Computer Organization and Embedded System

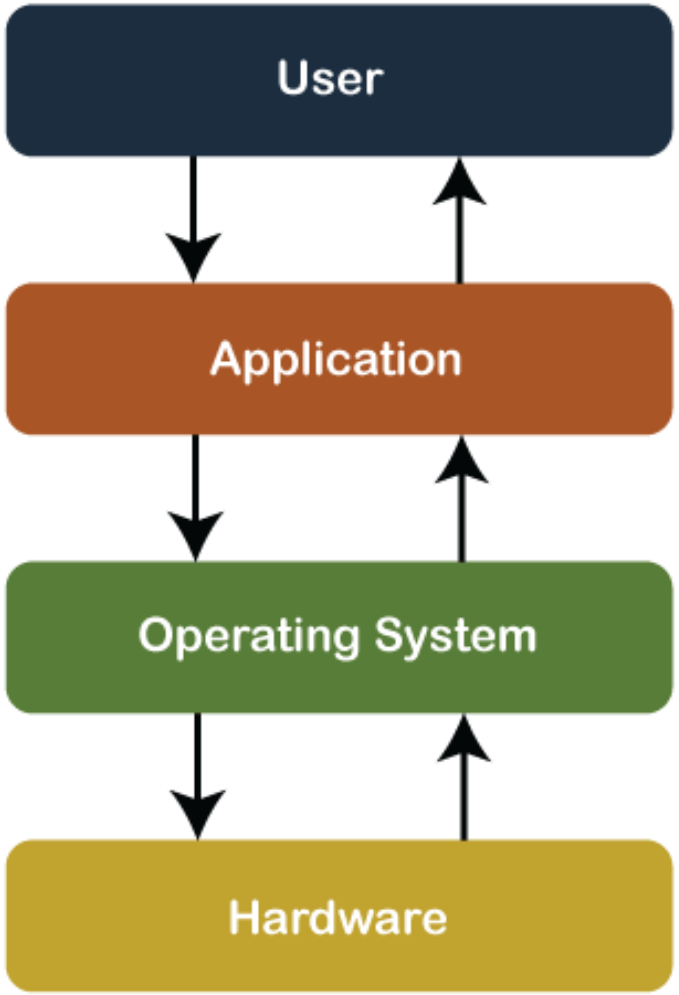
4.5. Real-Time operating and Control System (ACtE0405)

Ishwar Kumar Singh
Computer Engineer
Government of Nepal

4.5. Real-Time operating (ACtE0404)

- Operating System Basics,
- Task, Process, and Threads,
- Task Scheduling,
- Task Synchronization,

Operating System



- The operating system is a system program that serves as an interface between the computing system and the end-user.
- Operating systems create an environment where the user can run any programs or communicate with software or applications in a comfortable and well-organized way.
- Operating is a software program that manages and controls the execution of application programs, software resources and computer hardware.

Functions of OS

- **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.
- **Control over system performance :–**
  - Monitors overall system health to help improve performance.
  - records the response time between service requests and system response to having a complete view of the system health.
  - This can help improve performance by providing important information needed to troubleshoot problems.

Functions of OS

Memory Management :-

- The operating system manages the Primary Memory or Main Memory, which is fast storage and can be accessed directly by the CPU.
- An Operating System performs the following activities for memory management:
  - It keeps track of primary memory, i.e. which bytes of memory are used by which user program.
  - It allocates the memory to a process when the process requests and deallocates the memory when the process terminates or is performing an I/O operation.

Functions of OS

Process Management :-

- In a multi-programming environment, the OS decides the order in which processes have access to the processor and how much processing time each process has.
- This function of OS is called process scheduling.
- An Operating System performs the following activities for processor management.
  - Keeps track of the status of processes.
  - Allocates the processor to a process and De-allocates processor when a process is no more required.

Functions of OS

Device Management :-

- An OS manages device communication via their respective drivers.
- It performs the following activities for device management.
  - Keeps track of all devices connected to the system.
  - Decides which process gets access to a certain device and for how long.
  - Allocates devices in an effective and efficient way.
  - Deallocates devices when they are no longer required.

Functions of OS

File Management :-

- A file system is organized into directories for efficient or easy navigation and usage.
- These directories may contain other directories and other files.
- An Operating System carries out the following file management activities.
  - It keeps track of where information is stored.
  - It maintain and stores user access settings and status of every files.

Tasks, Processes and Threads

Task:

- In computing, a "task" is a unit of work that the operating system needs to perform.
- A task can be anything from running an application to handling a system operation.
- Tasks can be broken down into processes and threads.

Tasks, Processes and Threads

Process:

- A process is an instance of a program in execution.
- It contains the program code, its current activity, a program counter, registers and variables.
- The operating system creates and manages processes and assigns resources to them.
- Processes are independent of each other and are isolated in memory i.e. one process cannot directly access the memory of another.

Tasks, Processes and Threads

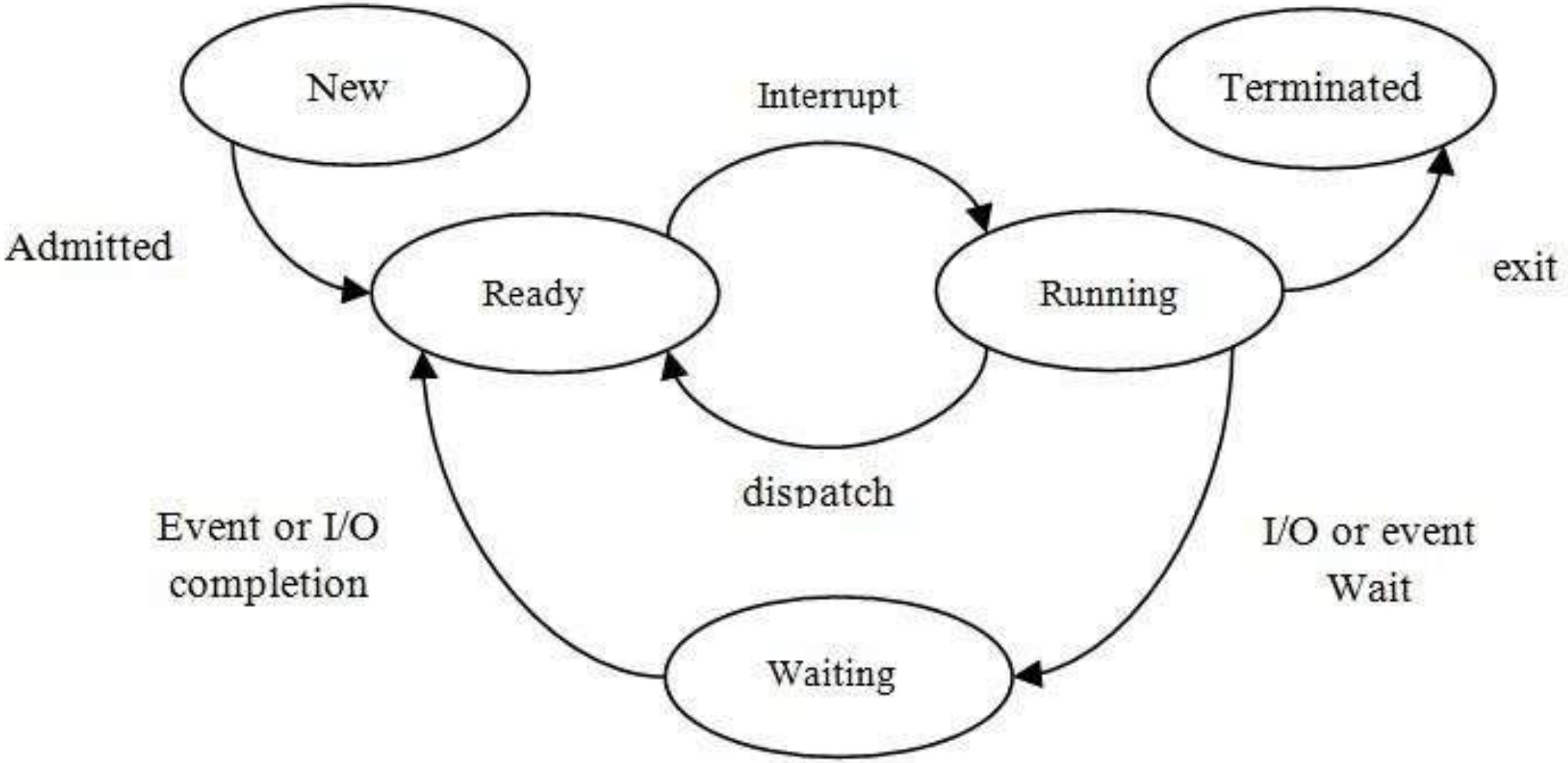
Key attributes of a process:

- **Process ID (PID):** A unique identifier for each process.
- **Memory:** Allocated to the process, including program code and data.
- **Execution Context:** The current state of the process, including registers and program counters.

Process State

- As a process executes, it changes state.
- The state of a process is defined part by the current activity of that process.
- Each process may be in one of the following states:
  - **New:** The process is being created.
  - **Running:** Instructions are being executed.
  - **Waiting:** The process is waiting for some event to occur (such as an I/0 completion or reception of a signal).
  - **Ready:** The process is waiting to be assigned to a processor.
  - **Terminated:** The process has finished execution.
- Many processes may be in ready and waiting state at the same time.
- But only one process can be running on any processor at any instant

Process State



Tasks, Processes and Threads

Thread:

- A thread is a smaller unit of a process.
- While a process may have multiple threads, all threads share the same memory space.
- Threads represent a sequence of instructions within a process.
- Threads within the same process can run concurrently, sharing resources like memory, which allows for efficient multitasking.
- Threads are more lightweight than processes.

Tasks, Processes and Threads

Types of threads:

- **User Threads:** Managed by a user-level library, not directly by the OS.
- **Kernel Threads:** Managed directly by the OS kernel.

Task Scheduling

- Task Scheduling is the method by which an OS decides which task (process or thread) should be executed by the CPU at any given time.
- The goal of scheduling is to ensure that all tasks are executed efficiently, fairly, and with minimal delay.
- The scheduler in the OS decides the order of task execution based on the scheduling algorithm.

Types:

- Non-Preemptive Scheduling
- Preemptive Scheduling

Task Scheduling

Non-Preemptive Scheduling:

- The process voluntarily yield control of the CPU.
- Once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.
- The operating system does not interrupt or preempt running process. Instead, it waits till the process completes its CPU burst time, and then after that it can allocate the CPU to any other process.
- Some Algorithms based on non-preemptive scheduling are: Shortest Job First (SJF basically non-preemptive) Scheduling and Priority (non-preemptive version) Scheduling, etc..

Task Scheduling

Preemptive Scheduling

- The **operating system** controls the CPU allocation to process.
- It decides when a process should be paused and another process should be given CPU time.
- The OS "preempts" or interrupts a running process to switch to another, ensuring that all process get a **fair share** of CPU resources.
- Some Algorithms that are based on preemptive scheduling are Round Robin Scheduling (RR), Shortest Remaining Time First (SRTF), Priority (preemptive version) Scheduling, etc.

Scheduling Algorithm

- There are mainly five types of process scheduling algorithms, which can be either **non-preemptive or preemptive**
  - First Come First Serve (FCFS)
  - Shortest Job First (SJF) Scheduling
  - Shortest Remaining Time (SRT) Scheduling
  - Priority Scheduling
  - Round Robin Scheduling

First Come First Serve

- First Come First Serve (FCFS) is the easiest and most simple CPU scheduling algorithm.
- In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.
- As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue.
- So, when CPU becomes free, it should be assigned to the process at the beginning of the queue.
- It is easy to implement and use.
- However, this method is poor in performance, and the general wait time is quite high.

Shortest Job First

- Shortest Job First (SJF) is a scheduling algorithm in which the process with the shortest execution time should be selected for execution next.
- This scheduling method can be preemptive or non-preemptive.
- It significantly reduces the average waiting time for other processes awaiting execution.
- It is associated with each job as a unit of time to complete.
- This algorithm method is useful for batch-type processing, where waiting for jobs to complete is not critical.
- It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

Shortest Remaining Time

- Shortest Remaining Time (SRT) is also known as SJF preemptive scheduling.
- In this method, the process will be allocated to the task, which is closest to its completion.
- This method prevents a newer ready state process from holding the completion of an older process.
- This method is mostly applied in batch environments where short jobs are required to be given preference.
- This is not an ideal method to implement it in a shared system where the required CPU time is unknown.

Priority Based Scheduling

- Priority scheduling is a method of scheduling processes based on priority.
- In this method, the scheduler selects the tasks to work as per the priority.
- Priority scheduling also helps OS to involve priority assignments.
- The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis.
- Priority can be decided based on memory requirements, time requirements, etc.

Round-Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm where, the OS defines a time quantum (slice).
- All the processes will get executed in the cyclic way.
- Each of the process will get the CPU for a small amount of time (called time quantum) and then get back to the ready queue to wait for its next turn.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.
- Time slice should be minimum, which is assigned for a specific task to be processed. However, it may vary for different processes.

Task Synchronization

- Task Synchronization refers to coordinating the execution of tasks (processes or threads) in a way that ensures correct results, especially when multiple tasks share resources.
- Without proper synchronization, tasks might interfere with each other, leading to inconsistent results or deadlock (where tasks cannot proceed).

Key concepts in task synchronization include:

- **Mutual Exclusion:** Ensures that only one task can access a shared resource at a time (e.g., using locks, semaphores).
- **Deadlock:** A situation where two or more tasks are waiting for each other to release resources, leading to a standstill.

Task Synchronization

- **Race Condition:** Occurs when multiple tasks try to access and modify shared resources simultaneously, leading to unpredictable outcomes.
- **Semaphores:** A synchronization tool that helps control access to shared resources.
- **Mutex (Mutual Exclusion):** A mechanism to enforce mutual exclusion by locking shared resources during access.

- Task synchronization is critical in multi-threaded and multi-process environments to prevent data corruption and ensure the system operates smoothly.

Program vs Process

Features	Process	Program
Definition	A program has a collection of instructions designed to accomplish a certain task.	A process is an example of an execution program.
Nature	It is an active entity.	It is a passive entity.
Lifespan	It has a limited lifespan.	It has a much higher lifespan.
Creation	The new process needs duplication of the parent process.	No much duplication is needed.

Program vs Process

Features	Process	Program
Resources	It has a high resources requirement, and it requires including CPU, memory address, disk, Input/output during its lifetime.	It doesn't have any resources requirements; it only needs memory space to store the instructions.
Required Process	It holds resources including CPU, disk, memory address, Input/Output etc.	The program is stored on a disk in a file and doesn't need any additional resources.

Program vs Process

Features	Process	Program
Computation Time	The process takes a long time to access and compute a single fact.	It has no computation time and cost.
Overhead	It has considerable overhead.	It has no significant overhead cost.
Cache Data	It may use the cache to store the retrieve the data as it uses OS paging scheme and cache replacement policy	It has the instruction to use cache for its data.

Thread vs Process

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.

Thread vs Process

S.N.	Process	Thread
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

User Level Threads vs Kernel Level Thread

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

Thank You.