Module II – Process Management

Module II

Operating System Functionality

- Process management
- Memory management
- Device management
- File system management
- Security and protection

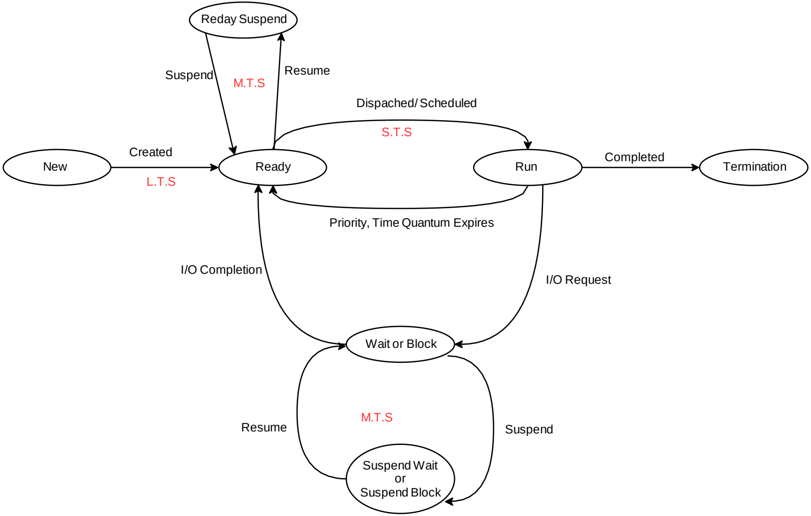
Process Management

- Process Concept: The program under execution is called process.
  - It should reside in the main memory.
  - It occupied the CPU to execute the instruction.
- Process will have various attributes:
  - **Process ID:** Unique ID assigned by the OS at time of process creation
  - **Process State**: Contains the current state information about where the process residing
  - **Program Counter:** Contains the address of next instruction to be executed
  - **Priority:** Parameter assigned by OS during process creation
  - General Purpose Registers
  - List of Open Files
  - List of Open Devices
  - Protection Information
- All the attributes of the process will be called context of the process
- The context of the process will be stored in Process control block (PCB)
- Every process will have its own PCB
- PCB will be stored in main memory

Process State and Operations

- Process State:
  1. New state
  2. Ready state
  3. Run state
  4. Termination or completion state
  5. Block or wait state
  6. Suspend ready
  7. Suspend wait or suspend block state
- Operations Performed on Process are:
  1. Creation
  2. Scheduling
  3. Dispatching
  4. Executing
  5. Termination or killing
  6. Suspending
  7. Resuming

Process State Diagram



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) ⇒ (must be fast)

Mid term scheduler

- Reponsible for suspending and resuming of the process
- Job done by M.T.S is called swapping.

Long-term scheduler (or job scheduler):

- Selects which processes should be brought into the ready queue (Ready state) Long-
- term scheduler is invoked infrequently (seconds, minutes) ⇒ (may be slow) The
- long-term scheduler controls the degree of multiprogramming

Processes can be described as either:

- I/O-bound process – spends more time doing I/O than computations, many short CPU bursts
- CPU-bound process – spends more time doing computations; few very long CPU bursts

Long-term scheduler strives for good process mix

Context Switch in OS

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

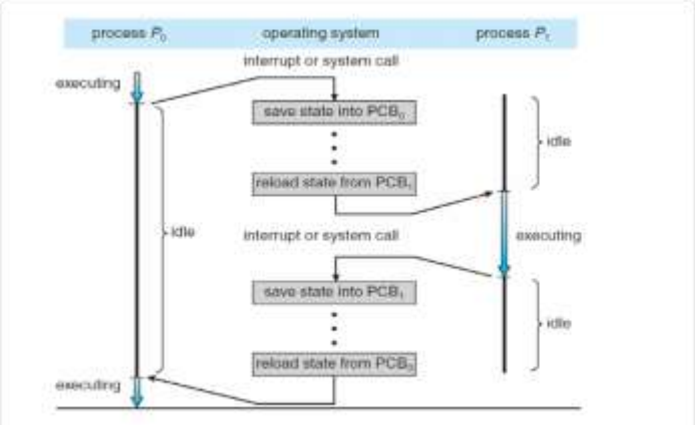
Context-switch time is overhead; the system does no useful work while switching

The more complex the OS and the PCB $\Rightarrow$ the longer the context switch

Time dependent on hardware support

Some hardware provides multiple sets of registers per CPU $\Rightarrow$ multiple contexts loaded at once

CPU Switch From Process to Process



CPU Scheduling

CPU Scheduling is a process that allows one process to use the CPU while another process is delayed (in standby) due to unavailability of any resources such as I/O etc, thus making full use of the CPU.

- CPU scheduling is essential for multitasking in operating systems.
- It determines which process runs at any given time.
- Understanding different process times is crucial for scheduling.

Types of CPU Scheduling Algorithms

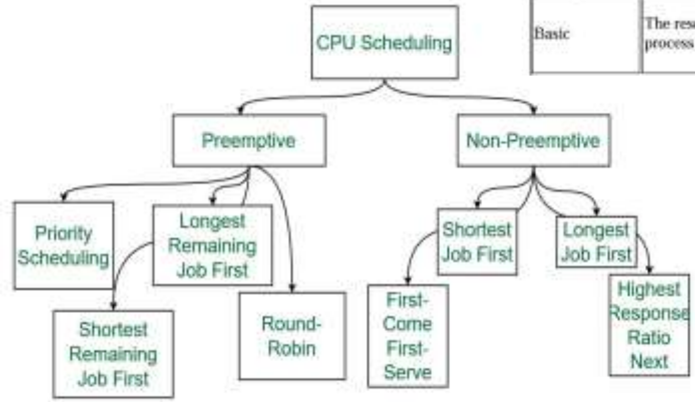
There are mainly two types of scheduling methods:

- **Preemptive Scheduling:** Preemptive scheduling is used when a process switches from running state to ready state or from the waiting state to the ready state.
- The resources (mainly CPU cycles) are allocated to the process for the limited amount of time and then is taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining.

  Non-Preemptive Scheduling: Non-Preemptive scheduling is used when a process terminates , or when a process switches from running state to waiting state.

  In this scheduling, once the resources (CPU cycles) is allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state.

Types of CPU Scheduling Algorithms



Basis for Comparison	Preemptive Scheduling	Non Preemptive Scheduling
Basic	The resources are allocated to a process for a limited time.	Once resources are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.

Types of Process Time

- Arrival Time:
  - The time when process is arrived in ready state.
- Burst Time:
  - Time a process spends executing on the CPU.
- Completion Time:
  - Time when process is completed its execution.
- Turnaround Time:
  - Total time taken from submission to completion of a process, $TAT = CT - AT$
- Waiting Time:
  - Time a process spends waiting in the ready queue, $WT = TAT - BT$
- Response Time:
  - Time from submission of a request until the first response is produced.

CPU Scheduling Algorithms

- First-Come, First-Served (FCFS) - Non Pre-emptive
- Shortest Job Next (SJN) / Shortest Job First (SJF) - Non Pre-emptive
- Priority Scheduling – Pre-emptive
- Round Robin (RR) – Pre-emptive
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

First-Come, First-Served (FCFS)

- The process which arrives first in the ready queue is firstly assigned to the CPU.
- In case of a tie, process with smaller process id is executed first.
- It is always non-preemptive in nature.
- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high. 3

First-Come, First-Served (FCFS)

Advantages

- It is simple and easy to understand.
- It can be easily implemented using queue data structure.
- It does not lead to starvation.

Disadvantages

- It does not consider the priority or burst time of the processes.
- It suffers from convoy effect i.e. processes with higher burst time arrived before the processes with smaller burst time.

First-Come, First-Served (FCFS)

Figure - The Convey Effect, Visualized

- Processes are scheduled in the order they arrive in the ready queue.
- Simple to implement but can suffer from the convoy effect.
- Criteria $\Longrightarrow$ Arrival Time, Mode: Non-preemptive

  Calculate Average T.A.T, Average W.T

Process	Arrival Time	Burst Time
P1	0	4
P2	1	3
P3	2	1

If the first process has large burst time (CPU Bound Process), then it will have major effect on the average waiting time of the process. This effect is called Convey Effect.

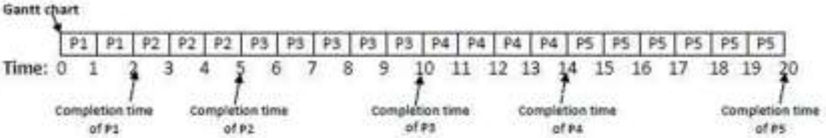
First-Come, First-Served (FCFS)

Example 1:

Q. Consider the following processes with burst time (CPU Execution time). Calculate the average waiting time and average turnaround time?

Process id	Arrival time	Burst time/CPU execution time
P1	0	2
P2	1	3
P3	2	5
P4	3	4
P5	4	6

Sol.



First-Come, First-Served (FCFS)

Turnaround time= Completion time – Arrival time

Waiting time= Turnaround time – Burst time

Process id	Arrival time	Burst time	Completion time	Turnaround time	Waiting time
P1	0	2	2	2-0=2	2-2=0
P2	1	3	5	5-1=4	4-3=1
P3	2	5	10	10-2=8	8-5=3
P4	3	4	14	14-3=11	11-4=7
P5	4	6	20	20-4=16	16-6=10

Average turnaround time= $\sum_{i=0}^{n}$ Turnaround time(i)/n where, n= no. of process

Average waiting time= $\sum_{i=0}^{n}$ Wating time(i)/n where, n= no. of process

Average turnaround time= 2+4+8+11+16/5 =41/5 =8.2

Average waiting time= 0+1+3+7+10/5 = 21/5 =4.2

Shortest Job Next (SJN) / Shortest Job First (SJF)

- Process which have the shortest burst time are scheduled first.
- If two processes have the same bust time, then FCFS is used to break the tie.
- This is a non-pre-emptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.
- Pre-emptive mode of Shortest Job First is called as Shortest Remaining Time First (SRTF).

Shortest Job Next (SJN) / Shortest Job First (SJF)

Advantages

- SRTF is optimal and guarantees the minimum average waiting time.
- It provides a standard for other algorithms since no other algorithm performs better than it.

Disadvantages

- It can not be implemented practically since burst time of the processes can not be known in advance.
- It leads to starvation for processes with larger burst time.
- Priorities can not be set for the processes.
- Processes with larger burst time have poor response time.

Shortest Job First (SJF)

Example-01:

Consider the set of 5 processes whose arrival time and burst time are given below-

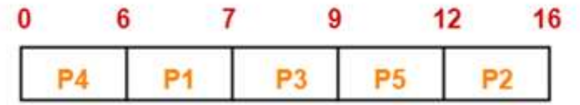
Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

Solution-

If the CPU scheduling policy is SJF non-preemptive, calculate the average waiting time and average turnaround time.

Shortest Job First (SJF)

<u>Gantt Chart-</u>



Gantt Chart

Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Shortest Job First (SJF)

Process Id	Exit time	Turn Around time	Waiting time
P1	7	7 – 3 = 4	4 – 1 = 3
P2	16	16 – 1 = 15	15 – 4 = 11
P3	9	9 – 4 = 5	5 – 2 = 3
P4	6	6 – 0 = 6	6 – 6 = 0
P5	12	12 – 2 = 10	10 – 3 = 7

Now,

- Average Turn Around time = (4 + 15 + 5 + 6 + 10) / 5 = 40 / 5 = 8 unit
- Average waiting time = (3 + 11 + 3 + 0 + 7) / 5 = 24 / 5 = 4.8 unit

Shortest Job First (SJF)

Example-02:

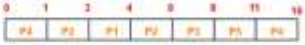
Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

If the CPU scheduling policy is SJF pre-emptive, calculate the average waiting time and average turnaround time.

Solution-

Gantt Chart-



Gantt Chart

Process Id	Exit time	Turn Around time	Waiting time
P1	4	4 – 3 = 1	1 – 1 = 0
P2	6	6 – 1 = 5	5 – 4 = 1
P3	8	8 – 4 = 4	4 – 2 = 2
P4	16	16 – 0 = 16	16 – 6 = 10
P5	11	11 – 2 = 9	9 – 3 = 6

Now,

- Average Turn Around time = (1 + 5 + 4 + 16 + 9) / 5 = 35 / 5 = 7 unit
- Average waiting time = (0 + 1 + 2 + 10 + 6) / 5 = 19 / 5 = 3.8 unit

Priority Scheduling

- Out of all the available processes, CPU is assigned to the process having the highest priority.
- In case of a tie, it is broken by FCFS Scheduling.
- Priority Scheduling can be used in both preemptive and non-preemptive mode.
- The waiting time for the process having the highest priority will always be zero in preemptive mode.
- The waiting time for the process having the highest priority may not be zero in non-preemptive mode.

Priority Scheduling

Advantages

- It considers the priority of the processes and allows the important processes to run first.
- Priority scheduling in pre-emptive mode is best suited for real time operating system.

Disadvantages

- Processes with lesser priority may starve for CPU.
- There is no idea of response time and waiting time.

Problem-01:

Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5

If the CPU scheduling policy is priority non-preemptive, calculate the average waiting time and average turnaround time. (*Higher number represents higher priority*)

Priority Scheduling

Solution-

Gantt Chart-

0–4	4–9	9–11	11–12	12–15
P1	P4	P5	P3	P2

Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	4	4 – 0 = 4	4 – 4 = 0
P2	15	15 – 1 = 14	14 – 3 = 11
P3	12	12 – 2 = 10	10 – 1 = 9
P4	9	9 – 3 = 6	6 – 5 = 1
P5	11	11 – 4 = 7	7 – 2 = 5

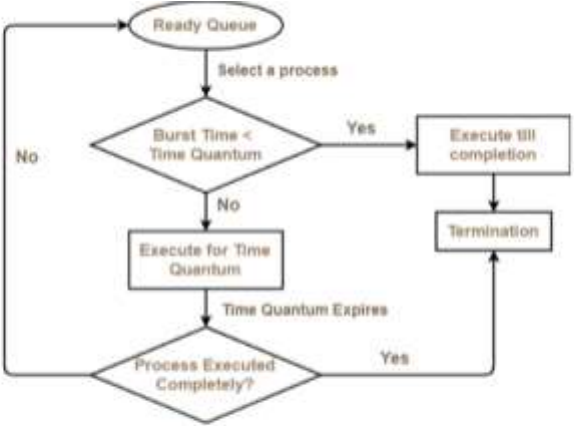
Now,

- Average Turn Around time = (4 + 14 + 10 + 6 + 7) / 5 = 41 / 5 = 8.2 unit
- Average waiting time = (0 + 11 + 9 + 1 + 5) / 5 = 26 / 5 = 5.2 unit

Round Robin (RR)

- CPU is assigned to the process on the basis of FCFS for a fixed amount of time.
- This fixed amount of time is called as time quantum or time slice.
- After the time quantum expires, the running process is preempted and sent to the ready queue.
- Then, the processor is assigned to the next arrived process.
- It is always preemptive in nature.

Round Robin (RR)



Round Robin Scheduling

Round Robin (RR)

Advantages

- It gives the best performance in terms of average response time.
- It is best suited for time sharing system, client server architecture and interactive system.

Disadvantages

- It leads to starvation for processes with larger burst time as they have to repeat the cycle many times.
- Its performance heavily depends on time quantum.
- Priorities can not be set for the processes.

Round Robin (RR)

Example 01:

Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

If the CPU scheduling policy is Round Robin with time quantum = 2 unit, calculate the average waiting time and average turnaround time.

Solution-

Ready Queue- P5, P1, P2, P5, P4, P1, P3, P2, P1

Gantt Chart-

0–2	2–4	4–5	5–7	7–9	9–11	11–12	12–13	13–14
P1	P2	P3	P1	P4	P5	P2	P1	P5

Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	13	13 – 0 = 13	13 – 5 = 8
P2	12	12 – 1 = 11	11 – 3 = 8
P3	5	5 – 2 = 3	3 – 1 = 2
P4	9	9 – 3 = 6	6 – 2 = 4
P5	14	14 – 4 = 10	10 – 3 = 7

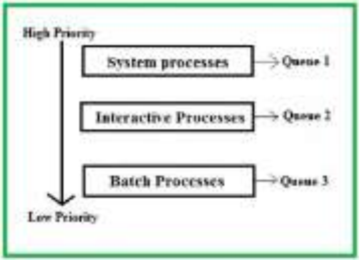
Now,

- Average Turn Around time = (13 + 11 + 3 + 6 + 10) / 5 = 43 / 5 = 8.6 unit
- Average waiting time = (8 + 8 + 2 + 4 + 7) / 5 = 29 / 5 = 5.8 unit

Multilevel Queue Scheduling

- Multiple queues for different priority levels.
- Each queue can have its own scheduling algorithm.
- Processes are permanently assigned to a queue.

Queue	Priority	Scheduling Algorithm
System Processes	High	FCFS
Interactive Processes	Medium	RR
Batch Processes	Low	SJF



Multilevel Feedback Queue Scheduling

- Similar to Multilevel Queue Scheduling but allows processes to move between queues.
- Dynamic adjustment based on process behavior.
- Helps in preventing starvation.

Queue	Priority	Scheduling Algorithm
High Priority	High	RR (Short Quantum)
Medium Priority	Medium	RR (Medium Quantum)
Low Priority	Low	FCFS

Important Points

- CPU scheduling algorithms are essential for efficient multitasking.
- Different algorithms are suitable for different types of workloads.
- Understanding process times and scheduling algorithms helps in optimizing system performance.

Multiple-Processor Scheduling

- Definition: Scheduling processes on multiple processors to optimize performance.
- Types of Multiple-Processor Scheduling:
  - Asymmetric Multiprocessing (AMP): A single processor, called the Master Server, handle all scheduling decisions and I/O processing, while the other processors only run user code.
  - Symmetric Multiprocessing (SMP): Each processor is self-scheduling.
- Processor Affinity:
  - Soft Affinity: The OS attempts to keep a process on the same processor but does not guarantee it.
  - Hard Affinity: The OS ensures that a process runs on the same processor.
- Load Balancing:
  - Push Migration: A specific task or process moves from an overloaded processor to an underloaded one.
  - Pull Migration: An underloaded processor pulls a task or process from an overloaded one.

Real-Time Scheduling

- Definition: Scheduling tasks to meet real-time constraints.
- Types of Real-Time Tasks:
  - Hard Real-Time: Tasks that must meet deadlines.
  - Soft Real-Time: Tasks where deadlines are desirable but not mandatory.
- Real-Time Scheduling Algorithms:
  - Rate Monotonic Scheduling (RMS): Priority is assigned based on the cycle duration. Shorter cycles get higher priority.
  - Earliest Deadline First (EDF): Tasks are prioritized based on their deadlines.
- Challenges in Real-Time Scheduling:
  - Ensuring predictability and meeting deadlines.
  - Handling priority inversion.

Threads - Overview

- Definition: A thread is the smallest unit of processing that can be performed in an OS (light weight process)
- Instruction will be less
- Large process can be divided into multiple threads
- Advantages of Threads:
  - **Responsiveness:** If a process is divided into multiple threads, the output from a completed thread can be responded to immediately, making it faster compared to waiting for the entire process to finish.
  - **Faster context switch:** The context switching time will be very less compared to the context switching time between the processes.
  - Effective utilization of multiprocessor system
  - **Resource Sharing:** Resources such as code, data, files and memory will be shared among the threads within the process except registers and stacks
  - **Economical :** Implementation does not requires and cost as various programming language provide support for threading

Multithreading Models

Threads are categorised into two type:

1. **User level thread** - managed entirely by the user-level thread library, without any direct intervention from the operating system's kernel
2. **Kernel level thread** - managed directly by the operating system's kernel

- Many-to-One Model:
  - Many user-level threads are mapped to one kernel thread.
  - Advantages: Simplicity and minimal OS support.
  - Disadvantages: Entire process blocks if a thread makes a blocking system call.
- One-to-One Model:
  - Each user-level thread maps to a kernel thread.
  - Advantages: More concurrency than the many-to-one model.
  - Disadvantages: Overhead of creating a kernel thread for each user thread.
- Many-to-Many Model:
  - Many user-level threads are mapped to many kernel threads.
  - Advantages: Multiplexing and flexibility.

Threading Issues

- The Fork-Join Problem:
  - Issues with fork system call creating a new process that duplicates all threads.
- Thread Cancellation:
  - Asynchronous Cancellation: Terminates the target thread immediately.
  - Deferred Cancellation: The target thread periodically checks if it should terminate.
- Signal Handling:
  - How signals are handled in multithreaded programs.
- Thread Pools:
  - Creating a number of threads at process startup and reusing them for different tasks.
- Thread-Local Storage (TLS):
  - Unique storage for each thread.

  Fork - a system call that creates a copy of a process, also known as a child process, which runs in parallel with the original process, also known as the parent process

Inter Process Communication (IPC)

The process are of two types:

- Independent process: Process is not affected by the execution of other processes
- Co-operating process: Execution of one process affects or affected by other process

Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both:

- Shared Memory
- Message passing

IPC

There are several reasons for providing an environment that allows process cooperation

- Information Sharing
- Computation Speedup
- Convenience

- Shared Memory – A region of memory that is shared by cooperating process is established. Process can then exchange information by reading and writing data to the shared region
- Message Passing – Communication takes place by means of messages exchanged between the cooperating processes.

IPC

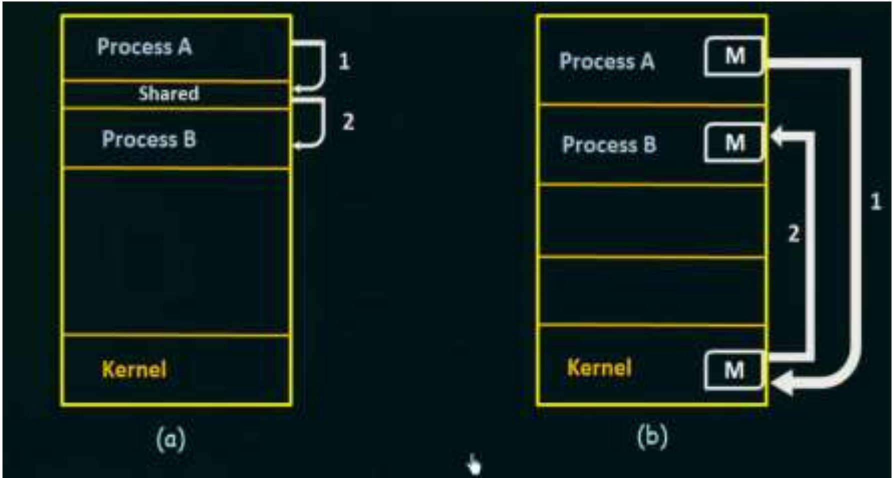


Fig: Communications models, (a) Shared memory, (b) Message Passing.

Process Synchronization

Process Synchronization is the coordination of execution of multiple processes in a multi-process system to ensure that they access shared resources in a controlled and predictable manner.

- Mechanism to ensure coordinated operation of multiple processes or threads.
- Prevents data inconsistencies and race conditions.
- Essential for maintaining data integrity in concurrent programming.

Steps

- Problem arise if there is no synchronization between processes
- Condition need to be followed to achieve synchronization
- Solutions

Producer Consumer Problem

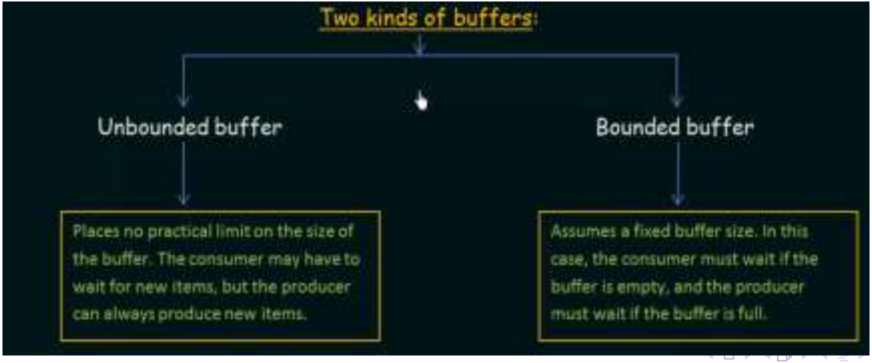
- A producer process produces information that is consumed by a consumer process

Example

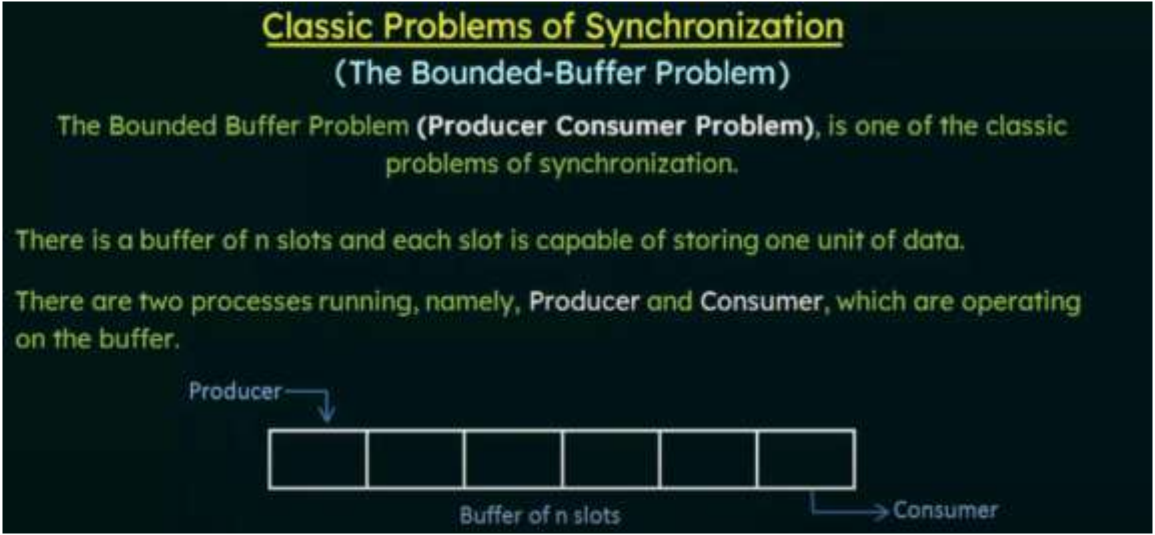
- A compiler may produce assembly code, which is consumed by the assembler.
- The Assembler in turn, may produce object modules which are consumed by the loader.
- One solution to the producer-consumer is using shared memory.
- To allow producer and consumer processes to run concurrently, we must have a buffer of items that can be filled by producer and emptied by the consumer.
- This buffer will reside in a region of memory that is shared by the producer and consumer processes.

Producer Consumer Problem

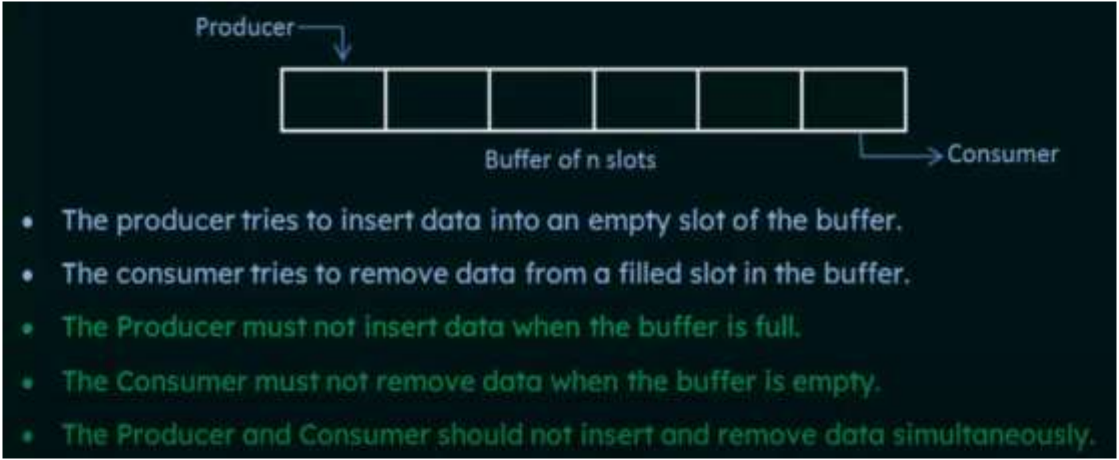
- The producer can produce one item while the consumer is consuming another item
- The producer and consumer must be synchronized, so that the customer does not try to consume an item that has not yet produced.



Producer Consumer Problem



Producer Consumer Problem



Producer Consumer Problem

Solution to the Bounded Buffer Problem using Semaphores:

We will make use of three semaphores:

1. m (mutex), a binary semaphore which is used to acquire and release the lock.
2. empty, a counting semaphore whose initial value is the number of slots in the buffer, since, initially all slots are empty.
3. full, a counting semaphore whose initial value is 0.

Producer Consumer Problem

1. m (mutex), a binary semaphore which is used to acquire and release the lock.
2. empty, a counting semaphore whose initial value is the number of slots in the buffer, since, initially all slots are empty.
3. full, a counting semaphore whose initial value is 0.

Producer

```
do {
  wait (empty); // wait until empty>0
                   and then decrement 'empty'
  wait (mutex); // acquire lock
  /* add data to buffer */
  signal (mutex); // release lock
  signal (full);  // increment 'full'
} while(TRUE)
```

Consumer

```
do {
  wait (full); // wait until full>0 and
                  then decrement 'full'
  wait (mutex); // acquire lock
  /* remove data from buffer */
  signal (mutex); // release lock
  signal (empty);  // increment 'empty'
} while(TRUE)
```

Condition need to be followed:

- If the buffer is full, producers must wait.
- If the buffer is empty, consumers must wait.
- Need to ensure mutual exclusion when accessing the buffer.
- Prevent race conditions where producers and consumers manipulate the buffer concurrently.

Universal Assumption: While executing any instruction, if interrupt occurs, then interrupt will be served after completion of the current instruction

Problem

If producer and consumer are not properly synchronized while sharing the common variable (critical section), then

- Inconsistent result
- Loss of data
- Deadlock

Solutions

- Software type:
  1. Lock variables
  2. strict alteration or Deckers algorithm: Process takes turn to enter the CS.
  3. Peterson's algorithm
- Hardware type:
  1. TSL instruction set: Test and set lock
- OS type:
  1. Counting Semaphore
  2. Binary Semaphore
- Programming language support type (Compiler):
  - Monitors

Semaphore

- Semaphore proposed by Edsger Dijkstra, is a technique to manage concurrent processes by using a simple integer value, which is known as a semaphore.
- Semaphore is simply a variable which is non-negative and shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment.
- A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait ( ) and signal ( ).

wait ( ) → P [from the Dutch word proberen, which means "to test"]

signal ( ) → V [from the Dutch word verhogen, which means " to increment"]

Semaphore

Definition of wait ( ):

```
P (Semaphore S) {
    while (S <= 0)
      ; // no operation
    S--;
}
```

Definition of signal ( ):

```
V (Semaphore S) {
    S++;
}
```

All the modifications to the integer value of the semaphore in the wait () and signal() operations must be executed indivisibly. That is, when one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.

Types of Semaphores:

1. Binary Semaphore:

The value of a binary semaphore can range only between 0 and 1. On some systems, binary semaphores are known as mutex locks, as they are locks that provide mutual exclusion.

2. Counting Semaphore:

Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

Other Points

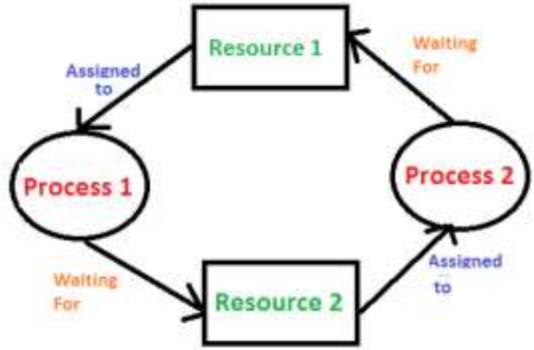
- The Producer-Consumer problem illustrates key concepts in synchronization.
- Proper use of semaphores prevents race conditions and ensures efficient data exchange.
- Understanding this problem is fundamental for studying operating systems and concurrent programming.

Importance of Process Synchronization

- Ensures that only one process accesses the critical section at a time.
- Prevents race conditions:
  - Situations where the outcome depends on the non-deterministic ordering of process execution.
- Maintains data consistency and integrity.
- Process synchronization is crucial for concurrent programming.
- It ensures data consistency and prevents race conditions.
- Common mechanisms include mutexes, semaphores, and monitors.

Deadlock in Operating System

A deadlock is a situation where a set of processes is blocked because each process is holding a resource and waiting for another resource acquired by some other process.



Basics of deadlock:

- Processes $\Longrightarrow$ Circle
- Resources $\Longrightarrow$ Square

Resource request/release life cycle

- The process will request for the resources to the OS.
- The OS will validate the request of the process
- If the request made by the process is valid, then OS will check for the availability of the resources
- If the resources are freely available then they will be allocated to the process, otherwise the process has to wait.
- If all the resources requested by the process for execution are allocated then the process will go into execution.
- Once the execution is completed the process will release all the resources.

Concepts in Deadlock

1. Deadlock characteristics
2. Deadlock prevention
3. Deadlock avoidance
4. Deadlock detection
5. Deadlock recovery

Deadlock characteristics

1. Mutual Exclusion:
   - Resources are allocated to only one process or are freely available
   - There should be one to one mapping between the the resource and process
2. Hold and wait: Process is holding the resources and waiting on a some other resources simultaneously
3. No pre-emption:
   - The resources has to be voluntarily released by the process after completion of the execution
   - Not allowed to forcefully preempt the resources from the process
4. Circular wait: The process are circularly waiting on each other for the resources

Deadlock Prevention

By dissatisfying the any one of the below condition the deadlock can be prevented

1. Mutual exclusion: It is not possible to dissatisfying the mutual exclusion always because of a shareable and non-shareable resources. Example- file(shareable), Printer(non-shareable)
2. Hold and wait:
   - Allocate all the resources before start of the execution
   - Process should release all the existing resource before making the new request
3. No Pre-emption: preempt the resource from the process if it waiting for the other resources
4. Circular wait: The resources are assigned with unique numerical numbers. The process can request the resources in the increasing order of enumeration.

Safe State

A safe state can be defined as a state in which there is no deadlock. It is achievable if:

- If a process needs an unavailable resource, it may wait until the same has been released by a process to which it has already been allocated. if such a sequence does not exist, it is an unsafe state.
- All the requested resources are allocated to the process.

Deadlock Avoidance

The deadlock avoidance will be implemented by using the Bankers algorithm.

- If we can satisfy the process needs with the current available resources, then the system is said to be in safe state, otherwise, the system is said to be in unsafe state.
- If system is in unsafe state then there is chance of deadlock
- The order in which we satisfy the processes needs to avoid deadlock is called safe sequence.
- The deadlock avoidance is less restrictive than deadlock prevention
- If the system is in safe state then request of the process will be granted. If the system is in unsafe state then request of the process will be denied and this way deadlock will be avoided.

Deadlock Detection

Deadlock detection is the process of finding out whether any process are stuck in loop or not. There are several algorithms like

- Resource Allocation Graph – for single instance resource.
- Banker's Algorithm: Illustrate with example

Deadlock Avoidance

The deadlock avoidance will be implemented using Banker's Algorithm:

- Maximum Need
- Current Allocation
- Available Resources
- Remaining Need

[redacted]

The order in which we satisfy the needs of all the processes is called the Safe Sequence. It may not be unique.

[redacted]

- If the system is in a safe state, it is not prone to deadlock.
- The unsafe state purely depends on the processes.

Banker’s Algorithm Example

	Maximum Need			Current Allocation			Remaining Need
Process	A	B	C	A	B	C	A B C
P0	3	2	2	1	2	1	2 0 1
P1	6	1	3	2	1	1	4 0 2
P2	3	1	4	2	0	2	1 1 2
P3	4	2	2	3	1	1	1 1 1
P4	2	2	3	1	0	1	1 2 2

Table 1: Resource Allocation Table

Current Available: ABC $\Longrightarrow$ 111

Banker's Algorithm Applied

Each and every time when the process requests for any resource, the Banker's algorithm is applied to identify whether the system is in a safe or unsafe state.

Banker's Algorithm Applied

Methods For Handling Deadlock

There are three ways to handle deadlock

- Deadlock Prevention or Avoidance : Banker's algorithm to avoid Deadlock
- Deadlock Recovery: There are several Deadlock Recovery Techniques:
  - Manual Intervention
  - Automatic Recovery
  - Process Termination
  - Resource Preemption
- Deadlock Ignorance : If a deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take.

References

[1] CS UIC: Operating Systems Structures
Available at: https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/2_Structures.html

[2] Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin, Operating System Concepts, Ninth Edition

[3] GeeksForGeeks: OS Basics
Available at: https://www.geeksforgeeks.org/what-is-an-operating-system/