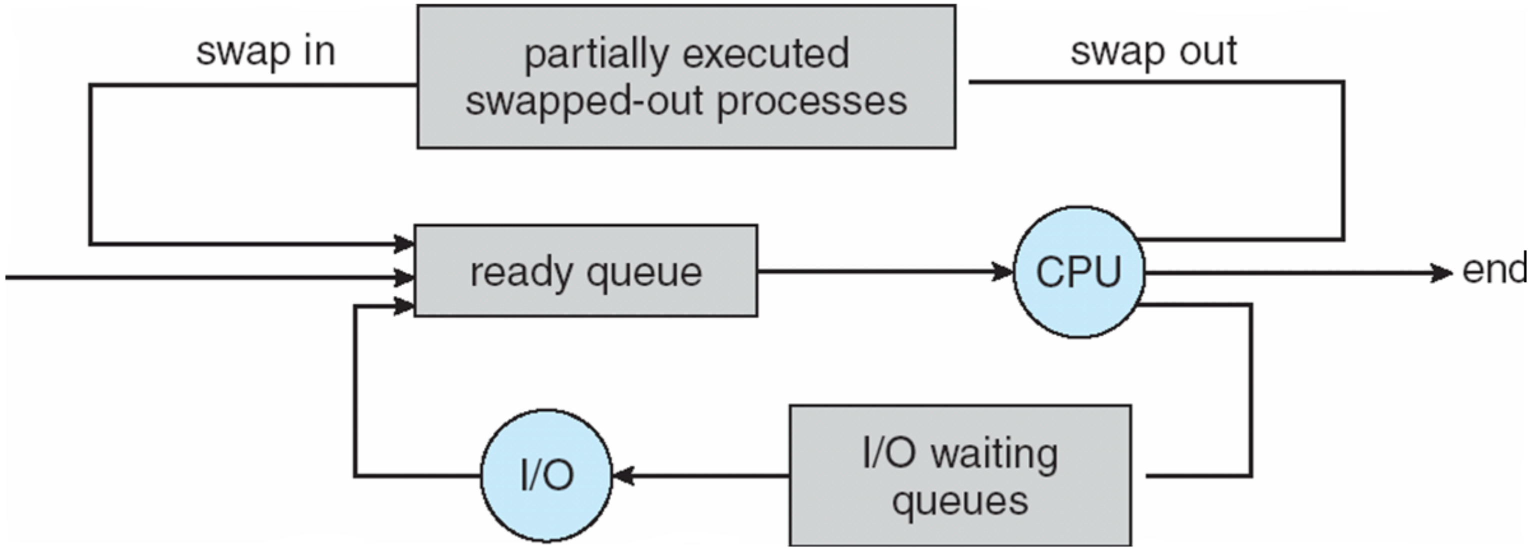
Processes can be described as either:

I/O-bound process – An I/O-bound process in an operating system is a process whose execution time is spent mostly on Input/Output (I/O) operations(like reading from disk, writing to a file, waiting for network data, or interacting with a user), rather than on CPU computations.

CPU-bound process – A CPU-bound process in an operating system is the opposite of an I/O-bound process. It is a process that spends most of its execution time performing computations (CPU time) rather than waiting for input/output operations.

Addition of Medium Term Scheduling

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**



CONTD..

Context Switch

- When CPU switches to another process, the system must **save the state** of the old process and load the **saved state** for the new process via a **context switch**
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB $\rightarrow$ the longer the context switch

The Context Switching Process

1. Interrupt or System Call:
A trigger (like a timer interrupt or a process needing to wait for I/O) occurs.

2. Save Context:
The operating system saves the state of the currently running process into its PCB.

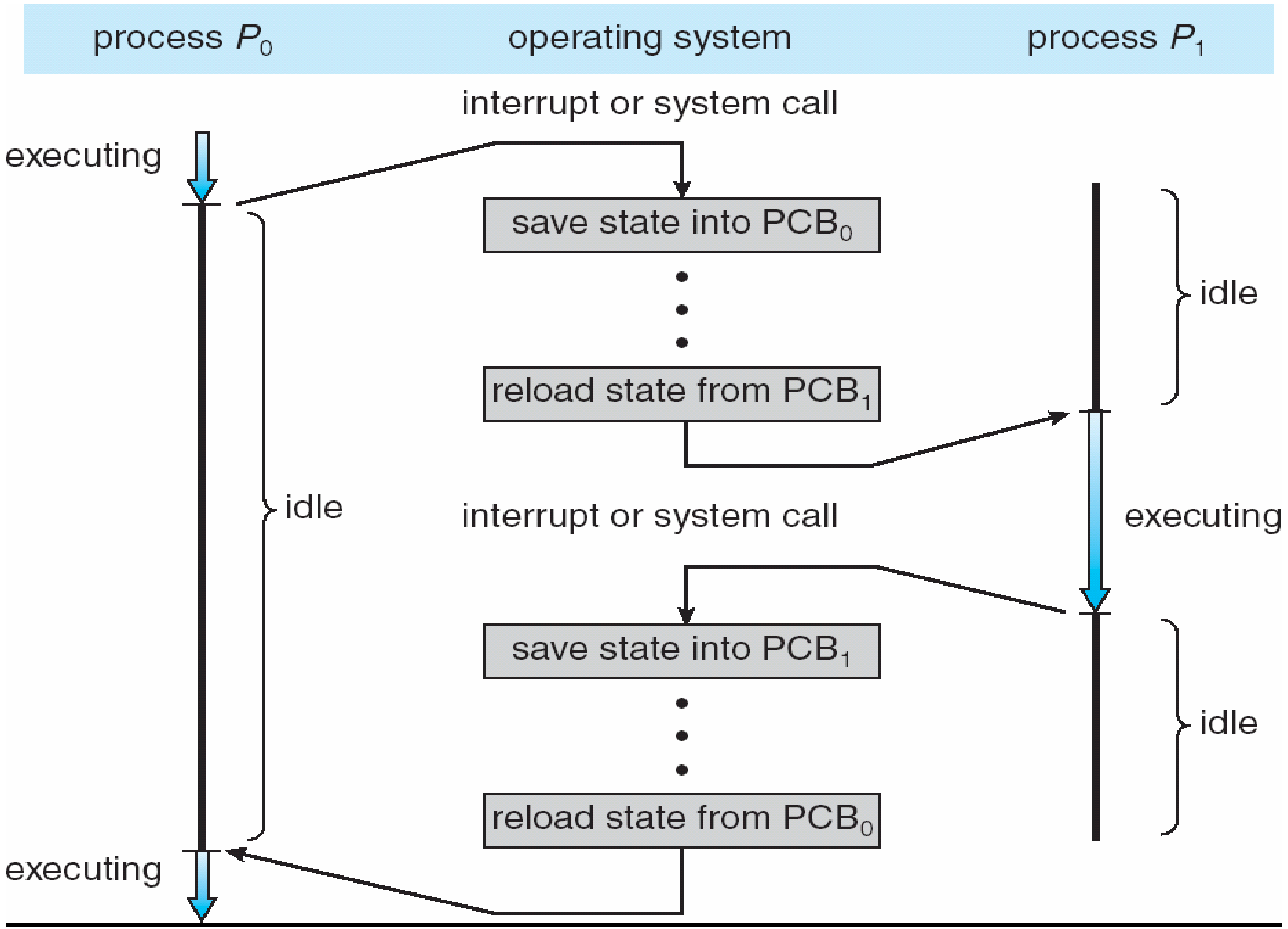
3. Update PCB:
The PCB of the current process is updated to reflect its new state (e.g., moved to the waiting queue).

4. Scheduler Selection:
The OS scheduler selects the next process to run from the ready queue.

5. Load Context:
The saved context of the selected process is loaded from its PCB into the CPU's registers.

6. Resume Execution:
The new process begins execution from the exact point where it last left off.

Cost of Context Switching
While essential for modern operating systems, context switching is not free. The process of saving and restoring contexts consumes CPU time and resources, which adds to the system's overhead.


process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
idle
reload state from PCB_1
idle
interrupt or system call
executing
save state into PCB_1
idle
reload state from PCB_0
executing

Interprocess Communication

▪Interprocess Communication (IPC) is a set of mechanisms provided by an operating system to allow multiple, independent processes to communicate and synchronize with each other, sharing data and coordinating activities to achieve a common goal

•IPC methods include shared memory (processes access a common memory region), message passing (processes exchange messages through the kernel),

Independent Process	Co-operating Process
1. A process is **independent** if it cannot affect or be affected by the other processes executing in the system	1. A process is **co-operating** if it can affect or be affected by the other processes executing in the system.
2. Any process that does not share data with any other process is IP	2. Any process that share data with any other process is CP

Interprocess Communication

Cooperating processes need **inter process communication** (**IPC**)

Two models of IPC:

I. **Shared memory**

II. **Message passing**

Shared Memory Model:

1. Establish Shared Memory:

A process requests a shared memory segment from the operating system, which the OS creates and allocates. This segment is identified by a unique segment ID.

2. Attach to Segment:

Other processes, upon receiving the segment ID, attach this shared memory region to their own address spaces.

3. Data Exchange:

Once attached, all processes can read from and write to the same memory region. For instance, a producer process can write data to the shared buffer, and a consumer process can read it.

4. Detach and Remove:

When communication is finished, a process detaches from the segment, and the shared memory segment can be removed from the system.

Benefits

High Performance:
Data transfer is very fast because there are no complex data exchange procedures, just direct memory access.

Reduced Memory Usage:
It avoids creating redundant copies of data by directing all processes to a single instance.

Drawbacks

Synchronization Complexity:
Developers must implement and manage synchronization to prevent race conditions and ensure data integrity.

Memory Protection:
The operating system needs to relax its normal memory protection, allowing processes to access memory outside their usual private address space.

Message Passing Model:

A message passing model is a method of inter-process communication (IPC) where processes exchange data by sending and receiving messages through a communication link, often via a message queue, without sharing direct memory.

1. Establish Connection:
Processes establish a communication link.
2. Send Operation:
A process uses a send() system call to send a message to a specific recipient or a message queue.
3. Message Transfer:
The message is copied from the sending process's memory to a kernel-managed message queue or buffer.
4. Receive Operation:
The receiving process uses a receive() call to get the message from the queue.
5. Process Message:
The receiving process then processes the message and takes action based on its contents.

Key Advantages

Isolation and Fault Tolerance:

Each process operates independently, with its own memory, enhancing fault tolerance because a problem in one process doesn't directly affect others.

Simplicity:

Memory management is hidden from the programmer, simplifying process communication and making the overall system easier to understand and implement compared to the shared memory model

No Synchronization Issues:

Because processes do not share memory, there is no need for explicit synchronization mechanisms to prevent race conditions or data corruption, which is a common source of bugs in shared memory systems,

Drawback

Slower Communication:

Message passing requires data to be copied between processes and transferred, which is slower than direct memory access, especially with large datasets.

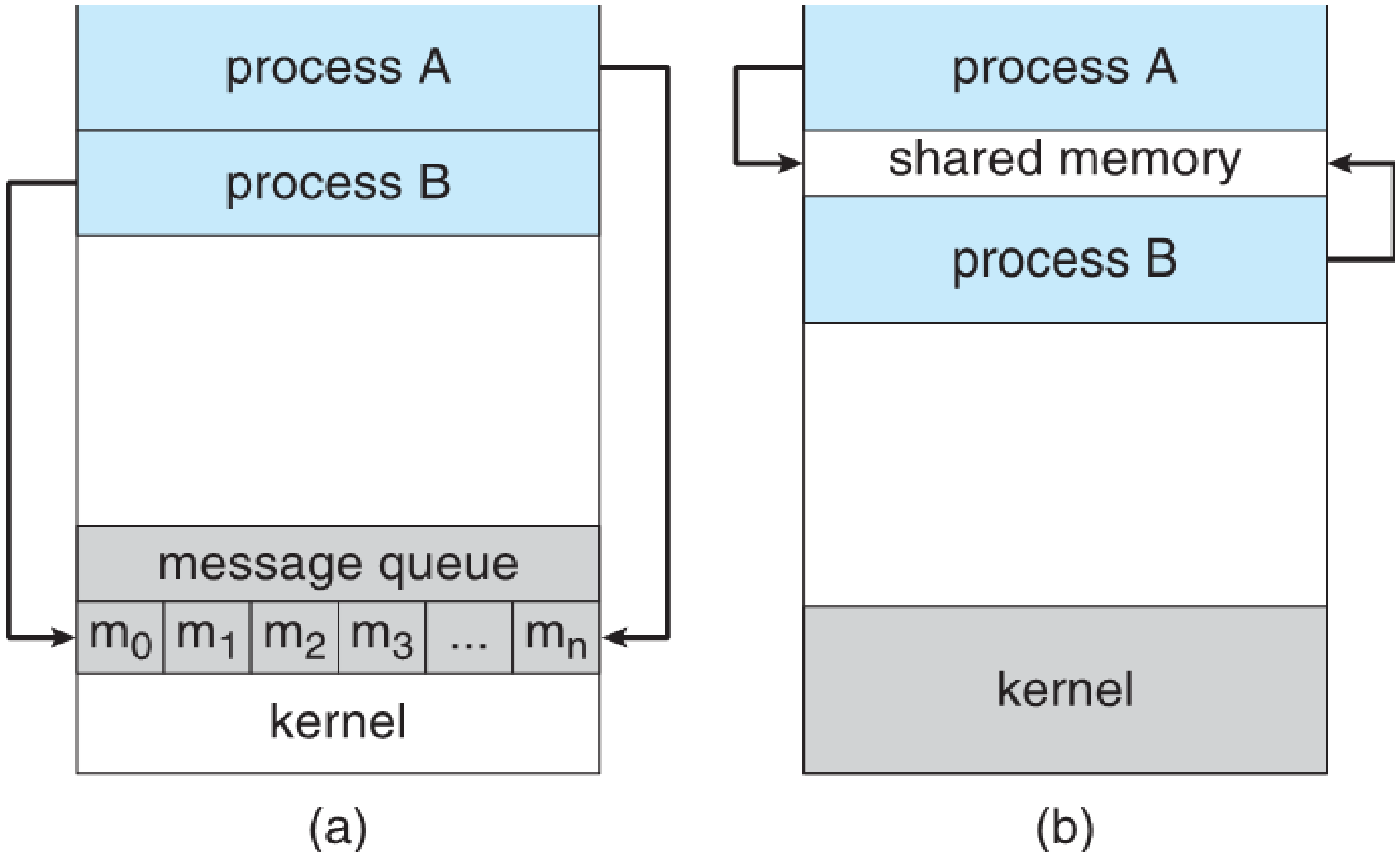
High Overhead:

There's significant overhead from kernel involvement in message routing and data conversion, as well as the setup and management of communication channels.

Resource Consumption:

Message transmission is costly, consuming more resources, particularly when large amounts of data are involved.

(a) Message passing. (b) shared memory.



THREADS

1. A Thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers .

2. It is also called as light weight process.

3. A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack.

4. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.
5. Threads run in parallel improving the application performance. Each such thread has its own CPU state and stack, but they share the address space of the process and the environment.

THREADS Overview

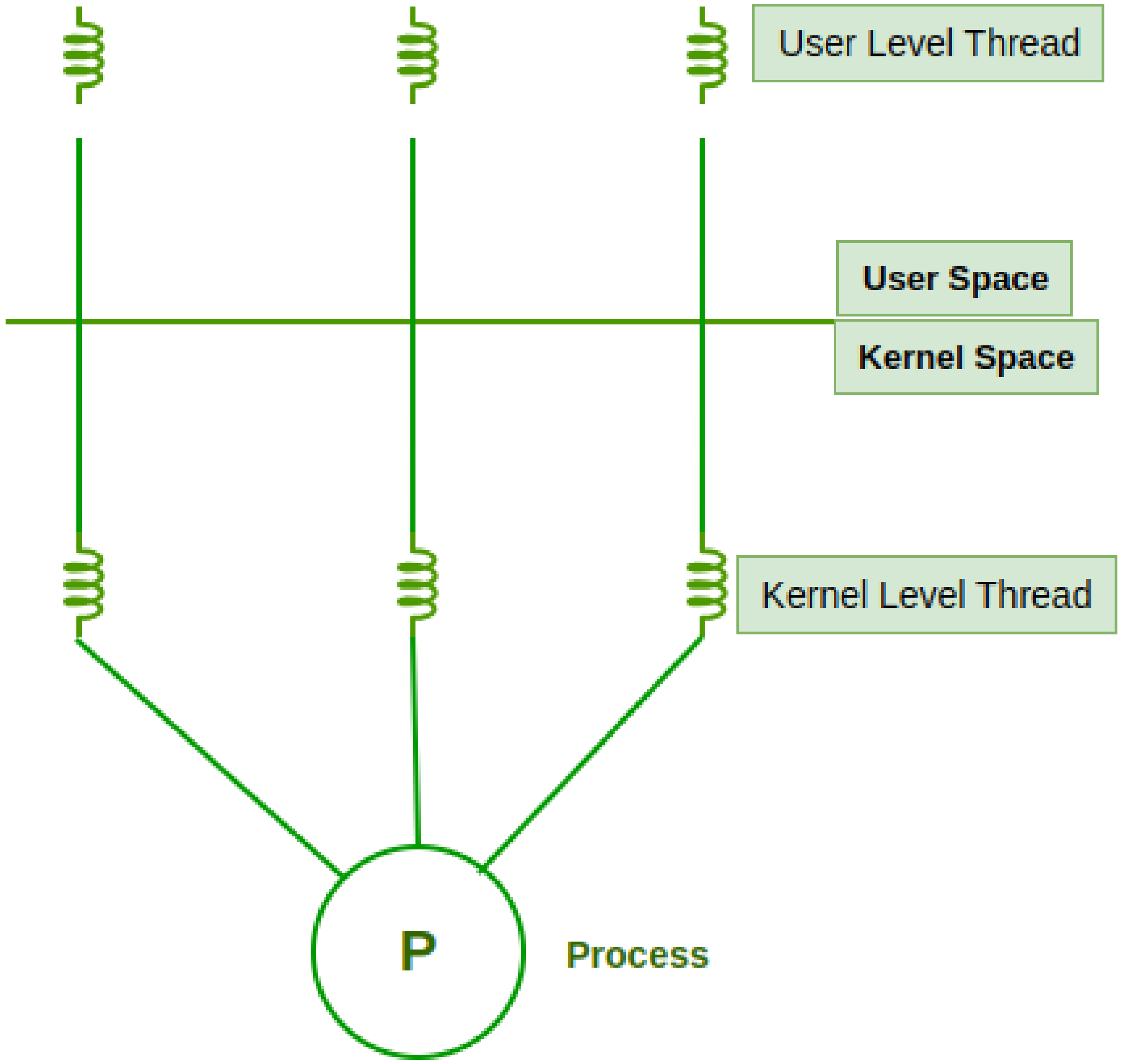
A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.

Components of Threads:

- ❖Stack Space
- ❖Register Set
- ❖Program Counter

Types of Thread in Operating System

- ➢User Level Thread
- ➢Kernel Level Thread


User Level Thread
User Space
Kernel Space
Kernel Level Thread
P
Process

User Level Threads

User Level Thread is a type of thread that is not created using system calls. The kernel has no work in the management of user-level threads. User-level threads can be easily implemented by the user. In case when user-level threads are single-handed processes, kernel-level thread manages them.

Advantages of User-Level Threads:

- Implementation of the User-Level Thread is easier than Kernel Level Thread.
- Context Switch Time is less in User Level Thread.
- User-Level Thread is more efficient than Kernel-Level Thread.
- Because of the presence of only Program Counter, Register Set, and Stack Space, it has a simple representation.

Disadvantages of User-Level Threads

- There is a lack of coordination between Thread and Kernel.
- Inc case of a page fault, the whole process can be blocked.

Kernel Level Threads

A kernel Level Thread is a type of thread that can recognize the Operating system easily. Kernel Level Threads has its own thread table where it keeps track of the system. The operating System Kernel helps in managing threads. Kernel Threads have somehow longer context switching time. Kernel helps in the management of threads.

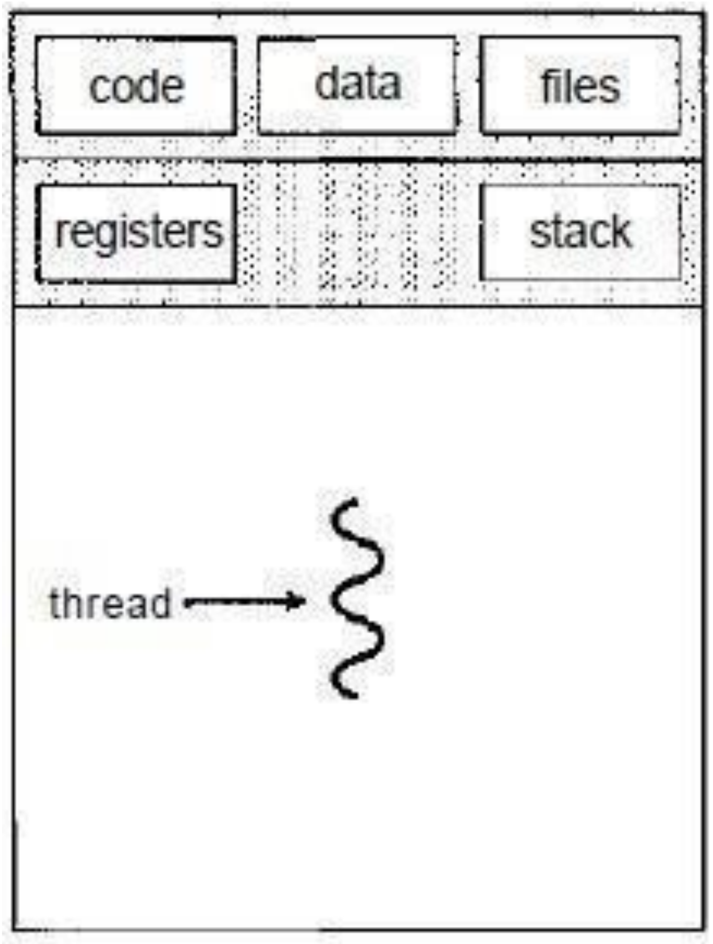
Advantages of Kernel-Level Threads

- It has up-to-date information on all threads.
- Applications that block frequency are to be handled by the Kernel-Level Threads.
- Whenever any process requires more time to process, Kernel-Level Thread provides more time to it.

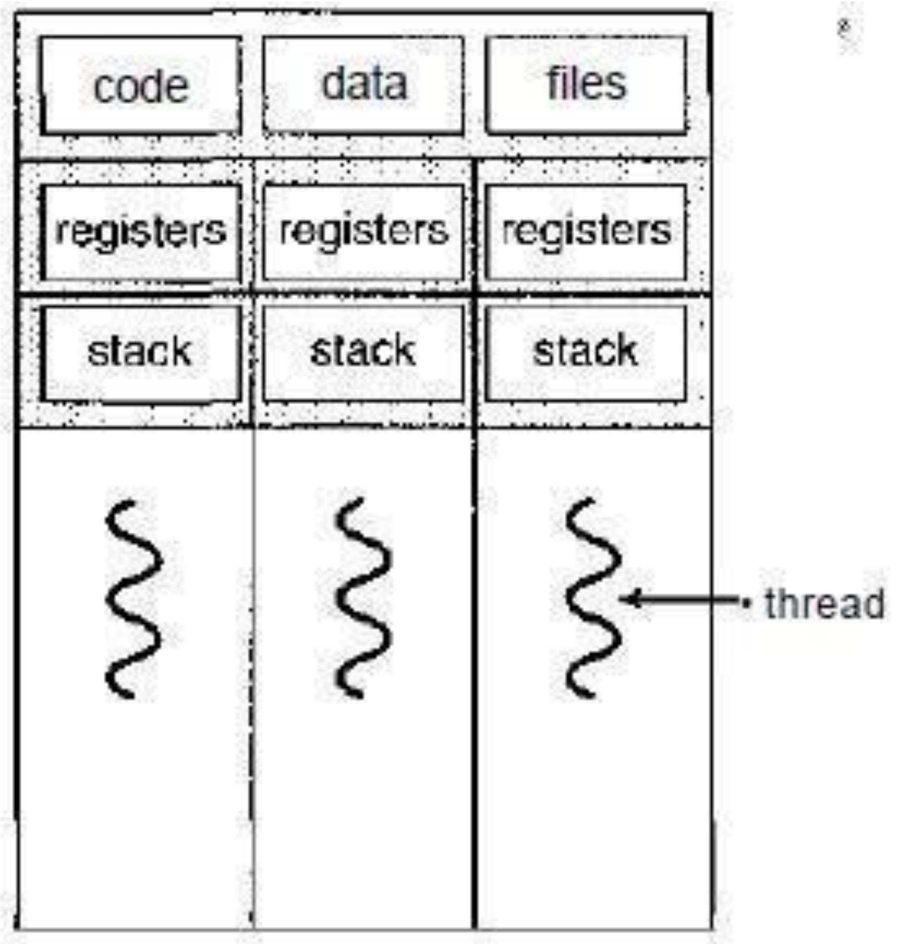
Disadvantages of Kernel-Level threads

- Kernel-Level Thread is slower than User-Level Thread.
- Implementation of this type of thread is a little more complex than a user-level thread.

Thread



single-threaded process



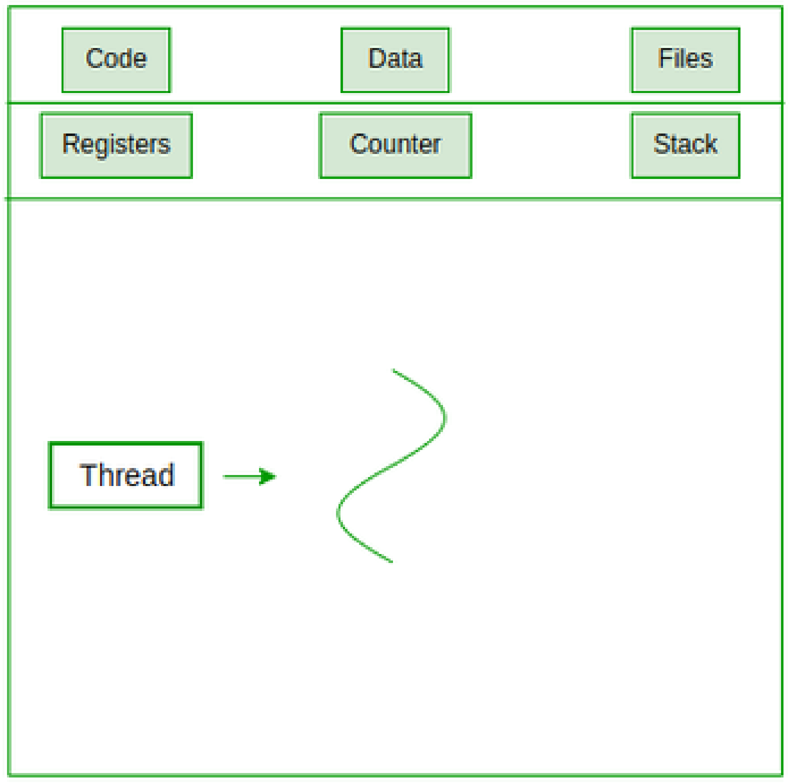
multithreaded process

What is Multi-Threading?

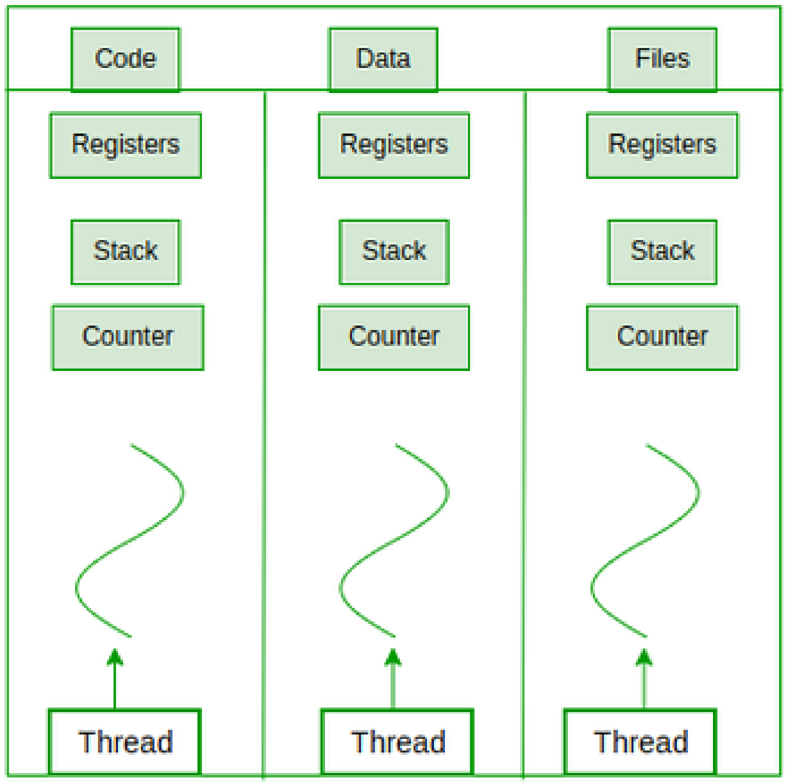
A thread is also known as a lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc.

Multithreading is a technique used in operating systems to improve the performance and responsiveness of computer systems. Multithreading allows multiple threads (i.e., lightweight processes) to share the same resources of a single process, such as the CPU, memory, and I/O devices.

Multithreading Models



Single Threaded Process



Multi Threaded Process

Advantages of Thread in Operating System

Responsiveness*:* If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

Faster context switch*:* Context switch time between threads is lower compared to the process context switch. Process context switching requires more overhead from the CPU.

Effective utilization of multiprocessor system*:* If we have multiple threads in a single process, then we can schedule multiple threads on multiple processors. This will make process execution faster.

Resource sharing*:* Resources like code, data, and files can be shared among all threads within a process. Note: Stacks and registers can't be shared among the threads. Each thread has its own stack and registers.

Communication*:* Communication between multiple threads is easier, as the threads share a common address space. while in the process we have to follow some specific communication techniques for communication between the two processes.

Enhanced throughput of the system*:* If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.

Difference between process and thread

Process	Thread
1.Process is heavy weight	1.Light weight
2. It needs interaction with OS	2.Does not required with OS
3. It creates own memory of files	3.it share same set of open files,diff processes
4.Multiple processes without using Threads use more resources	4.multiple threaded processes are fewer resoruces
5. It is an independent	4.one thread having read, write or change another thread