NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

PROGRAM: SOFTWARE ENGINEERING

OPERATING SYSTEMS THEORY

ASSIGNMENT-01

SUBMITTED BY:

Name: Ahmed Ali

Roll No: 22P-9318

INSTRUCTOR NAME: Sir Usman Malik

A DEPARTMENT OF COMPUTER SCIENCE

NAME : AHMED ALI
ROLL NO : 22P-9318
SECTION : BS(SE)-5B

⇒ ASSIGNMENT 01

⇒ OPERATING SYSTEMS THEORY.

QUESTION NUMBER - 01

a) HOW DOES CPU INTERFACE WITH THE DEVICE TO CO-ORDINATE THE TRANSFER:

The CPU Interface with the device through a DMA controller; The process is as follow:

1) The CPU initializes DMA CONTROLLER by setting up the source & destination addresses, the Amount of Data to be transferred and direction of the transfer (whether it's from memory to device or from device to memory.)

2) After initialization, the DMA controller takes over the transfer process, directly managing the data movement b/w device & memory, freeing the CPU from handling individual I/O devices requests.

(b) HOW DOES CPU KNOWS WHEN THE MEMORY OPERATIONS ARE COMPLETE?

⇒ In DMA operations, CPU is informed about the completion of data transfers through interrupts.

⇒ Once the DMA ~~operation~~ controller competes the transfer of data block it sends an interrupt signal to the CPU.

⇒ This interrupt signals (the) CPU that the DMA operation has finished, allowing the CPU to take any necessary post-processing steps.

The interrupt mechanism is crucial for efficient CPU utillization, as it allows the CPU to perform other tasks while the data is being transferred by the DMA controller.

c)

DMA can cause some interference with the execution of programs, though this interference is generally minimal but possible forms of interference includes:

=> **INTERRUPT HANDLING:** When the DMA transfer is complete, the CPU recieves an interrupt, which briefly pauses the execution of the current program to handle the DMA completion. This, (context) switch & ~~small~~ introduces a small delay in user program execution.

=> **BUS ARBITRATION:** During the DMA transfer both the CPU & DMA controller may compete for access to the memory bus.

When the DMA controller is using bus for data transf the CPU cannot access memory, which can make a little delay in CPU operations. This concept in memory bus is called cycle stealing, as the DMA controller effectively "steals" memory cycles from CPU.

Q-NO.02

SYMMETRIC PROCESSING:

•) Each processor performs all tasks.

1) All processors have equal access to memory & I/O devices.
2) Each processor performs tasks from the operating system Independently.
3) Workload is distributed evenly among processors, improving performance & efficiency.

ASYMMETRIC PROCESSING:

•) Each processor is assigned specie task.

1) A master-slave relationship exists b/w processors.
2) The master processor controls the system & assigns tasks to ~~solve~~ slave processors
3) Only the master processor accesses the operating system, while slave processor execute the tasks assigned to them.

DIFFERENCES;

=> **CONTROL:**

SMP: All processors share control.
AMP: One processor (master) control others (slaves)

=> **WORK DISTRIBUTION:-**

SMP: Processors share tasks equally
AMP: Master assigns tasks to slaves.

=> **SYSTEM COMPLEXITY;**

SMP: More complex due to equal task distribu
AMP: Simpler as the master controls the system.

MULTIPROCESSOR SYSTEMS:-

=> **DEFINITION:-** Multiprocessor systems use 2 or more processors to perform tasks concurrently. These processors share memory & are connected to improve performance, reliability & processing power.

•) involve multiple CPU's that can execute processors in parallel.

=> **Advantages :**

① **INCREASED THROUGHPUT:** More processor lead to better overall system performance & faster task completion.

② **FAULT TOLERANCE :** if one processor fails, other can take over, improving reliability.

③ **COST EFFICIENCY:** Sharing resources like memory & I/O devices reduced system cost.

④ **SCALABILITY:** Additional processors can be added to handle more tasks as needed.

=> **DISADVANTAGES:**

① **COMPLEXITY:-** Managing multiple processors requires sophisticated softwar & hardware support.

② **RESOURCES CONTENTION:** ~~processors~~ processors may compete for shared resources, leading to potential bottlenecks.

③ **HIGHER POWER CONSUMPTION:** More processor consume more power which can increase operational costs.

Q- NO- 03

PURPOSE OF INTERRUPTS:

Interrupts are used to transfer control to the operating system in response to events that ~~occur~~ occur asynchronously to the current instruction execution. Their main purpose include:

=> **I/O HANDLING:** Devices like disks & network interfaces send interrupts when they are ready for data transfer, signaling the OS to stop its current activity & handle the event.

=> **IMPROVING EFFICIENCY:** Instead of constantly polling hardware devices, the CPU can execute ~~their~~ other instructions while waiting for an interrupt, thus using it's time more efficiently.

=> WHEN AN INTERRUPT OCCURS:

The CPU:

1) stop it's current activities.
2) saves the state of the current process.
3) Transfer control to a predetermined Interrupt service routine to handle the event.

⟶ DIFFERENCE B/W A TRAP & AN INTERRUPT:

•) **SOURCE:**

=> **INTERRUPT:** Generated by external hardware devices such as I/O devices or timers.

=> **TRAP:** A software generated interrupt caused by an error (e.g division by zero) or a specific request from a program (system call)

•) **ASYNCHRONOUS VS SYNCHRONOUS:**

=> **INTERRUPT:** Asynchronous meaning, it can happen at any time, regardless of the current state of CPU.

=> **TRAP:** Synchronous, meaning it is triggered by the execution of an instruction in the current process.

•) **PURPOSE:**

=> **INTERRUPT:** Signals the need for the atten
to an ~~(extend)~~ event, such as I/O completion or a timer inte
(external)

=> **TRAP:** Usually used to handle software errors or to execute system calls.

=> CAN TRAPS BE GENERATED INTENTIONALLY BY a USER PROGRAM:

Yes, Traps can be generated, (intentionally) by a user program, typically in the form of <u>system calls</u>.

=> System calls are mechanism for a program to request services from the operating system kernal that it does not have the privilege to execute directly; such as:

=> **I/O OPERATIONS:** (eg; reading or writing)
=> **PROCESS CONTROL** (e.g; creating or terminating a process)
=> **MEMORY MANAGEMENT:** (e.g; allocating or freeing memory)

These system calls generate traps, which transfer control from user mode to kernal mode, where the O.S can safely execute the requested operation. This ensures that user programs cannot directly access hardware or critical system resources, maintaining system stability and security.

Q-No. 4

SLOWEST

→ Magnetic Tapes
↓
→ Optical disks
↓
→ Hard disk drives
↓
→ Non-Valatile Memory
↓
→ Main memory (RAM)
↓
→ Cache Memory
↓
→ Registers

↓

FASTEST

Q-NO. 05

MULTIPROGRAMMING SYSTEMS:

•) **DEFINITION;** A system where multiple programs are loaded into memory & executed simultaneously by CPU.

•) **KEY-FEATURE:** CPU switches b/w programs to maximize resource utillization, but only one process runs at any given time.

•) **GOAL:** Increase CPU utillization by reducing idle time through task switching

•) **CPU COUNT:** Involves a Single CPU that handles multiple jobs by switching b/w them.

MULTIPROCESSING SYSTEMS:

•) **DEFINITION:-** A system that uses 2 or more CPUs (processors) to perform multiple tasks concurrently.

•) **KEY FEATURE:** can execute different processes simultaneously, allowing true parallelism.

•) **GOAL:** To enhance processing power, speed & reliability by using more than one CPU.

•) **CPU COUNT:** Involves multiple CPU's each capable of executing tasks independently.

KEY DIFFERENCES:-

① **PROCESSING UNITS:**

⇒ MULTIPROGRAMMING: Single CPU
⇒ MULTI PROCESSING: Multiple CPU

② **TASK EXECUTION:**

M. PROGRAMMING: CPU switches b/w tasks
M. PROCESSING: Multiple tasks run in parallel on different (processors)

③ **GOAL:**

M. PROGRAMMING: Maximize CPU utillization by minimizing idle time.
M. PROCESSING: Increase overall processing power & speed.

Q-NO-06:

REASONS - WHY CACHES ARE USEFUL:

SPEED ENHANCEMENT: Caches store frequently accessed data closer to CPU, allowing much faster data access compared to retrieving data from slower memory (e.g. main memory or disk). This improves overall system performance by reducing the time the CPU spends waiting for data.

2) REDUCED LATENCY: By storing recently accessed or frequently used data, caches help minimize latency in data retrieval, allowing the CPU to process information more efficiently.

PROBLEMS CACHES SOLVE:

1) SLOWER MEMORY ACCESS: caches help bridge the speed gap b/w the CPU & slower memory/storage devices (like RAM, hdd), reducing time spent on accessing data

2) FREQUENT DATA REACCESS: Caches optimize systems by keeping recently used ~~&~~ or frequently needed data close to the CPU, reducing the need to repeatedly access slower storage systems.

PROBLEMS CACHES CAUSE:

1) INCREASED COMPLEXITY: Managing a cache requires additional hardware & software complexity, including cache coherence, eviction policies & managing consistency b/w the cache & main memory.

2) CACHE MISS PENALTY: if data is not found in Cache (a "cache miss"), the system must retrieve it from slower memory, which may lead to performance degradation, especially if cache misses are frequent.

⇒ WHY NOT MAKE CACHE AS LARGE AS DEVICE & ELIMINATE DEVICE:

1) COST: Caches memory are more expensive than RAM or disk storage, so making a large cache would be costly.

2) TECHNOLOGY CONSTRAINTS: cache use faster, smaller technology, & scaling them up would be negating their speed advantage.

3) POWER CONSUMPTION: large cache consume more power, leading to inefficiencies, especially in low power devices.

4) DIMINISHING RETURNS: Increasing cache size offers little performance gain beyond a certain point, making large caches inefficient.

Q NO. 07

KEY DIFFERENCES B/W CLIENT-SERVER & PEER TO PEER (P2P) MODELS;

=> CONTROL & MANAGEMENT:

=> **CLIENT SERVER:** centeralized control; the server manages data, services and resources. Clients rely on the server for all requests.

=> **P2P:** Decenteralized control; each peer manages its own resources and can act as both a client and a server.

=> SCALABILITY:

=> **CLIENT SERVER:** Limited by server capacity; as the number of clients increases, the server can become a bottleneck

=> **P2P:** Highly scalable: more peers distribute the load & share resources, making it easier to scale without a central bottleneck.

=> ~~RESOURCE SHARING &~~ RELIABILITY:

=> **CLIENT SERVER:** if server goes down, client lose access to service.

=> **P2P:** if one peer fails, other can still provide services. (more resilent)

=> ~~RELIABILITY~~ RESOURCE SHARING:

=> **CLIENT SERVER:** Resources and data are stored on the server & clients request access.

=> **P2P:** Resources are distributed across peers, with each peer sharing resources with others.

=> COMMUNICATION:

CLIENT SERVER: Communication is typically oneway, from clients to server.

P2P: 2 way communication with peers both sending & recieving data.

NO. 8

URPOSE OF SYSTEM CALLS:

System calls provide an interface b/w user application & the operating system, allowing programs to request services such as file operation, memory management & process control from the OS.

RELATION TO THE OS: System calls how the OS to manage hardware resources & perform privileged tasks on behalf of user programs, ensuring secure & controlled access to system resources.

RELATION TO DUAL-MODE OPERATION:

① USER MODE: Regular application runs with limited privileges, ensuring they cannot directly access hardware or critical OS resources.

② KERNAL MODE: System calls transition the CPU from User to kernal mode, giving the OS full control to safely execute sensetive operations like interacting with hardware.