USN | | | | | | | | | | |

Internal Assessment Test 1 – Nov 2024

Sub:	Operating System					Sub Code:	BCS303	Branch:	CSE	
Date:	8.11.2024	Duration:	90 mins	Max Marks:	50	Sem / Sec:	III/ A, B, C		OBE	

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1	a) Distinguish between the following terms. (i) Multiprogramming and Multitasking (ii) Multiprocessor System and Clustered System b) Explain the services of the operating system that are helpful for the user and the system	4+6	CO1	L2
2	a) What are system calls? Briefly explain all its types. b) Explain the layered approach of operating system structure with a supporting diagram	5+5	CO1	L2
3	a) Explain Producer-Consumer Problem with example? b) Describe the implementation of inter-process communication using shared memory and message passing.	5+5	CO3	L2
4	Explain Process and explain various operation on processes with a neat diagram	10	CO1	L2
5	Compute the waiting time and turnaround time and average waiting time and average turnaround time for the given processes using non preemptive SJF and SRTF and scheduling algorithm.	5+5	CO2	L3
6	Describe briefly critical section? What are the three requirements to be met by a solution to the critical section Problem?	3+7	CO2, CO3	L2

Process	Arrival Time	Burst Time
P1	**0**	**8**
P2	**1**	**4**
P3	**2**	**9**
P4	**3**	**5**

CI CCI HOD

USN | | | | | | | | | | |

Internal Assessment Test 1 – Nov 2024

Sub:	Operating System					Sub Code:	BCS303	Branch:	CSE	
Date:	8.11.2024	Duration:	90 mins	Max Marks:	50	Sem / Sec:	III/ A, B, C		OBE	

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1	a) Distinguish between the following terms. (i) Multiprogramming and Multitasking (ii) Multiprocessor System and Clustered System b) Explain the services of the operating system that are helpful for the user and the system	4+6	CO1	L2
2	a) What are system calls? Briefly explain all its types. b) Explain the layered approach of operating system structure with a supporting diagram	5+5	CO1	L2
3	a) Explain Producer-Consumer Problem with example? b) Describe the implementation of inter-process communication using shared memory and message passing	5+5	CO3	L2
4	Explain Process and explain various operation on processes with a neat diagram	10	CO1	L2
5	Compute the waiting time and turnaround time and average waiting time and average turnaround time for the given processes using non preemptive SJF and SRTF and scheduling algorithm.	5+5	CO2	L3

Process	Arrival Time	Burst Time
P1	**0**	**8**
P2	**1**	**4**
P3	**2**	**9**
P4	**3**	**5**

6	Describe briefly critical section? What are the three requirements to be met by a solution to the critical section Problem?	3+7	CO2, CO3	L2

CI CCI HOD

IAT1 SOLUTION KEY OPERATING SYSTEM

a) Distinguish between the following terms.
(i) Multiprogramming and Multitasking (ii) Multiprocessor System and Clustered System
b) Explain the services of the operating system that are helpful for the user and the system

Difference between Multitasking and Multiprocessing :

S No.	Multi-tasking	Multiprocessing
1.	**The execution of more than one task simultaneously is known as multitasking.**	**The availability of more than one processor per system, that can execute several set of instructions in parallel is known as multiprocessing.**
2.	**The number of CPU is one.**	**The number of CPUs is more than one.**
3.	**It takes moderate amount of time.**	**It takes less time for job processing.**
4.	**In this, one by one job is being executed at a time.**	**In this, more than one process can be executed at a time.**
5.	**It is economical.**	**It is less economical.**
6.	**The number of users is more than one.**	**The number of users is can be one or more than one.**
7.	**Throughput is moderate.**	**Throughput is maximum.**

1b)Key Differences Between Multiprocessor and Clustered Systems

Feature	Multiprocessor System	Clustered System
Architecture	Multiple processors within a single machine	Multiple independent computers (nodes)
Memory	Shared memory (uniform access)	Distributed memory (each node has its own)
Coupling	Tightly coupled (processors are close-knit)	Loosely coupled (independent nodes connected by network)
Communication	Direct communication via shared memory or interprocessor communication	Communication via network (e.g., TCP/IP)
Scalability	Limited scalability due to shared memory and hardware constraints	Highly scalable by adding more nodes to the network
Fault	Less fault tolerant (failure of one	High fault tolerance (failure of

Feature	Multiprocessor System	Clustered System
Tolerance	processor can affect the system)	one node does not impact others)
Performance	Suitable for tasks requiring tightly coordinated parallel processing	Better for distributed tasks, can scale with the workload
Cost	More expensive due to specialized hardware and interconnects	Can be more cost-effective, as it utilizes off-the-shelf hardware
Examples	SMP systems, multiprocessor servers	Cloud computing, data centers, grid computing

b) Explain the services of the operating system that are helpful for the user and the system
A1b)
Operating System Services:
1) User Interface
2) Program Execution
3) I/O Operations
4) File System Manipulations
5) Communications
6) Error Detection

2a)What are system calls? Briefly explain all its types.

System Calls:

A **system call** is a programmatic way in which a program requests a service from the operating system (OS). The services provided by the OS might include hardware-related services like input/output operations, memory management, process control, and more. System calls provide an interface between user applications and the kernel of the operating system.

When an application needs to perform a task that requires access to system resources (such as creating files, sending data over a network, or interacting with hardware), it makes a system call. These calls allow programs to use the OS's services in a controlled and secure way, without giving direct access to the hardware.

Types of System Calls:

System calls can be categorized into several types based on the type of service they provide. Here are the main types:

Process Control System Calls
These system calls manage processes, including their creation, execution, and termination.

- • `fork()`: Creates a new process by duplicating the calling process. The new process is a child of the calling process.
- `exec()`: Replaces the current process image with a new process image, typically to execute a new program.
- `exit()`: Terminates the current process and returns a status code to the operating system.
- `wait()`: Makes a process wait until one of its child processes terminates.
- `kill()`: Sends a signal to a process, which can instruct it to terminate or perform some other action.

- **File Management System Calls**

These system calls allow applications to interact with the file system—creating, reading, writing, and deleting files.

1.
    1. `open()`: Opens a file to either read, write, or both.
    2. `read()`: Reads data from an open file.
    3. `write()`: Writes data to an open file.
    4. `close()`: Closes an open file descriptor.
    5. `delete()`: Deletes a file from the file system.
    6. `rename()`: Renames an existing file.
    2.

Device Management System Calls
These system calls provide access to device-specific services like interacting with hardware devices (e.g., disk drives, printers).

`ioctl()`: Allows a program to configure or control hardware devices via the device driver.

`read()`/`write()`: These can also be used to interact with devices, as in reading from or writing to devices like disk drives or keyboards.

Information Maintenance System Calls
These system calls are used to obtain or set various information related to system resources

1. `getpid()`: Retrieves the process ID of the calling process.
2. `getuid()`: Retrieves the user ID of the calling process.
3. `setuid()`: Sets the user ID of the calling process.
4. `gettimeofday()`: Returns the current time and date.
5. `uname()`: Retrieves system information, like kernel version and system architecture.

Communication System Calls
These system calls provide mechanisms for inter-process communication (IPC), enabling processes to communicate with each other.

1. `pipe()`: Creates a pipe, which allows one-way communication between processes.
2. `shmget()`: Creates a shared memory segment that can be used for communication between processes.
3. `msgget()`: Creates a message queue for passing messages between processes.
4. `semop()`: Performs operations on semaphores to synchronize access to shared resources.

Summary of System Calls Types:

Type	Example System Calls	Description
Process Control	fork(), exec(), exit(), wait()	Manage processes, including creation, execution, and termination.

Type	Example System Calls	Description
File Management	open(), read(), write(), close()	Interact with the file system—create, read, write, and delete files.
Device Management	ioctl(), read(), write()	Interact with hardware devices, such as reading or writing data.
Information Maintenance	getpid(), getuid(), setuid()	Retrieve or modify information about processes or system status.
Communication	pipe(), shmget(), msgget(), semop()	Facilitate inter-process communication (IPC).

2b) Explain the layered approach of operating system structure with a supporting diagram?
Ans 2b)

The layered approach of operating system structure is a design model that organizes the system into a set of layers, where each layer provides specific functionality and interacts with adjacent layers in a well-defined manner. This approach helps in managing complexity, increasing modularity, and improving maintainability.

Layered Approach Breakdown:

In the layered architecture, the operating system is divided into several layers, each with a specific role. The layers are arranged in a hierarchical manner, with the most basic functionalities at the bottom and the more complex services at the top. Below is a common structure:

Layer 0 - Hardware:

1. This is the lowest layer, which represents the physical hardware of the system, including the CPU, memory, input/output devices, etc.
2. This layer provides the fundamental resources and services to the upper layers.

Layer 1 - Hardware Abstraction Layer (HAL):

1. This layer abstracts hardware-specific details and provides a uniform interface to the upper layers.
2. It allows the operating system to be independent of the underlying hardware.

Layer 2 - Operating System Kernel:

1. The kernel is the core of the operating system that provides basic system services like process management, memory management, and communication between hardware and software.
2. It interacts directly with the hardware and ensures smooth execution of system processes.

Layer 3 - System Libraries:

1. These are collections of functions or routines that provide an interface for user-level applications to interact with the kernel and hardware.
2. They implement higher-level services like file handling, I/O operations, and network protocols.

Layer 4 - System Utilities:

1. This layer consists of essential tools and utilities that help in system management and monitoring, such as disk utilities, system configuration, and performance tools.

Layer 5 - User Interface:

1. The top layer provides the user interface, such as command-line shells, graphical user interfaces (GUIs), or other interaction modes.
2. This is where the user interacts with the operating system through applications, shells, or graphical environments.

Diagram Representation:

Below is a simple conceptual diagram of the layered approach:

Key Points:

- **Modularity**: Each layer is responsible for a specific functionality, and changes in one layer usually do not affect others, making it easier to manage and update.

- **Abstraction**: Higher layers do not need to understand the complexities of the hardware or lower-level details, thanks to abstraction layers.
- **Inter-layer Communication**: Layers communicate with each other, where a higher layer invokes services from the layer directly below it.

3a)Demostrate Producer-consumer probem?

The **Producer-Consumer problem** is a classic synchronization problem where two types of processes share a common, finite-size buffer. The **producer** generates data, stores it in the buffer, and then goes back to producing more. The **consumer** takes the data from the buffer and consumes it. The problem arises when multiple producers and consumers access the buffer concurrently, potentially causing data corruption or inefficiency.

The main challenges to solve in the Producer-Consumer problem are:

- **Mutual Exclusion**: Ensuring that no two processes (producer or consumer) access the buffer at the same time.
- **Buffer Limits**: Ensuring that the producer does not add data when the buffer is full and the consumer does not try to consume when the buffer is empty.
- **Synchronization**: Ensuring that producers and consumers are synchronized correctly to avoid race conditions.

Key Concepts and Synchronization:

To solve this problem, we use **semaphores** (or other synchronization primitives like mutexes or condition variables) to control access to the buffer.

- **Semaphore**: A signaling mechanism to control access to a shared resource.
    - A **producer** will wait until there's space in the buffer (i.e., the number of items in the buffer is less than the buffer's maximum size).
    - A **consumer** will wait until the buffer contains data (i.e., the number of items in the buffer is greater than 0).

Components:

- **Shared Buffer**: A fixed-size buffer to hold the data.
- **Producer**: Generates data and stores it in the buffer.
- **Consumer**: Takes data from the buffer and processes it.

Pseudocode Solution (with Semaphores):

1. `mutex = 1 // Binary semaphore for mutual exclusion empty = N // N is the number of empty slots in the buffer full = 0 // Initially, there are no filled slots`

Initialize semaphores:

- `Producer Process: The producer waits if the buffer is full, produces data, and then places it in the buffer. mutex`: To provide mutual exclusion while accessing the shared buffer (binary semaphore).
- `empty`: To keep track of the available empty slots in the buffer.
- `full`: To keep track of the number of filled slots in the buffer.

Consumer Process: The consumer waits if the buffer is empty, consumes data, and then removes it from the buffer.

```
plaintext
Copy code
Consumer:
while True:
    wait(full)          // Wait until there is data in the buffer
    wait(mutex)         // Lock the buffer for exclusive access
    consume_item()      // Consume an item from the buffer
    remove_item_from_buffer() // Remove the consumed item from the buffer
    signal(mutex)       // Unlock the buffer
    signal(empty)       // Notify that there is space in the buffer
```

Inter-Process Communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions. Two common techniques for IPC are **shared memory** and **message passing**. Both methods have distinct advantages and are used based on the needs of the system or application.

1. Inter-Process Communication using Shared Memory

Shared memory is a memory region that can be mapped into the address space of more than one process. It provides a way for processes to communicate by directly reading and writing to the shared region.

Key Characteristics:

- **Direct Access**: Processes share a portion of memory, which can be accessed directly by all processes that have mapped it into their address space.
- **Fast Communication**: Since processes can directly access shared memory, communication is faster compared to other IPC methods.
- **Synchronization Required**: Although processes can directly access shared memory, synchronization mechanisms (like semaphores or mutexes) are needed to avoid race conditions.

Steps to Implement Shared Memory

Create or Attach Shared Memory Segment:

- The operating system provides system calls like `shmget` (to create a segment) and `shmat` (to attach the segment to a process's address space).

Write and Read Data

- Processes can write to or read from the shared memory region. However, proper synchronization is required to prevent data corruption.

4) Explain Process and explain various operation on processes with a neat diagram?

Process in Operating Systems

A **process** is a program in execution. It is an active entity that requires resources such as CPU time, memory, and I/O devices to perform its task. A process goes through various stages during its life cycle, and it can interact with other processes.

Components of a Process

A process consists of several key components:

- **Program code**: The executable code that is executed by the CPU.
- **Program Counter (PC)**: Keeps track of the next instruction to be executed.
- **Stack**: Stores temporary data such as function parameters, local variables, and return addresses.
- **Heap**: Used for dynamic memory allocation during the execution of the program.
- **Data Section**: Contains global and static variables.

Process Life Cycle

A process goes through various states during its life cycle:

1. **New**: The process is being created.
2. **Ready**: The process is ready to execute but waiting for the CPU.
3. **Running**: The process is being executed by the CPU.
4. **Blocked/Waiting**: The process is waiting for some event or resource (e.g., I/O completion).
5. **Terminated**: The process has finished executing.

Operations on Processes

Creation of a Process (Fork):

1.
1. When a new process is created, it is assigned a unique process ID (PID).
2. It starts from the **New** state and may transition to the **Ready** state.
3. Operating System creates a **Process Control Block (PCB)** that holds all the necessary information about the process.

Scheduling:

1. The operating system's **scheduler** determines which process from the **Ready** queue gets the CPU.
2. Various algorithms like **Round Robin**, **First-Come-First-Serve (FCFS)**, and **Shortest Job Next (SJN)** may be used to select the process.

Context Switching

1. Context switching occurs when the CPU switches from one process to another. It involves saving the state (context) of the currently running process and restoring the state of the next process. This is an overhead for the operating system.

Process Blocking and Waiting:

1. A process can be blocked or put into a waiting state if it is waiting for some external resource, like an I/O operation, to complete.
2. Once the resource becomes available, the process is moved back to the **Ready** state.

Termination:

1. Once the process finishes its execution, it enters the **Terminated** state.
2. The operating system releases all resources allocated to the process.

Process Control Block (PCB)

Each process is represented by a **Process Control Block (PCB)**, which stores:

- **Process ID (PID)**: Unique identifier for the process.
- **Program Counter (PC)**: Address of the next instruction.
- **CPU Registers**: Temporary data for the process.
- **Memory Management Information**: Information about the allocated memory.
- **Process State**: Current state of the process (e.g., Ready, Running).
- **I/O Status Information**: Information about I/O devices being used.

Summary of States:

- **New**: A process is being created.
- **Ready**: The process is ready for execution and is waiting for the CPU.
- **Running**: The process is being executed by the CPU.
- **Blocked/Waiting**: The process is waiting for an event (e.g., I/O completion).
- **Terminated**: The process has completed execution

Q5)Compute the waiting time and turnaround time and average waiting time and average turnaround time for the given processes using non preemptive SJF and SRTF and scheduling algorithm.

Process	Arrival Time	Burst Time
P1	**0**	**8**
P2	**1**	**4**
P3	**2**	**9**
P4	**3**	**5**

Non Preemptive SJF

Average Turnaround Time=14.25

Average Waiting Time=7.75

SRTF:

Average Turnaround Time=12.25

Average Waiting Time=5.6

Q6)Describe briefly critical section?What are the three requirements to be met by a solutions to the critical section ?

A **critical section** is a part of a program where shared resources (such as data or hardware) are accessed by multiple processes or threads concurrently. If these resources are not properly managed, conflicts and inconsistent results may arise. The goal is to ensure that only one process or thread executes within the critical section at any given time.

The three requirements for a solution to the critical section problem are:

Mutual Exclusion: Only one process or thread can be inside the critical section at any moment. This ensures that no two processes simultaneously access shared resources.

Progress: If no process is in the critical section, and multiple processes want to enter, one of them should be allowed to enter the critical section. The decision should not be delayed indefinitely.

Bounded Waiting: A process requesting access to the critical section must have a limit on the number of times other processes can enter the critical section before it gets a chance to enter. This ensures fairness and prevents starvation.