REAL TIME OPERATING SYSTEM

Chapter 6

Outline

- **Operating System basics**
- **Task Process and Threads**
- **Multiprocessing and Multitasking**
- **Task Scheduling**
- **Task Synchronization**
- **Device Drivers**

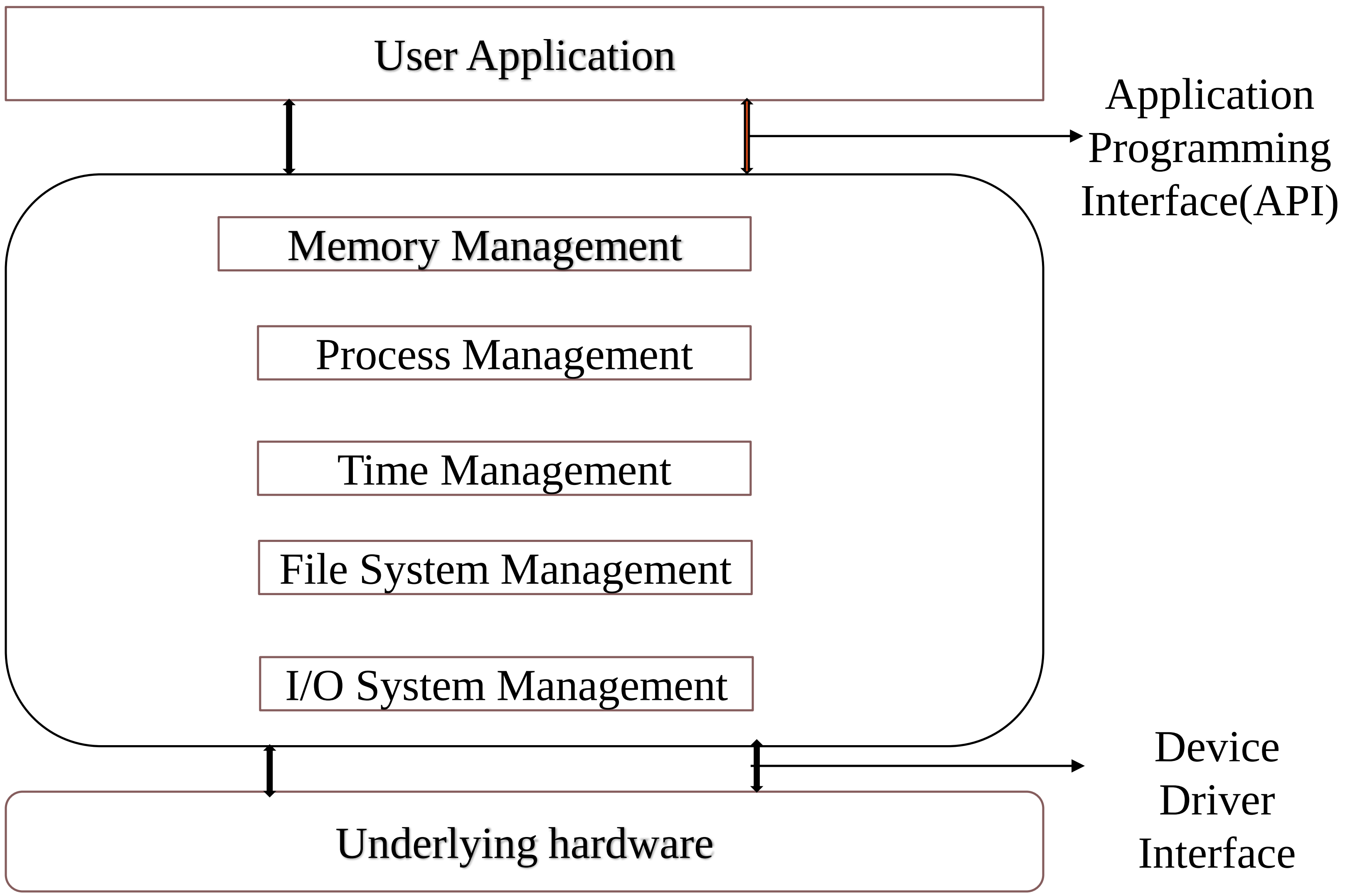
OPERATING SYSTEM BASICS

- acts as a bridge between the user application/ tasks & the underlying system resources through a set of system functionalities and services.
- Manages the system resources and makes them available to the user application/task on a need basis.

Primary Functions of OS

- manage the computer's resources, such as the central processing unit, memory, disk drives, and printers
- establish a user interface making user easy to use
- execute and provide services for applications software.

Architecture of Operating system



RTOS and GPOS

RTOS	GPOS
Deterministic and time sensitive Predictable execution pattern, respond within strictly defined time	Non-Deterministic and time insensitive, time to execute its service is not fixed
Task Scheduling is priority based, high priority process executes first	Uses fairness policy, no guarantee high priority thread will execute
Preemptive Kernel Low priority task will be preempted even if it is of system call	Non-Preemptive Kernel High-priority user thread can never preempt a kernel call
Generally used for special applications	Generally used for general purpose applications

Kernel

- Core of the operating system
- Responsible for managing the system resources and the communication among the hardware and other system services.
- Acts as the abstraction layer between system resources and user applications.
- Contains a set of system libraries and services.

Kernel Services

- Process Management
- Primary Memory Management
- File System Management
- I/O System (Device) Management
- Secondary Storage Management
- Protection Systems
- Interrupt Handler

Process Management

- Setting up the memory space for the process
- loading the process's code into the memory space
- allocating system resources
- scheduling and managing the execution of the process
- setting up and managing the process control block (PCB)
- Inter Process Communication and Synchronization
- process termination/deletion, etc.

Primary Memory Management

- Keeping track of which part of the memory area is currently used by which process
- Allocating and De- allocating memory space on a need basis (Dynamic memory allocation)

File System Management

- The creation, deletion and alteration of files and directories
- Saving of files in the secondary storage memory
- Providing automatic allocation of file space based on the amount of free space available
- Providing flexible naming convention for the files

I/O System (Device) Management

- Responsible for routing the I/O requests
- To provide access to disallowed devices through a set of Application Programming Interfaces (APIs)
- The kernel maintains a list of all I/O devices of the system.
- The list may be available in advance and recent kernel dynamically updates the list of available devices.
- The service 'Device Manager' of the kernel is responsible for handling all I/O device related operations.

Secondary Storage Management

- Deals with managing the secondary storage memory devices
- Disk storage allocation
- Disk scheduling (time interval at which the disk is activated to backup data)
- Free Disk space management

Protection Systems

- OS supports multiple users with different levels of access permissions (For example: Administrator, Standard, Restricted, Guest, etc)
- Implementing security policies to restrict the access to both user and system resources
- One user may not be allowed to view or modify the whole/portions of another user's data or profile details.
- Some application may not be granted with permission to make use of some of the system resources.

Interrupt Handler

- Kernel provides a mechanism to handle all external/internal interrupts generated by the system.
- Based upon the priority of the interrupt the process either runs in the foreground or background.

KERNEL SPACE AND USER SPACE

- OS dependent Partition: Kernel space and user space
- The memory space at which the kernel services related code is located is known as ‘Kernel Space’.
- Similarly, all user applications are loaded to a specific area of primary memory and this memory area is referred as ‘User Space’.
- User space is the memory area where user applications are loaded and executed.

TYPES OF KERNEL

- Monolithic Kernel
- Micro-Kernel

Monolithic Kernel

- All kernel services run in the kernel space.
- It runs all basic system services and provides powerful abstraction of the underlying hardware.
- Amount of context switches and messaging involved are greatly reduced which makes it run faster than microkernel.
- LINUX, SOLARIS, MS-DOS kernels are examples of monolithic kernel.

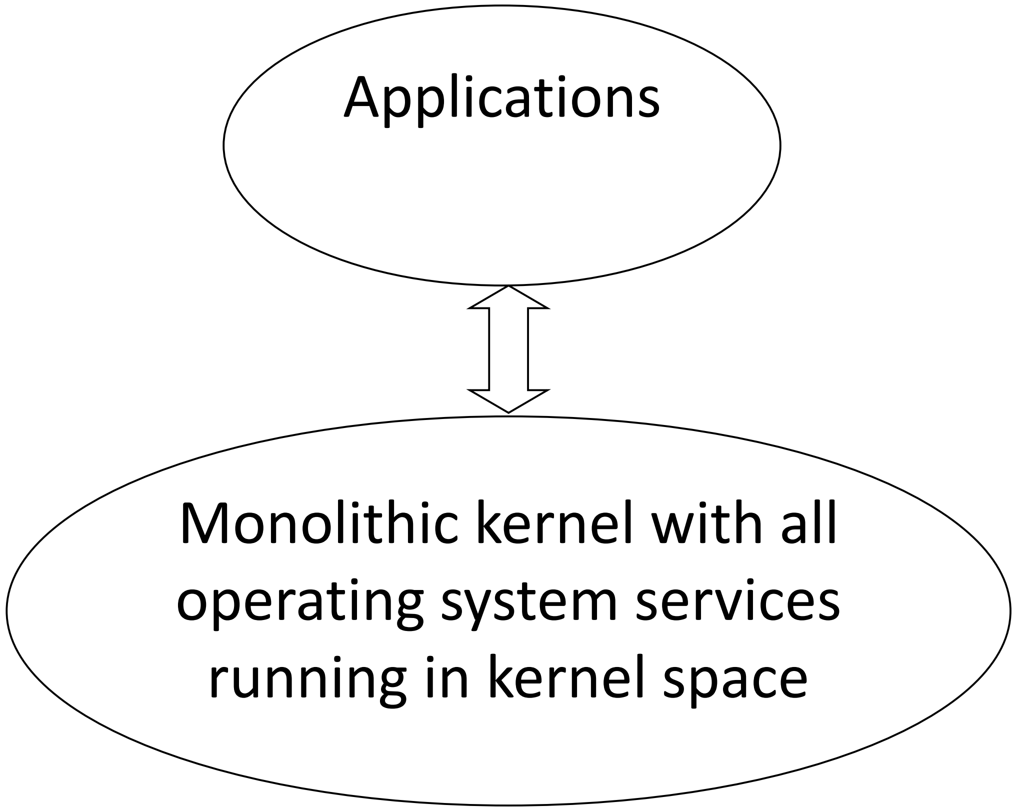

Applications
Monolithic kernel with all operating system services running in kernel space

Figure: The Monolithic Kernel Model

Drawbacks

- Error or failure in any one of the kernel modules leads to the crashing of the entire kernel application.
- The inclusion of all basic services in kernel space leads to different drawbacks such as requirement of large kernel size, lacking extensibility, poor maintainability.

Microkernel

- Essential set of operating system services such as communication and I/O control into the kernel.
- The rest of the operating system services are implemented in programs known as 'Servers' which runs in user space.
- It is more stable than monolithic as the kernel is unaffected even if the server fails.
- Memory Management, process Management, timer systems and interrupt handlers are the essential services, which forms the part of microkernel.

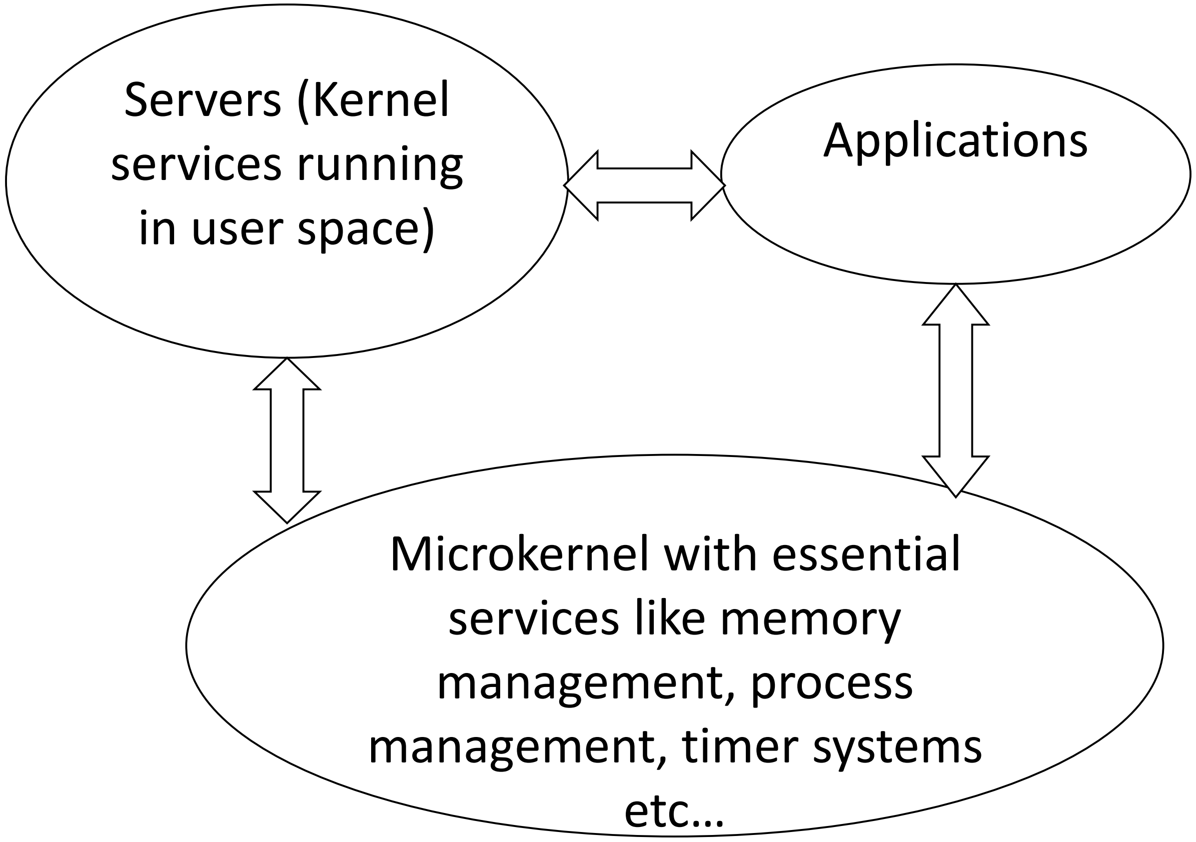

Servers (Kernel services running in user space)
Applications
Microkernel with essential services like memory management, process management, timer systems etc…

Figure: The Microkernel Model

Benefits

- **Robustness**: If a problem is encountered in any of the service, which runs as 'Server' application, the same can be reconfigured and re-stated without the need for re-starting the entire OS.
- **Configurability**: services can be changed, updated without corrupting the essential services residing within the microkernel.

TASKS, PROCESS AND THREADS

- A task is defined as a program in execution and related information maintained by OS for that program.
- Task is also known as 'Job' in the operating system context.
- A program or part of it in execution is also called a 'Process'.
- The terms 'Task', 'Job' and 'Process' refer to the same entity in the operating system context and most often they are used interchangeably.

PROCESS

- An instance of a program or part of program in execution
- A process requires various system resources such as the CPU for executing the process, memory for storing the code corresponding to the process and associated variables, I/O devices for information exchange etc.
- A process is an active entity
- A program becomes a process when an executable file is loaded into memory.

Structure of a process

- A process holds a set of registers, process status, Program Counter (PC), stack.
- From a memory perspective, the memory occupied by the process is separated into three regions, stack memory, data memory and code memory.
- The stack memory holds all temporary data such as variables local to the process.
- Data memory holds all global data for the process.
- The code memory contains the program code (instructions) corresponding to the process.

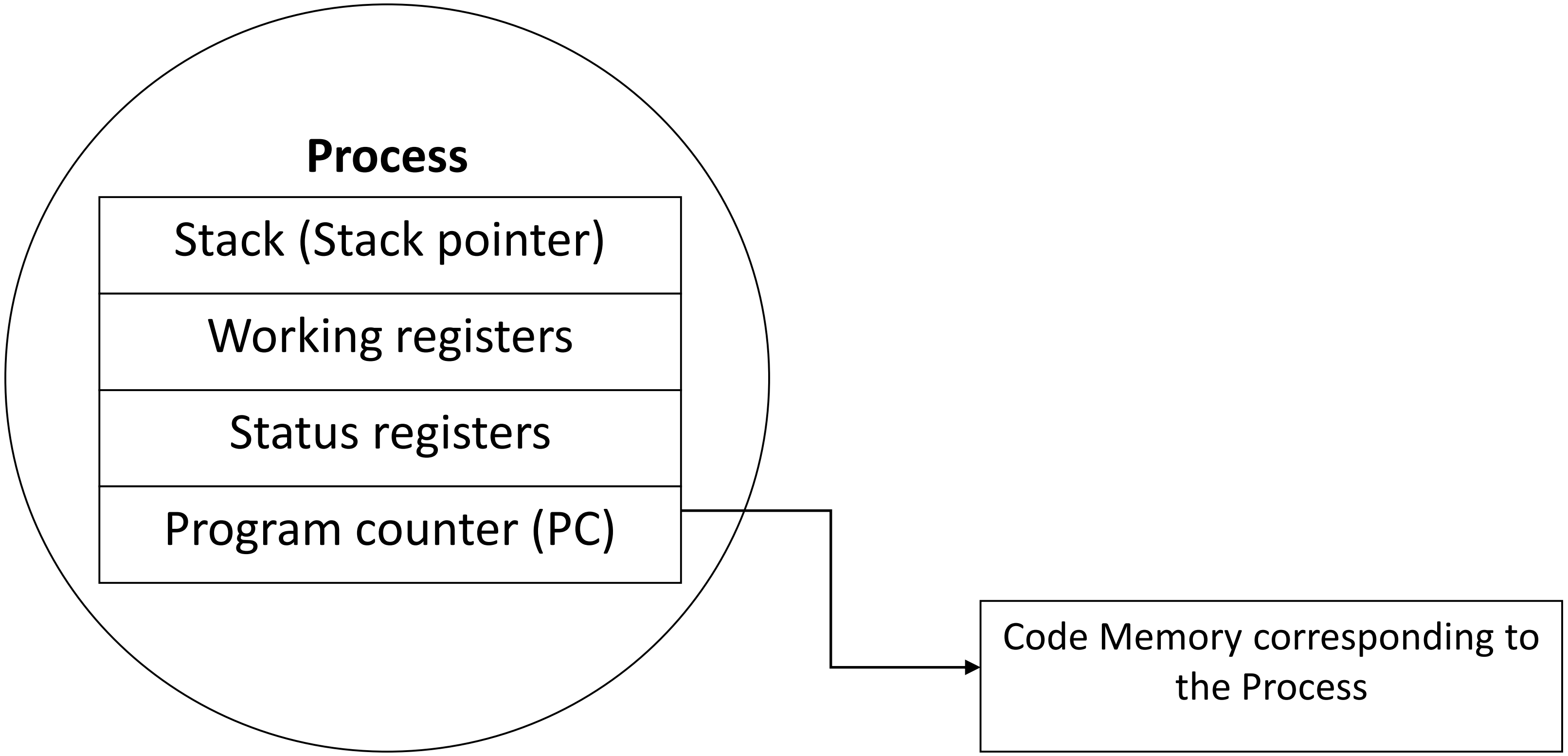


Figure: Structure of a Process

Process States and State Transition

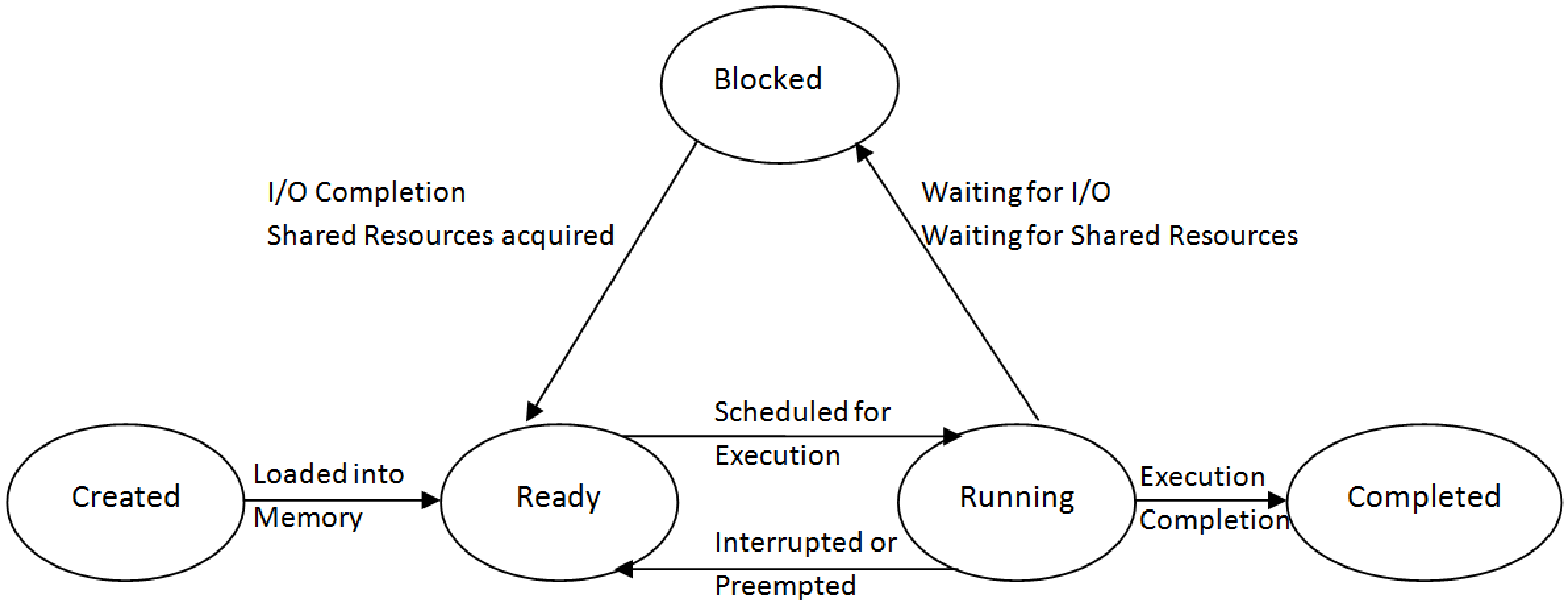


Figure: Process states and state transition representation

Process Control Block (PCB)

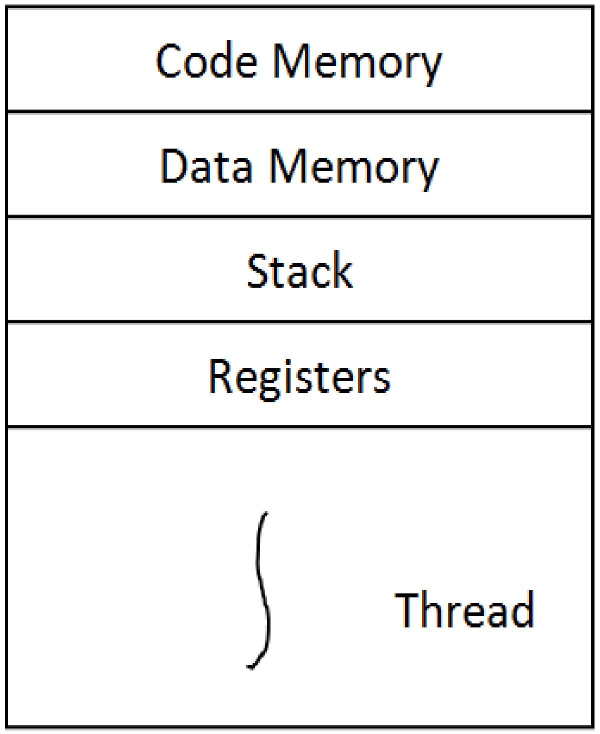
- Process state: The state may be new, ready, running, waiting/blocked/pending or completed.
- Program counter: It indicates the address of next instruction to be executed for current process.
- CPU registers: They include accumulators index registers, stack pointers, general purpose registers along with any status registers. The content of PC along with the state information of a process must be saved when an interrupt occurs.

…

- CPU Scheduling Information: This information includes the process priority and the pointers to the scheduling queues
- Memory management Information: This information includes the value of the base registers, page tables depending upon the memory system used by the OS.
- Accounting information: This information includes the amount of CPU time, time limits and process numbers.
- I/O status information: It includes the list of I/O devices allocated to a process.

THREADS

- A thread, basic unit of CPU utilization, is a single sequential flow of control within a process
- Different threads which are part of process share the data memory, code memory and the heap memory.
- However, the threads maintain their own thread status (CPU register value), Program Counter (PC) and stack.
- Multi threaded process: If a process has multiple threads of control, it can perform more than one task at a time.
- Single threaded process: If a process has a single thread of control it can perform a single task and is called.



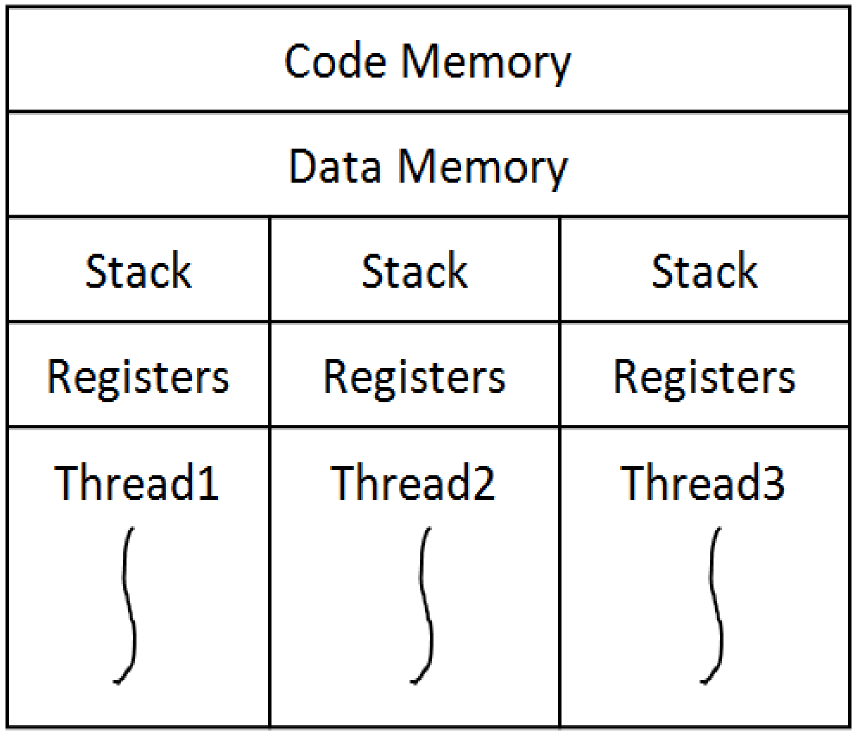


Figure: Single-Threaded Process

Figure: Multi-Threaded Process

Concept of Multithreading

- Inefficient CPU utilization when executed in sequence
- Process split into different threads for various sub-functions
- Multiple threads can be switched when threads enter the wait state
- Leads to more speedy execution of the process and the efficient utilization of the processor time and resources

Benefits of Multi-Threading

- Responsiveness
- Economical
- Utilization of multiprocessor architecture
- Efficient CPU utilization

User Level Threads

- The user level threads don't have kernel/OS support and they exists only in a running process.
- Even if a process contains multiple user level threads, the OS treats it as a single thread.
- It is the responsibility of the process to schedule each thread as and when ever required.
- User level threads of a process are non-preemptive at the thread level from the OS perspective.

Kernel Level Threads

- These are individual units of execution, which the OS treats as separate threads.
- The OS interrupts the execution of the currently running kernel thread and switches the execution of another kernel thread based on the scheduling policies implemented by the OS. Kernel level threads are pre-emptive.

Thread Libraries

- A thread library provides the programmer with an API for creating and managing threads
- **POSIX threads: Portable OS interface**
- **Win32 threads**
- **Java threads**

Thread and Process

Thread	Process
single unit of execution and is a part of the process	a program in execution and combines one or more threads
shares the code, data, heap memory with other threads of the same process	has its own code, data and stack memory
cannot live independently	contains at least one thread
inexpensive to create	Expensive to create

…

Thread	Process
Context switching is inexpensive and fast	Context switching is complex and involves lot of OS overhead and is comparatively slower.
If a thread expires, its stack is reclaimed by the process.	If a process dies, the resources allocated to it are reclaimed by the OS and all the associated threads of the process also dies.

Multiprocessing and Multitasking

- Multiprocessing
  - Execute multiple processes simultaneously
  - Requires multiprocessor systems
  - Multiple processors to execute multiple processes concurrently

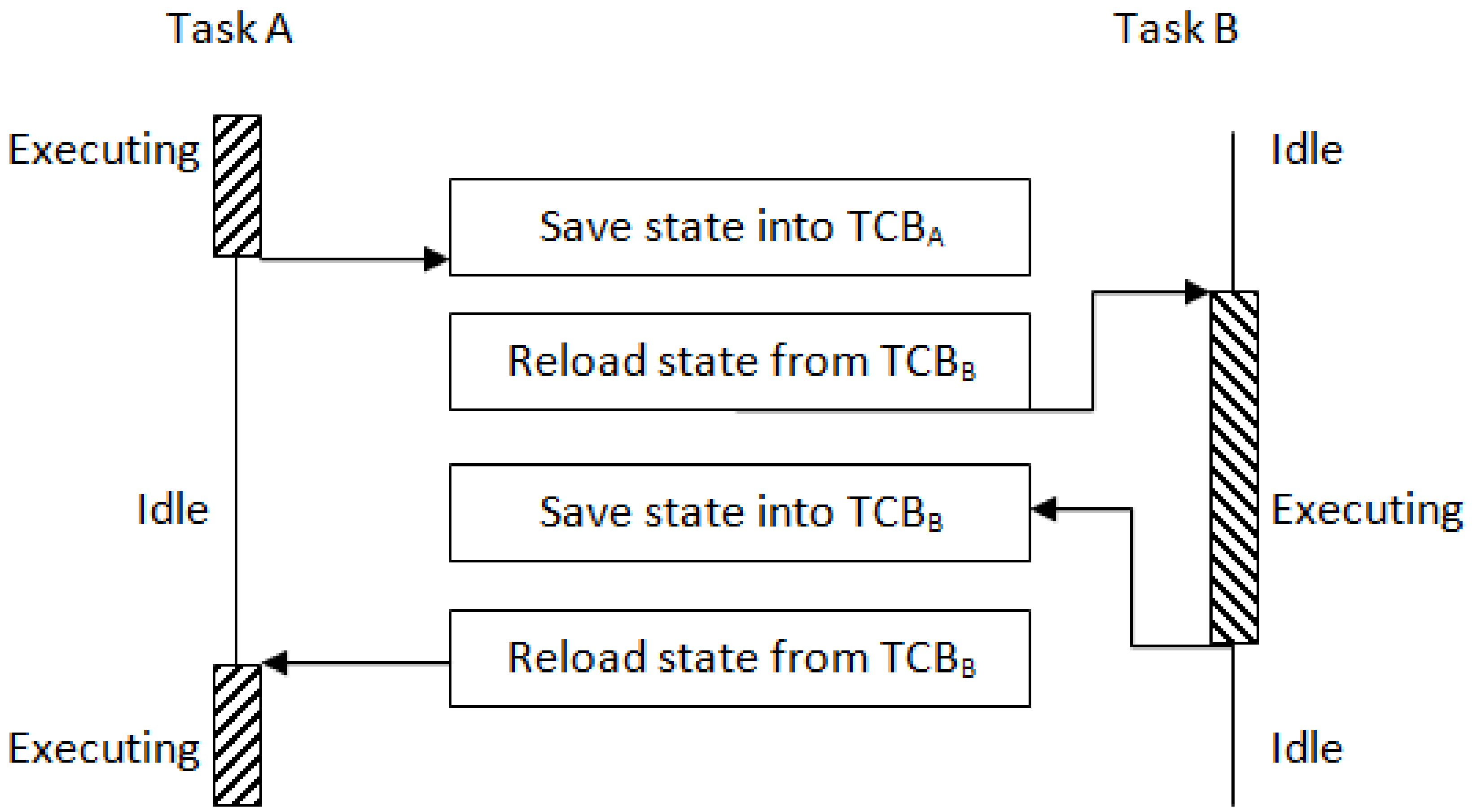
…

- Multitasking
  - Uni-processor system cannot execute multiple processes concurrently
  - Achieve some degree of pseudo parallelism by switching the execution among different processes
  - It is the ability of an OS to hold multiple processes in memory and switch the processor from executing one process to anther process
  - It involves context switching, saving and retrieval

Context Switching

- Each task exists in any one of different states
- During execution, tasks change from one state to another
- At any instant, single process is in running state
- CPU control changes from one process to another
    - Context of to be suspended task will be saved
    - Context of the to be executed task will be retrieved

…


Task A
Task B
Executing
Idle
Save state into TCB_A
Reload state from TCB_B
Idle
Save state into TCB_B
Executing
Reload state from TCB_B
Executing
Idle

…

- **Context Saving** is the act of saving the current contents which contains the context details
  - register details, memory details
  - system resource usage details
- **Context retrieval** is the process of retrieving the saved context details for a process which is going to be executed due to CPU switching.
- Context switch time is pure overhead because the system does no useful work while switching.

Types of Multitasking

- Co-operative Multitasking
- Preemptive Multitasking
- Non-preemptive Multitasking

Co-operative Multitasking

- Process gets chance to execute only when currently executing process voluntarily relinquished the CPU
- Process can hold CPU as mush time as it wants
- Process may not relinquish CPU when it enters waiting state
- Non-cooperative process can cause other processes to wait for a long time

Preemptive Multitasking

- Every process gets a chance to execute based on preemptive scheduling
- Currently running process is preempted to give chance to other process to execute
- Preemption of task may be based on time slots or priority

Non-preemptive Multitasking

- Process is allowed to execute until it terminates or enters waiting state (system resource required)
- The process in execution cannot be halted by another process

Task Scheduling

- Determining which process to execute at a given instant
- Scheduling policies run by the kernel as a service
- Decision takes place when a process switches to
    - Ready state from running state
    - Blocked state from running state
    - Ready state from blocked state
    - Completed state

Factors for selection of scheduling criterion

- **CPU Utilization** – high
  - Percentage of CPU being utilized
- **Throughput** – high
  - Number of processes executed per unit of time
- **Turnaround time** – minimal
  - Time required for complete execution of a process
- **Waiting time** – minimal
  - Time spent by a process in a Ready Queue
- **Response time** – as least as possible
  - Time between the submission of a process and first response

Queues During Scheduling

- Job Queue
  - Contains all the processes of the system
- Ready Queue
  - Collection of processes ready for execution and waiting for CPU for executing it
- Device Queue
  - List of processes waiting for an I/O device

Scheduling Algorithm

- **Non-Preemptive Scheduling**
  - Employed in systems with non-preemptive multitasking model
  - Currently executing process is allowed to run
    - Till it terminates
    - Enters the wait state waiting for a system resources
- **Preemptive Scheduling**
  - Employed in systems with preemptive multitasking model
  - Currently executing process can be preempted (stop temporarily) and select another process for execution
    - Preempted process is moved to the ready queue

Non – Preemptive Scheduling

- First Come First Served (FCFS) / FIFO
  - First entered process is served first
- Last Come First Served (LCFS) / LIFO
  - Last entered process is serviced first
- Shortest Job First (SJF)
  - Process with shorted estimated run time is scheduled first
- Priority Based Scheduling
  - High priority process is served first

Preemptive Scheduling

- Preemptive SJF / SRT (Shortest Remaining Time)
  - Process with shorter execution time is scheduled for execution
- Round Robin Scheduling
  - Each process is executed for a pre-defined time slot
- Priority Based Scheduling
  - High priority process is scheduled for execution first

Task Synchronization

- Concurrent processes share system resources
- Issue arises when two processes try to access the same resource
- Each process needs to be aware of shared resource access

Task Synchronization Issues

- Racing
- Deadlock
- Livelock
- Starvation

RACING

- Multiple processes compete to access and manipulate shared data
- Final value of shared data may be undesired one
  - Final process acting on data defined the final value
- Suppose that two processes A and B have access to a shared variable Count:
  - Process A: Count = Count + 5
  - Process B: Count = Count + 10

…

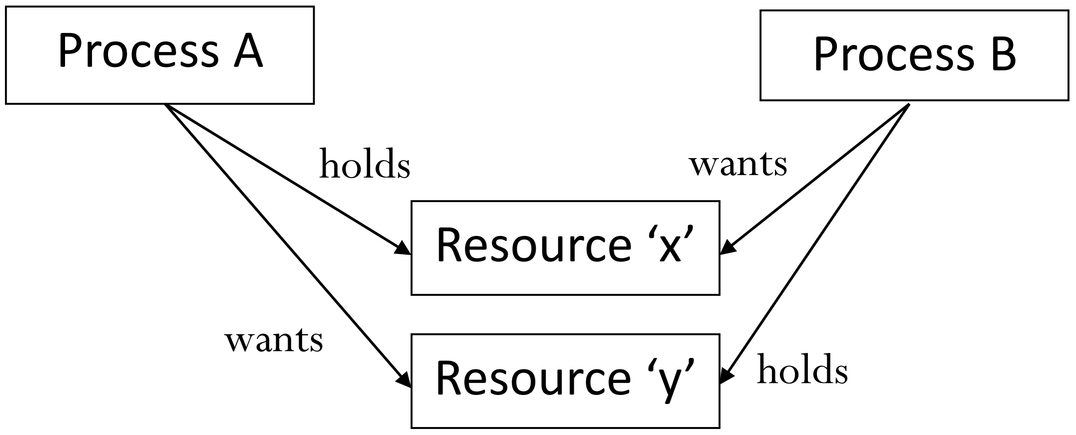
- Each statement requires several machine level instructions
  - For Count = Count + 5
    - A1: Load Ra, Count
    - A2: Add Ra, 05
    - A3: Store Count, Ra
  - For Count = Count + 10
    - B1: Load Rb, Count
    - B2: Add Rb, 10
    - B3: Store Count, Rb

…

Scenario 1	Scenario 2
A1: Load Ra, Count A2: Add Ra, 05 A3: Store Count, Ra **Context Switch** B1: Load Rb, Count B2: Add Rb, 10 B3: Store Count, Rb	A1: Load Ra, Count A2: Add Ra, 05 **Context Switch** B1: Load Rb, Count B2: Add Rb, 10 B3: Store Count, Rb **Context Switch** A3: Store Count, Ra
Count increased by 15	Count increased by 5

DEADLOCK

- None of the processes make any progress
- A process waiting for resource held by another process which is waiting for a resource held by the first process



Coffman Conditions

- Conditions favoring Deadlock Situations
  - Mutual Exclusion
    - Only one process can hold a resource at a time
  - Hold and Wait
    - A process must hold a shared resource and wait for another resource held by another process
  - No resource Preemption
    - Held resource can only be released voluntarily: no OS preemption

…

- Circular Wait
  - A process waiting for a resource which is currently held by another process which in turn waits for a resource held by another process.
  - Forms a circular wait queue

Deadlock Handling

- **Ignore Deadlocks**: Assuming deadlock free system
- **Detect and recover**: analyze the resource graph by graph analyzer algorithm and terminate a process or preempt the resource
- **Avoid Deadlocks**: careful resource allocation techniques
- **Prevent Deadlocks**: negating one of four conditions that cause deadlock

LIVELOCK

- Process changes its state but unable make any progress in the execution completion.
- While in deadlock a process enters a wait state for a response and continues in that state forever without making any progress in execution.
- For example: Two people attempting to cross each other in a narrow corridor.

STARVATION

- A process not getting required resources to continue its execution for a long time
- May arises from:
  - Byproduct of preventive measures of deadlock
  - Scheduling policies favoring high priority tasks, tasks with shortest execution time etc

Task Synchronization Techniques

- Synchronization essential to
  - Avoid conflicts in shared resources access
  - Ensure proper sequence of operation
  - Communicate between processes
- Critical Section: code memory area holding program for accessing shared resource
- Access of critical section should be exclusive to maintain synchronization

Techniques

- Mutual Exclusion through Busy Waiting/Spin Lock
- Mutual Exclusion through Sleep and Wakeup
  - Semaphore
    - Binary Semaphore
    - Counting Semaphore
  - Events

Mutual Exclusion through Busy Waiting/Spin Lock

- Uses lock variable for implementing mutual exclusion
- Each process/thread checks this lock variable before entering the critical section
    - Lock is set to 1 if process is already in critical section otherwise the lock is set to 0
- Processes always check the state of a lock and waits till the lock is available

…

- Challenge in implementing lock variable based synchronization
  - Non-availability of single atomic instruction for reading, comparing and setting of lock variable
  - Three different operations are required in general
  - Can violate mutual exclusion policy when a process is preempted just before it is about to set the lock variable
- Issued tackled by combining reading, testing and setting the lock in single step
  - Processors support single instruction Test and Set Lock (TSL)

Pros and Cons

- Useful in handling scenarios where processes are likely to be blocked for a shorter period of time
  - Avoid OS overheads on context switching and process re-scheduling
- Lock held for long time by a thread and preempted by OS then
  - Other threads have to spin for longer time
- Keeps the process always active
  - Wastage of processor time
  - High power consumption

Mutual Exclusion through Sleep & Wakeup

- Process undergoes sleep and enters blocked state rather than active when critical section is not available
- Blocked process is awakened by a process which leaves the critical section
  - Sends wakeup message

Semaphore

- Sleep and wakeup based mutual exclusion implementation for shared resource access
- Process requiring shared resource should acquired semaphore first and represents shared resources in use to other processes
- Resources shared among a process can be either for exclusive use by a process or for use by a number of processes at a time

Binary Semaphore (Mutex)

- Provides exclusive access to shared resource
- Allocates the resource to a single process at a time
- Mutex is a synchronization object provided by OS for process synchronization
    - Any process can create a mutex object and other processes can use this mutex object at a time
    - Set to signaled when not owned by any process
    - Set to non-signaled when owned by any process

Counting Semaphore

- Limits the access of resources to a fixed number of process
- Maintains a count between zero and a value
- State of counting semaphore object is set to signaled when count is greater than zero
- Count associated with semaphore object is decremented by one when process acquires it
- Count is incremented by one when released
- State of semaphore object is set to non-signaled when maximum supported process acquires it

Events

- Uses notification mechanism for synchronization
- Useful in sending a signal to a thread indicating that a particular event has occurred
- Creating and handling event objects for notification is OS kernel dependent

Device Drivers

- Software that is a bridge between the OS and the hardware
- Direct device access not allowed
- It initializes and manages communication with peripherals
  - setting up various CPU registers
  - Transfer of data
- OS loads driver files, usually .dll files, into the memory when device is connected

Types of device driver

- Built in drivers
  - Part of OS
- Installable drivers
  - Needs installation before hardware access
- User mode driver
  - Drivers running in user space
- Kernel Mode Driver
  - Drivers running in kernel space

Function of Device Driver

- Device Initialization
  - Configures different registers of the device
- Interrupt Configuration
  - Set interrupt type, enable, set priority
  - Bind interrupt with interrupt request (IRQ)
  - Register an interrupt service routine (ISR)

…

- Interrupt handling and processing
  - Served based on priority
  - Processing of an interrupt is handled in ISR
- Client Interfacing
  - Use of inter process communication mechanisms