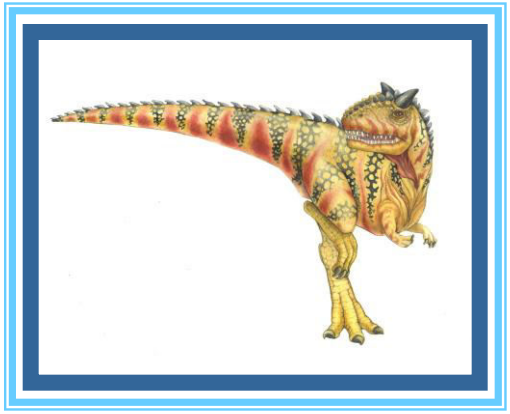
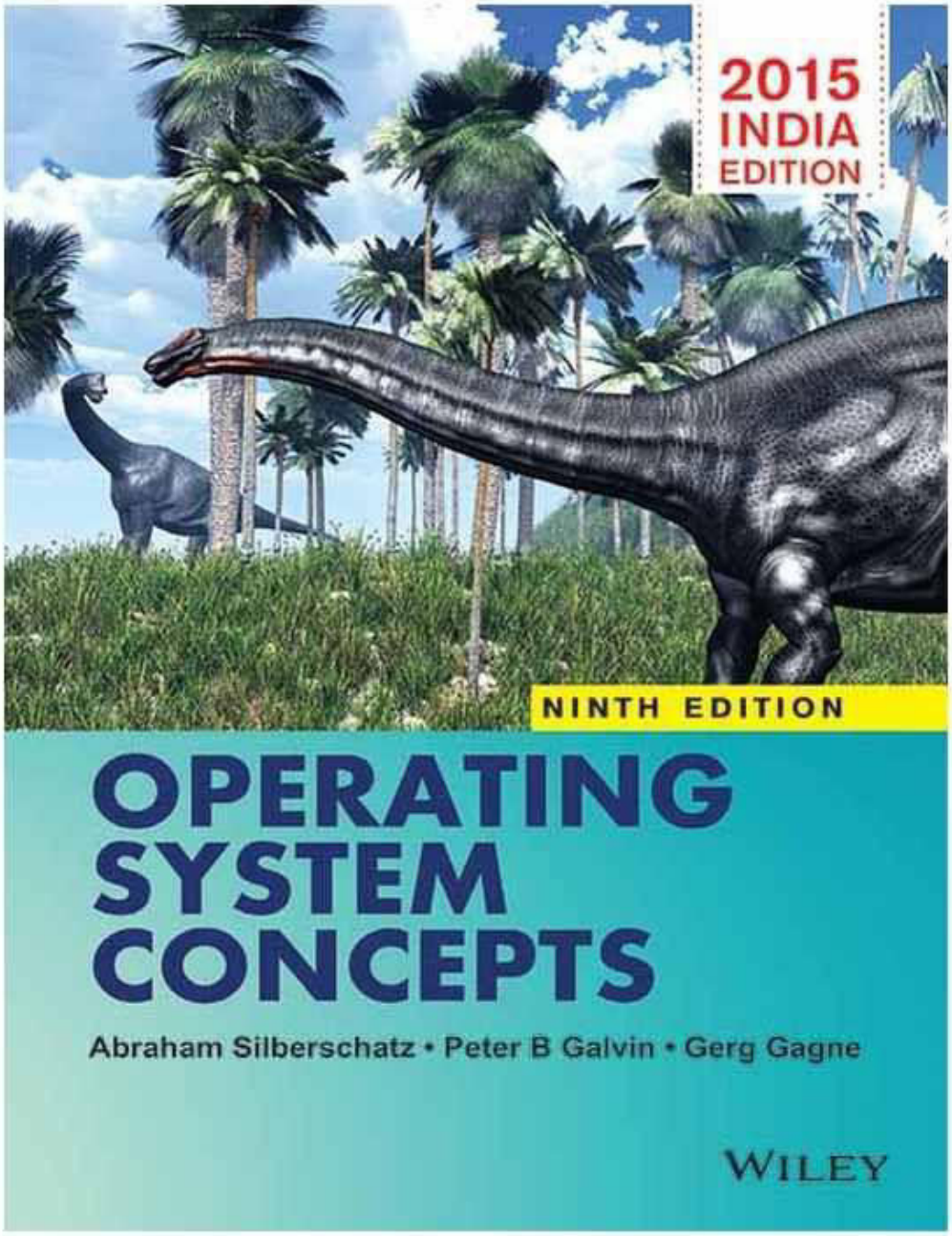
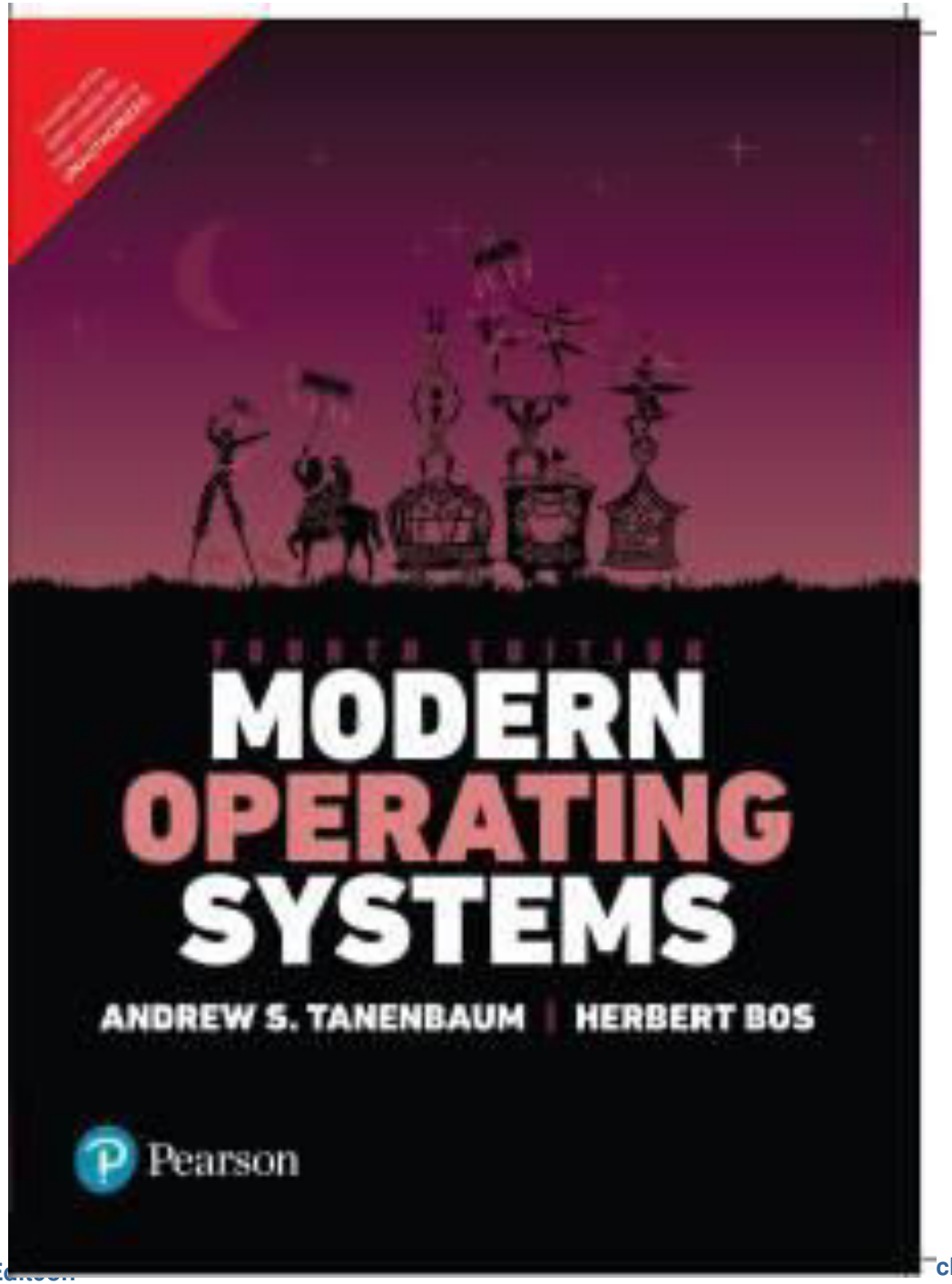
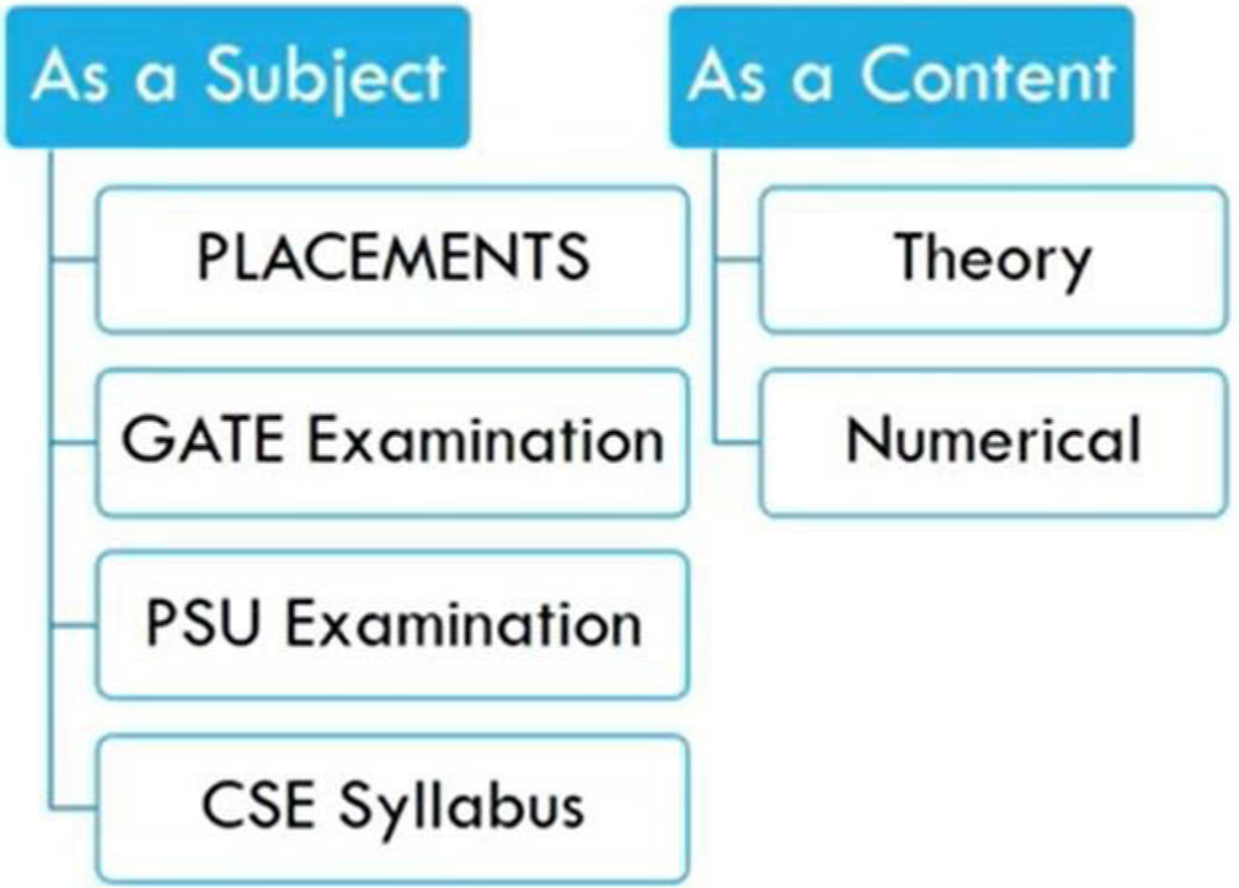
Introduction





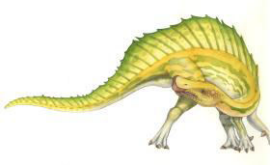

FOURTH EDITION
MODERN
OPERATING
SYSTEMS
ANDREW S. TANENBAUM
HERBERT BOS
Pearson



Introduction

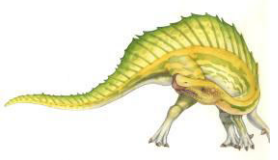
- ***System software:*** *System software refers to a collection of programs that manage and control the computer hardware, facilitate communication between hardware components, and provide a platform for running application software. It acts as an intermediary between the user and the hardware, enabling the proper functioning of the computer system. Like: os, device drivers, utility tools and firmware.*

- ***Application software:*** *Meant to enable the user to carry out some specific set of tasks or functions. Like web browser, Microsoft excel, Microsoft word.*

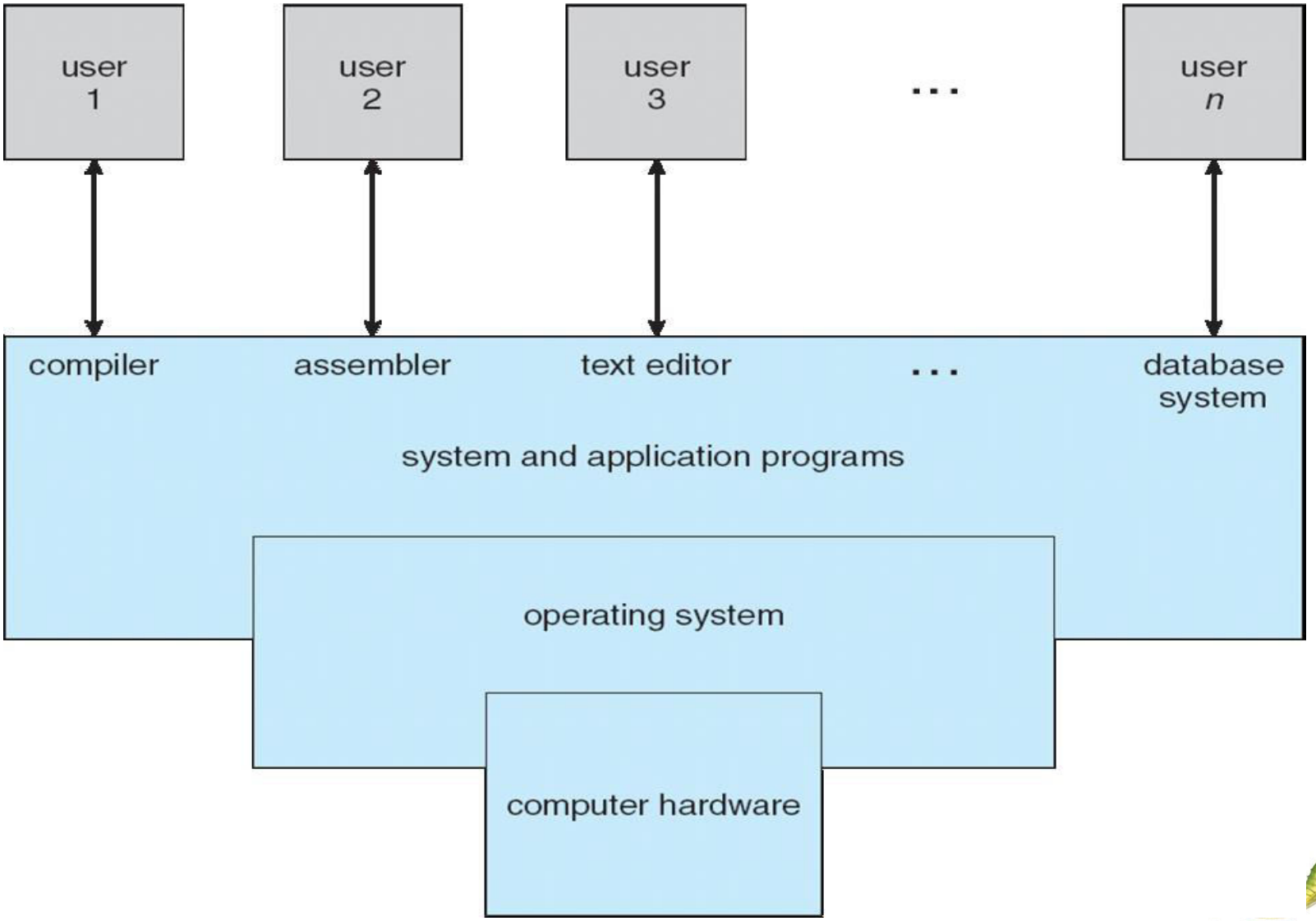


What is an Operating System?

*Operating system is a **system software** that act as an **intermediary** between the **hardware and user-level software**, making it possible for applications to interact with the computer's hardware components **without needing** to know the specific details of the underlying hardware.*

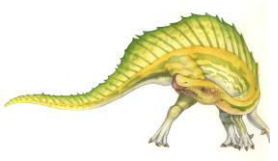


Four Components of a Computer System



Abstract Views of an Operating System

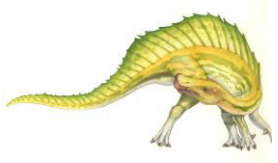
- *Computer system can be divided into **four components**:*
  - ***Hardware:** Provides basic computing resources for the system: CPU, memory, I/O devices*
  - ***Operating system:** Controls and coordinates use of hardware among various applications and users*
  - ***Application programs:** Define the ways in which the system resources are used to solve the computing problems of the users*
    - *Word processors, compilers, web browsers, database systems, video games*
  - ***Users:** People, machines, other computers*



Operating System's Goals

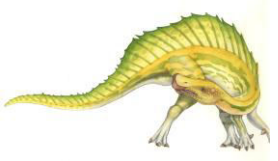
Convenience + Efficiency

- *Make the computer system convenient to use.*
- *Ensures use of the computer hardware in an efficient manner.*
- ***Primary Goal:*** ***Convenience***
- ***Secondary Goal:*** ***Efficiency***



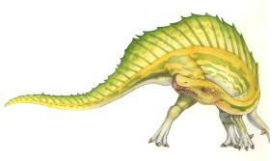
Operating System

- *Some operating system are designed to be **convenient** (personal computer and android), others to be **efficient** (mainframe, supercomputer) and some are combination of both.*



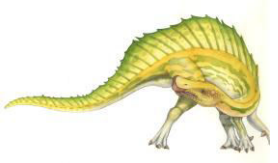
Operating System

- *OS is a **resource allocator/resource manager.***
  - *Manages all resources, in unbiased fashion so that it can operate the computer system fairly and efficiently.*
  - *Decides between numerous and conflicting requests for efficient and fair resource use.*
- *OS is a **control program.***
  - *Manages the execution of user programs to prevent errors and improper use of the computer.*



Functionalities of Operating System

- Resource Management
- Abstraction: (*By hiding the complex details of hardware from application developers.*)
- Processor Management/Process Management
- I/O Management/Device Management
- Storage Management/File management
- Memory Management
- Security & Privacy
- Job Accounting
- User Interface

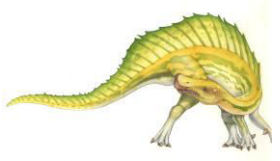


Functionalities of Operating System

- ***Resource Manager:*** Operating System manages all the Resources those are attached to the system like memory, processor and all the Input output Devices.

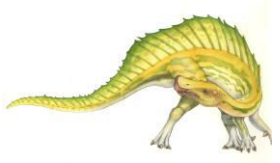
- ***Process Management :*** The Operating System also Treats the Process Management means all the Processes those are given by the user or the Process those are System ‘s own Process are Handled by the Operating System

- **Memory Management:** Operating System also Manages the Memory of the Computer System means Provide the Memory to the Process and Also Deallocate the Memory from the Process. And also defines that if a Process gets completed then this will deallocate the Memory from the Processes.



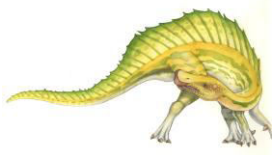
Functionalities of Operating System

- **Processor Management:** In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called process scheduling.
- **Storage Management:** Operating System also Controls the all the Storage Operations means how the data or files will be Stored into the computers and how the Files will be accessed by the users etc
- **Device Management:** An Operating System manages device communication via their respective drivers. It Keeps tracks of all devices.
  - Decides which process gets the device when and for how much time.
  - Allocates the device in the efficient way.
  - De-allocates devices.



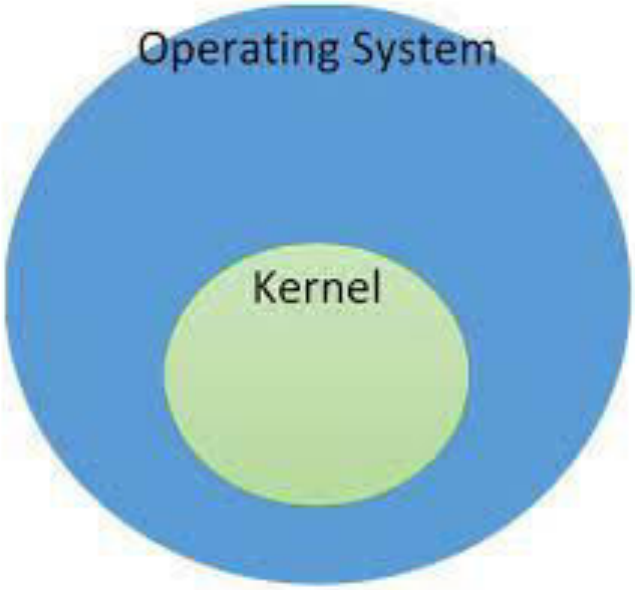
Functionalities of Operating System

- ***Security:*** *The operating system uses password protection to protect user data and similar other techniques. it also prevents unauthorized access to programs and user data.*
- ***Job accounting:*** *It is function available on every i5/OS™ system that can be used to track usage of system resources.*
- ***Error detecting aids:*** *The operating system constantly needs to be aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time). For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.*

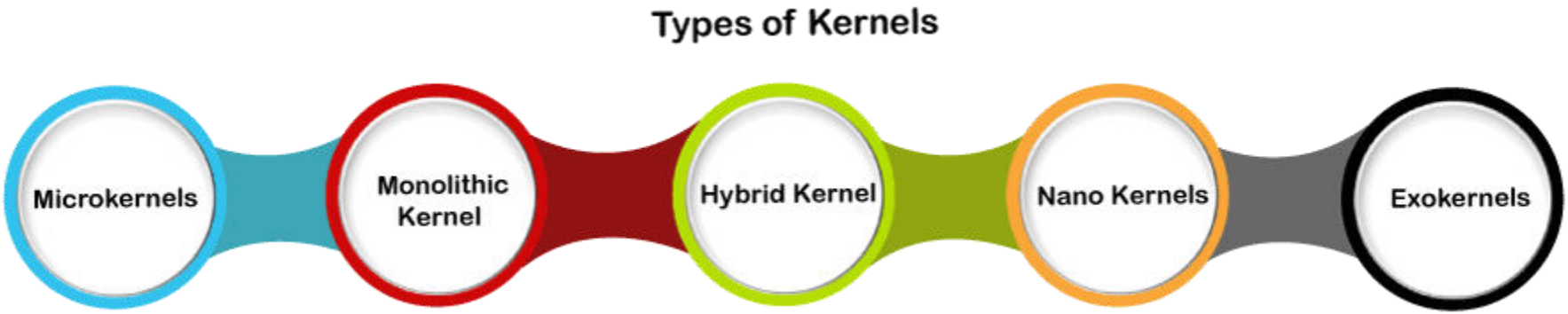


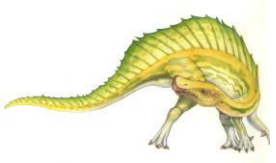
Parts of Operating System

- *The one program running at all times on the computer" is the **kernel**.*
- *Kernel **loads first** into memory when an operating system is loaded and remains into memory until operating system is shut down again.*



Parts of Operating System

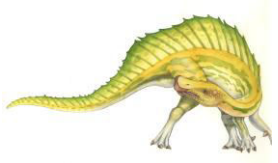




Monolithic Kernel

- *In a monolithic kernel, the same memory space is used to implement user services and kernel services.*
- *The execution of processes is also faster than other kernel types as it does not use separate user and kernel space.*
- *If any service generates any error, it may crash down the whole system.*

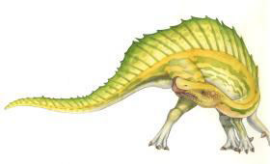
Examples : Unix, Linux, Open VMS, XTS-400, etc.



Micro Kernel

- *In this, user services and kernel services are implemented into two different address spaces: user space and kernel space.*
- *Since it uses different spaces for both the services, so, the size of the microkernel is decreased, and which also reduces the size of the OS.*

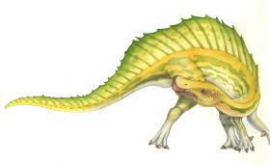
Examples of Microkernel are L4, AmigaOS, Minix, K42, etc



Hybrid Kernel

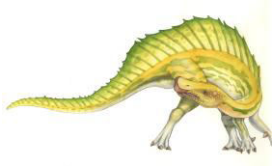
- *A hybrid kernel can be understood as the extended version of a microkernel with additional properties of a monolithic kernel.*
- *These kernels are widely used in commercial OS, such as different versions of MS Windows.*

Examples: Windows NT, Netware, BeOS, etc.



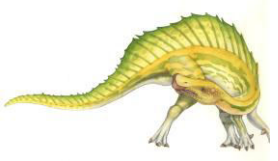
Nano Kernel

- *As the name suggests, in Nanokernel, the complete code of the kernel is very small, which means the code executing in the privileged mode of the hardware is very small.*
- *Examples: EROS etc.*



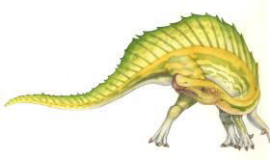
Dual-mode and Multimode of OS

- *The dual-mode operation provides a mechanism for enforcing security, protecting system resources, and ensuring stability. Let's understand each mode:*
- *User Mode*
- *Kernel Mode*



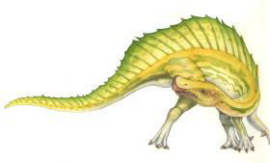
User Mode

- *The processor executes user-level code, which includes applications and processes running on the system.*
- *User mode* ***restricts direct access to hardware***
- *User mode provides a limited set of instructions and* ***system calls*** *that allow user-level programs to request services from the kernel and perform specific tasks through well-defined interfaces.*



Kernel Mode

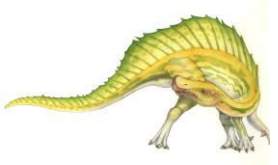
- *In kernel mode, the processor executes* ***kernel-level code,*** *which includes the core components of the operating system, device drivers, and critical system services.*
- *Only code running in kernel mode can directly interact with hardware, access certainprivileged instructions, and perform sensitive tasks that require elevated privileges.*



Dual-mode and Multimode of OS

- *During the context switch, the processor changes its privilege level from user mode to kernel mode, allowing the kernel to execute the requested service on behalf of the user process. After the service is completed, the context switch occurs again, returning the processor to user mode, and the user process continues its execution.*

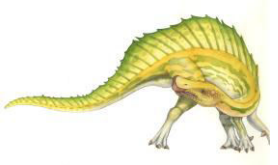
- *This dual-mode operation ensures that the operating system maintains control over system resources, protects them from unauthorized access or interference, and provides a secure and stable environment for user-level applications to run.*



Dual-mode and Multimode of OS

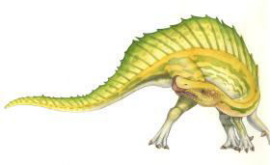
Privileged instructions: *That may cause harm. The hardware allows privileged instructions to be executed only in kernel mode.*

- *I/O instructions and Halt instructions*
- *Turn off all Interrupts*
- *Set the Timer*
- *Context Switching*
- *Clear the Memory or Remove a process from the Memory*
- *Modify entries in the Device-status table*

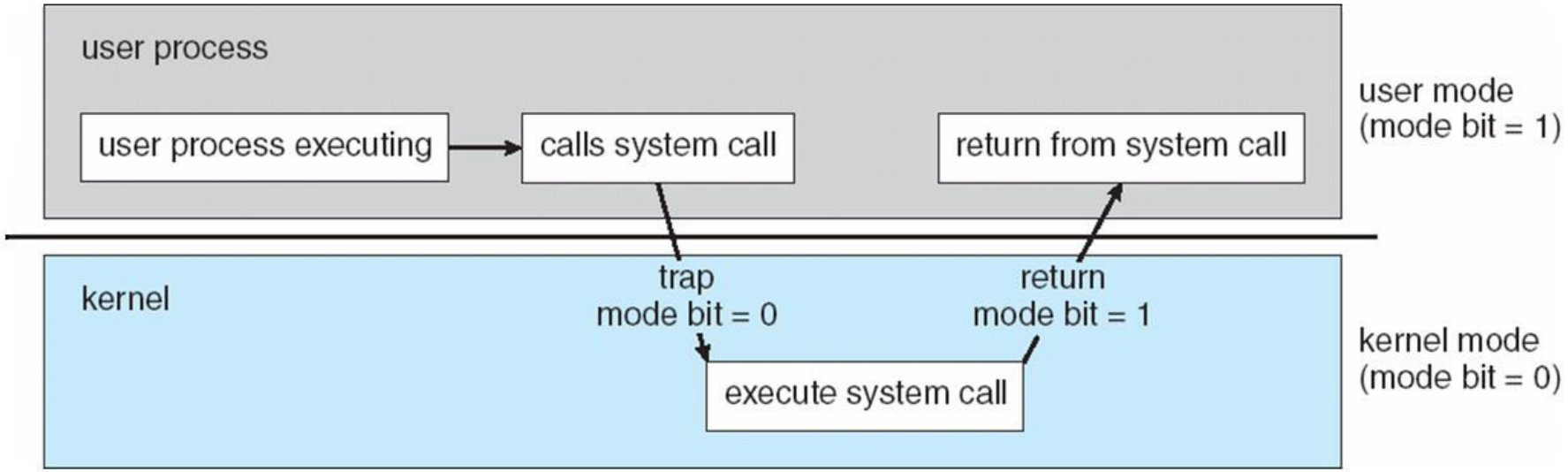


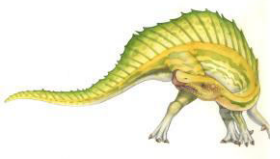
Dual-mode and Multimode of OS

- *A bit, called the **mode bit** is added to the hardware of the computer to indicate the current mode: **kernel (0) or user (1).***
- *With the mode bit, we are able to distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user.*

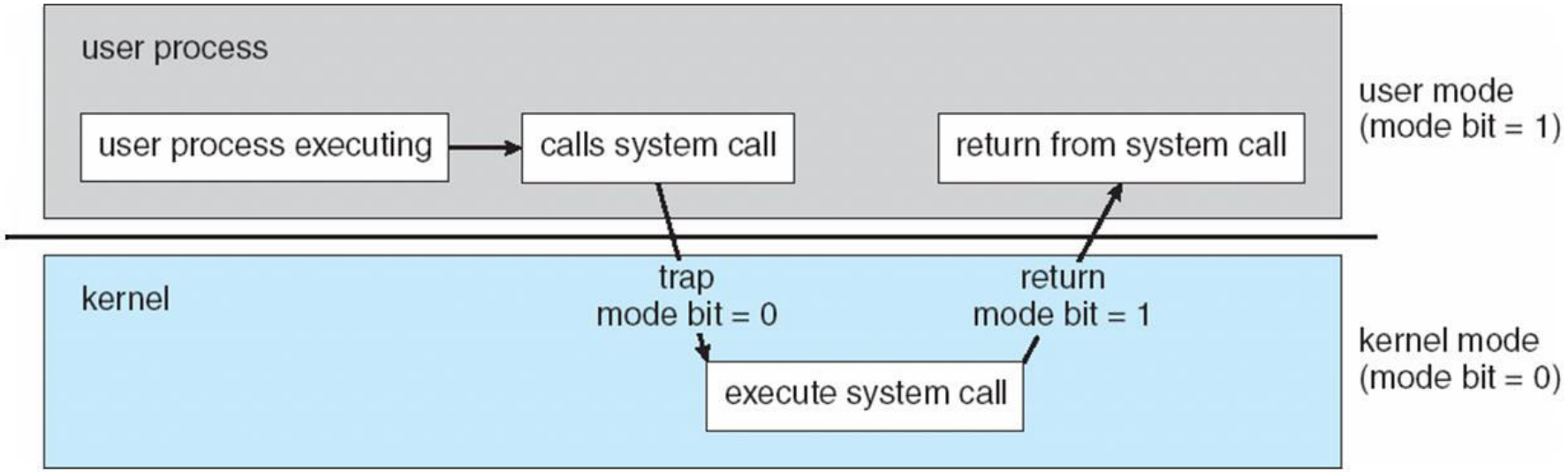


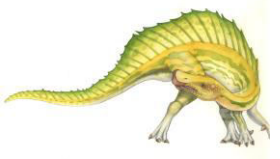
Transition from User to Kernel Mode





Transition from User to Kernel Mode

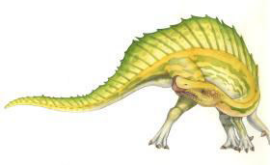




Dual-mode and Multimode of OS

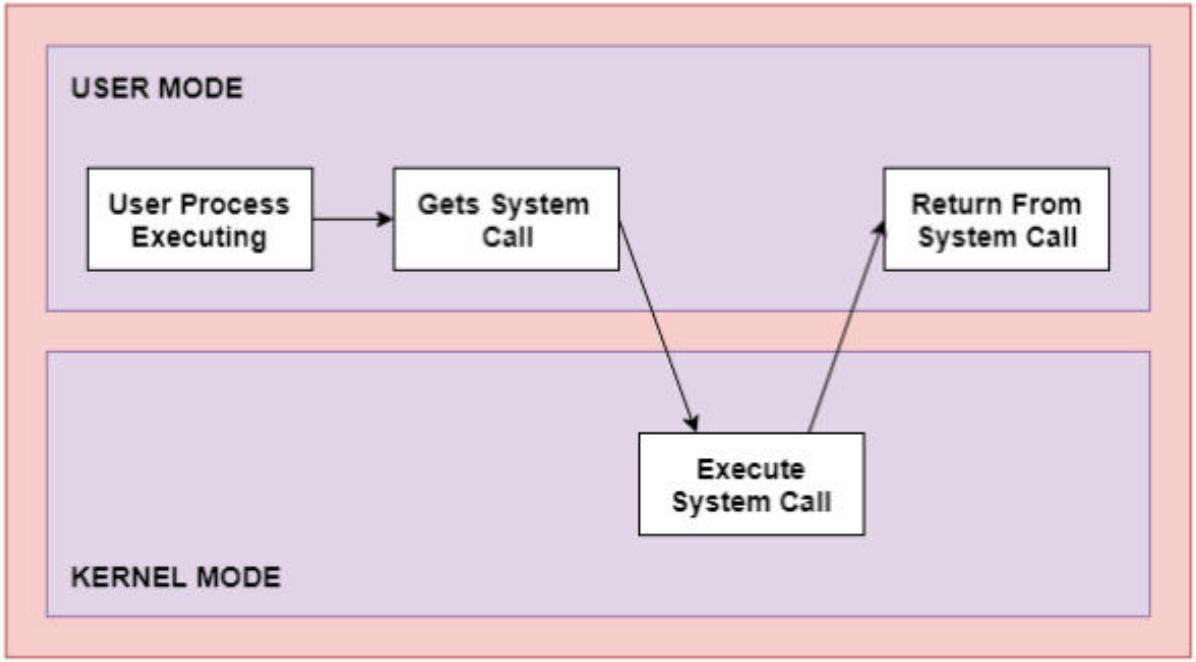
Various examples of Non-Privileged Instructions include:

- *Reading the status of Processor*
- *Reading the System Time*
- *Generate any Trap Instruction*
- *Sending the final printout of Printer*

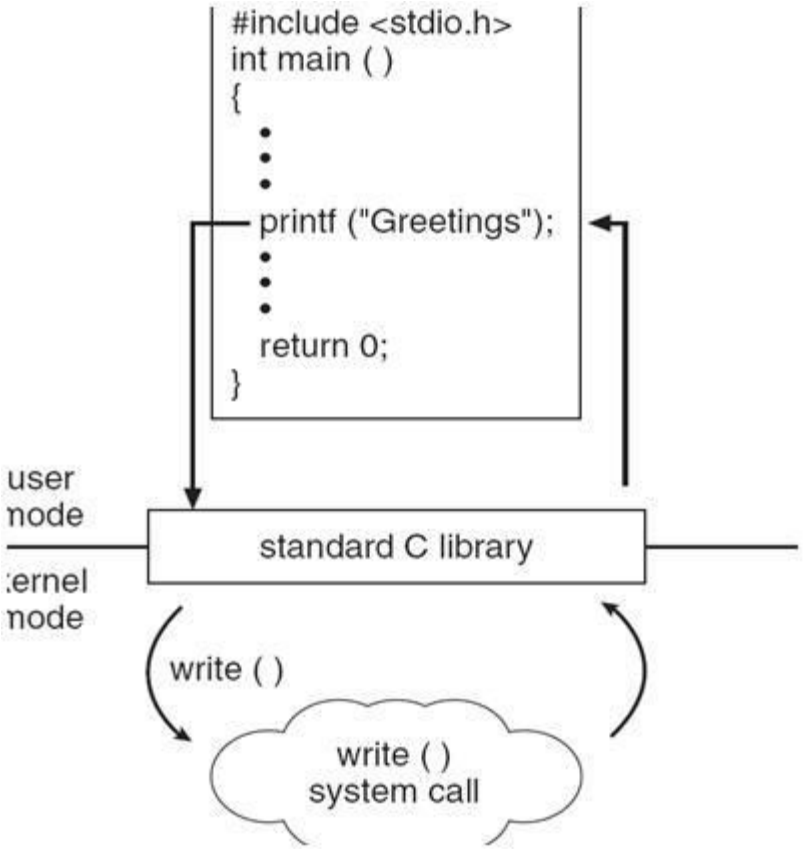


System Call

System call is the programmatic way in which computer program requests a services from the kernel of the operating system.

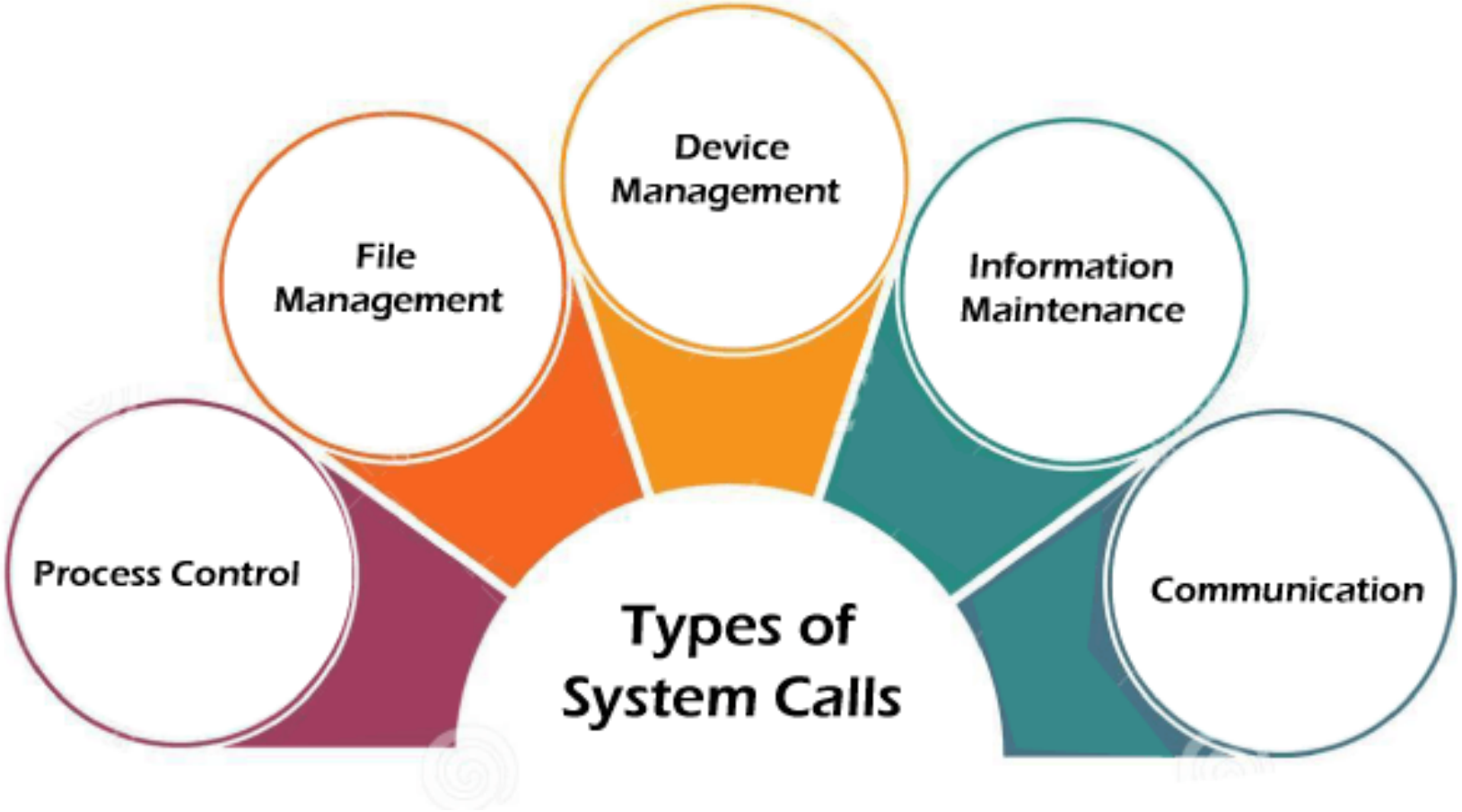


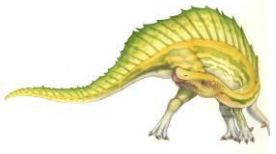
System Call



- *The standard **C library provides a portion of the system call interface** for many version of UNIX and Linux.*
- *As an example, let's assume a C program invokes **printf()** statement.*
- *The C library intercepts this call and **invokes the necessary system call(s)** in the OS.*
- *The C library takes the value returned by **write()** and passes it back to the user program*

System Call

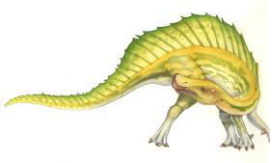

Device Management
File Management
Information Maintenance
Process Control
Communication
Types of System Calls



	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

fork() System Call

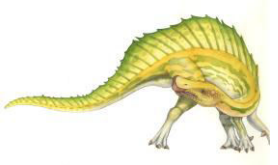
- *fork() is a system call in Unix-like operating systems that creates a new process, which is almost an **identical copy of the parent process** that called the fork() function.*
- *This new process is referred to as the child process, and the original process is referred to as the parent process.*



fork() System Call

When the fork() function is called, the following happens:

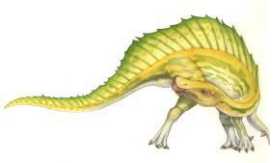
- *The parent process creates an **exact duplicate of itself**, including the **program counter**, memory, open file descriptors, and other attributes.*
- *Both the parent and child processes continue execution **from the point where the fork() function was called**.*
- *The operating system schedules these two processes independently, so **they can run concurrently**.*



fork() System Call

The fork() function returns different values in the parent and child processes:

- ***Negative Value:*** *The creation of a child process was unsuccessful.*
- ***Zero:*** *Returned to the newly created child process.*
- ***Positive value:*** *Returned to parent or caller. The value contains the process ID of the newly created child process.*



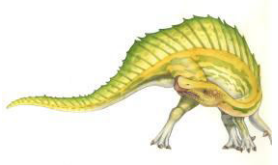
Example-1

```
int main()
{
        fork();
        printf("Hello world!\n");
        return 0;
}
```

Output:

Hello world!

Hello world!

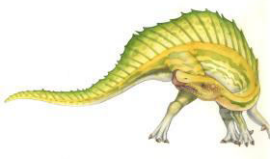


Example-1

```
int main()
{
    fork();
    fork();
    fork();
    printf("Hello world!\n");
    return 0;
}
```

Output:

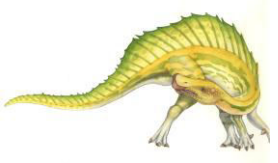
```
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
```



Example-1

```
int main()
{
    if (fork() == 0)
        printf("Hello from Child!\n");

     else
        printf("Hello from Parent!\n");
    return 0;
}
```

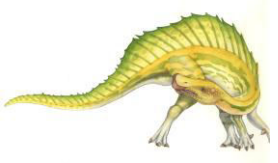


Example-1

```
int main() {
  if(fork() == 0){
    if(fork()){
      printf("Hello world!!\n");}}
  exit(0);
}
```

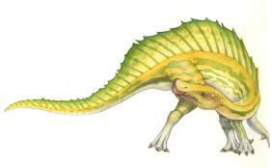
Output:

Hello world!!



Computer Startup

- ***Bootstrap program: The initial program that runs when a computer is powered up and rebooted.***
  - *Typically stored in ROM, generally known as **firmware.***
  - *It must know how to load OS and start executing that system.*
  - *To accomplish this goal, It must locate and Load into memory the operating system kernel and starts execution.*



Storage-Device Hierarchy