SMVS
Page No.
Date : / /

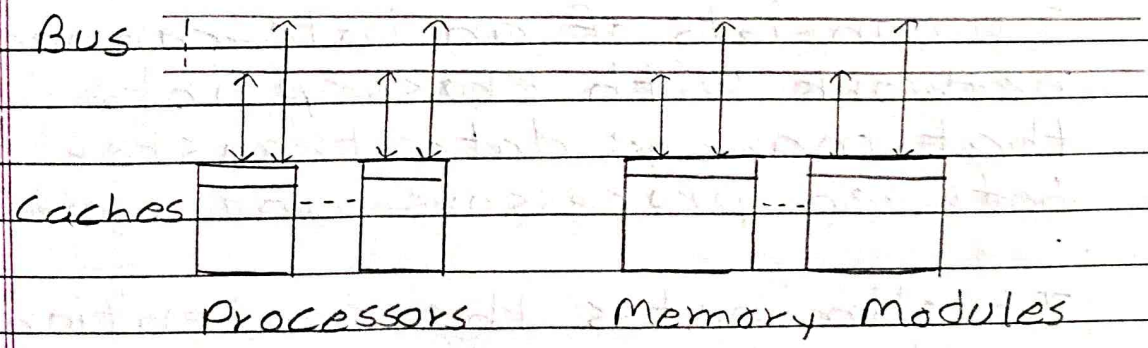
Programming with Shared Memory

1 Explain Shared memory multiprocessor system For Following:
(a) Using a Single Bus
(b) Using a Crossbar Switch

=> Using a Single Bus:

All Processors and memory modules are connected to a single shared communication medium, the bus.

The architecture operates with a single address space, where each memory location has a unique address accessible by all processors.



Date : / /

Each Processor is equipped with one or more levels of cache memory to store frequently accessed data.

This system effective for only small system with up to 8 processors.

The system architecture is simple to design and suitable for low-cost systems.

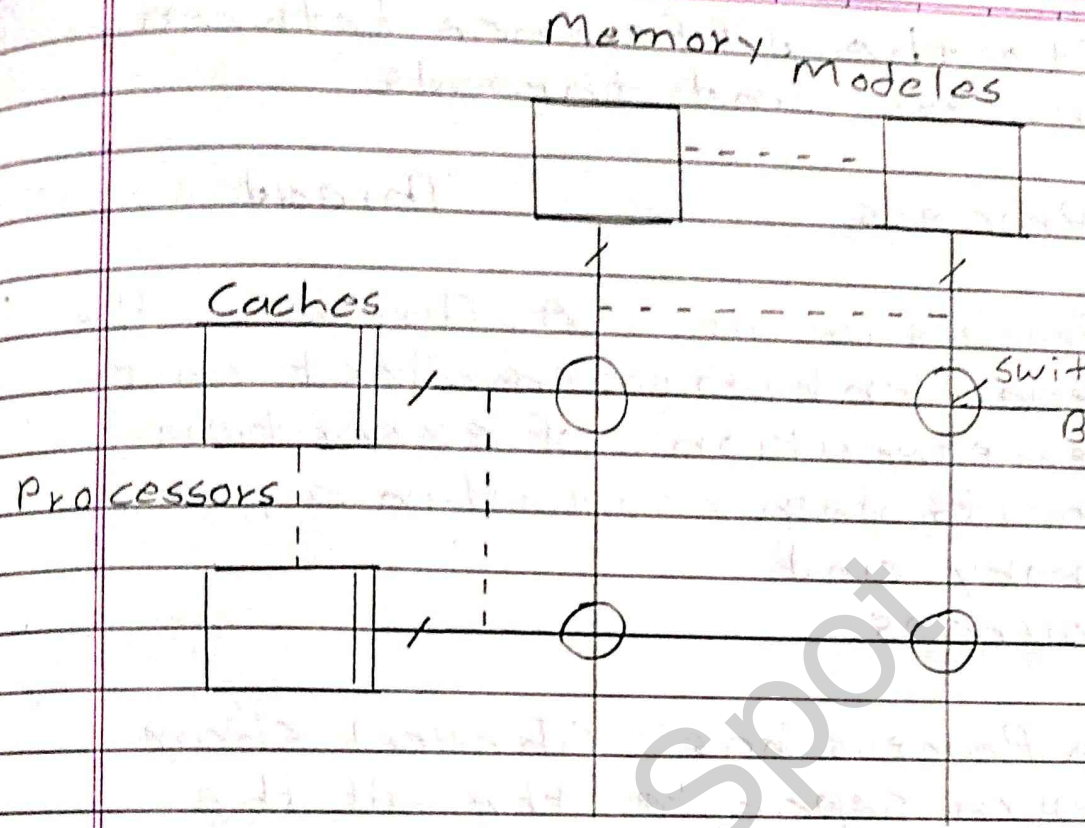
=> Using a Crossbar Switch:

A Crossbar switch provides a direct communication between processors and memory modules, allowing multiple processors to access different memory modules simultaneously.

It consists of an interconnection network with cross-points that manage data transfer between processors and memory.

It eliminates the contention problem inherent in single-bus systems.

Page No.
Date : / /



Each Processor can independently communicate with any memory module without interface of any other processors.

Supports a Large number of processor Compared to single-bus systems.

Its scalability is Limited by the cost and complexity of the switch.

SMVS
Page No.
Date : / /

2 Write the difference between a process and threads.

=>

	Process	Thread
1	A Process is an independent program under execution with its own memory and resources.	A Thread is the smallest unit of execution within a process.
2	Each Process has its own separate memory space and resources.	Thread share the all the Process resources and memory space.
3	Processes are independent from each other.	Thread are not isolated.
4	Creating a process involves more overhead due to resource allocation.	Creating thread is Lightweighted Process.
5	Context switching between processes is slower.	Context switching between thread is faster.

SMVS
Page No.
Date : / /

3 Explain the Followings under the sharing data:

=> Locks:

A lock is a synchronization primitive that allows only one thread or process to access a critical section at a time.

Types:

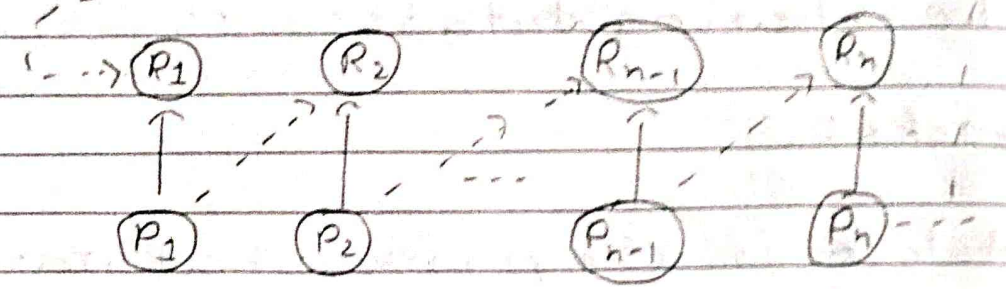
1) Mutex : A Binary Lock that permits only one thread to hold the lock at any time.

2) Read-Write Locks: Allow multiple readers but only one writer at a time.

=> DeadLock:

DeadLock occurs when two or more thread or process are waiting for each other to release resources and none can proceed.

Caues: Circular Wait
Mutual Exclusion
Hold-and-Wait



Use timeout mechanisms to release locks after a period.

=> Semaphores:

A Semaphores is a synchronization mechanism used to control access to a shared resource by multiple thread or processor.

Types:

1) Binary Semaphores: Acts like a mutex with value 0 and 1

2) Counting Semaphores: Allows multiple threads to access a resource simultaneously, up to a specified unit.

Operation:

1) Wait (P operation): Decrements

SMVS
Page No.
Date : / /

the semaphore value.
If the value is 0, the thread waits.

2) Signal(CV Operation): Increments the semaphore value, waking up waiting threads.

=> Monitor:

A Monitor is a high-level synchronization construct that encapsulates shared variable, data structure and methods to operate them.

It ensures that only one thread can execute a monitor procedure at a time.

It simplifies synchronization code by abstracting low-level mechanisms.

4 Explain Pthread with its condition variable.

=> Condition variable in Pthread are synchronization ~~ta~~ primitives

SMVS
Page No.
Date : / /

used to coordinate the execution of threads.

It allowing threads to wait for certain condition to be met before proceeding.

-> Thread Condition Variables:

1 Declaration and Initialization:

Condition variable are declared using the pthread_cond_t type.

They are initialized using the pthread_cond_init() function.

2 Destruction:

Condition Variables can be destroyed using pthread_cond_destroy().

3 Waiting on Condition Variable:

To make a thread wait for a condition:

```
pthread_cond_wait(&cond1,
                  &mutex1);
```

Page No.
Date : / /

4 Signaling Threads:

- To notify a waiting thread:
  pthread_cond_signal(&cond1);
- To notify all waiting threads:
  pthread_cond_broadcast(&cond1);

5 Atomicity:

The action of unlocking the mutex and putting in the waiting state are atomic to prevent race conditions.

5 Briefly discuss Shared Memory Programming Performance Issues

=> This are the Major Issuess:

1 Shared Data Access Issues:

(a) Cache Memory Usage: Cache memory speed up processor access to data compared to main memory.

(b) False Sharing: Multiple processors

Date : / /

altering separates bytes within the same cache block can degrade performance.

(c) Compiler Optimizations: Data layout can be optimized to reduce False sharing, but may cause storage inefficiencies.

2 Shared Memory Synchronization Issues:

(a) Mutual Exclusion Issues: Critical sections can serialize execution, reducing parallel performance.

(b) Barrier Synchronization: Can cause unnecessary waiting in synchronization algorithm.

(c) Event Synchronization: Used for communication between processes / threads.

3 Sequential Consistency Issue:

Instructions must execute in program order for each processor.

Date : / /

Memory consistency issues can arise due to interleaving, cache delays and reordering.