SNS COLLEGE OF TECHNOLOGY

Coimbatore-35.

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

COURSE NAME : 19CSB201 – OPERATING SYSTEMS

II YEAR/ IV SEMESTER

UNIT – II Process Scheduling And Synchronization

Topic: CPU Scheduling : Multiple-Processor Scheduling

Multiple-Processor Scheduling

If multiple CPUs are available, **load sharing** becomes possible—but scheduling problems become correspondingly more complex.

Approaches to Multiple-Processor Scheduling

- **asymmetric multiprocessing**
- **symmetric multiprocessing (SMP)**

A multiprocessor system has all scheduling decisions, I/O processing, and other system activities handled by a single processor—the master server.

The other processors execute only user code.

This **asymmetric multiprocessing** is simple because only one processor accesses the system data structures, reducing the need for data sharing.

- **symmetric multiprocessing (SMP), where each** processor is self-scheduling.
- All processes may be in a common ready queue, or each processor may have its own private queue of ready processes.
- Regardless, scheduling proceeds by having the scheduler for each processor examine the ready queue and select a process to execute.
- If we have multiple processors trying to access and update a common data structure, the scheduler must be programmed carefully.
- We **must ensure that two separate processors do not choose to schedule the same process** and that processes are not lost from the queue.

Processor Affinity

If the process migrates to another processor. The contents of cache memory must be invalidated for the first processor, and the cache for the second processor must be repopulated.

Because of the high cost of invalidating and repopulating caches, most SMP systems try to avoid migration of processes from one processor to another and instead attempt to keep a process running on the same processor.

This is known as **processor affinity—that is, a process has an affinity for the processor on which it is currently running.**

Soft & Hard affinity

When an operating system has a policy of attempting to keep a process running on the same processor—but not guaranteeing that it will do so—we have a situation known as **soft affinity**.

Here, the operating system will attempt to keep a process on a single processor, but it is possible for a process to migrate between processors.

In contrast, some systems provide system calls that support **hard affinity**, thereby allowing a process to specify a subset of processors on which it may run.

The **main-memory architecture** of a system can **affect processor affinity issues**.

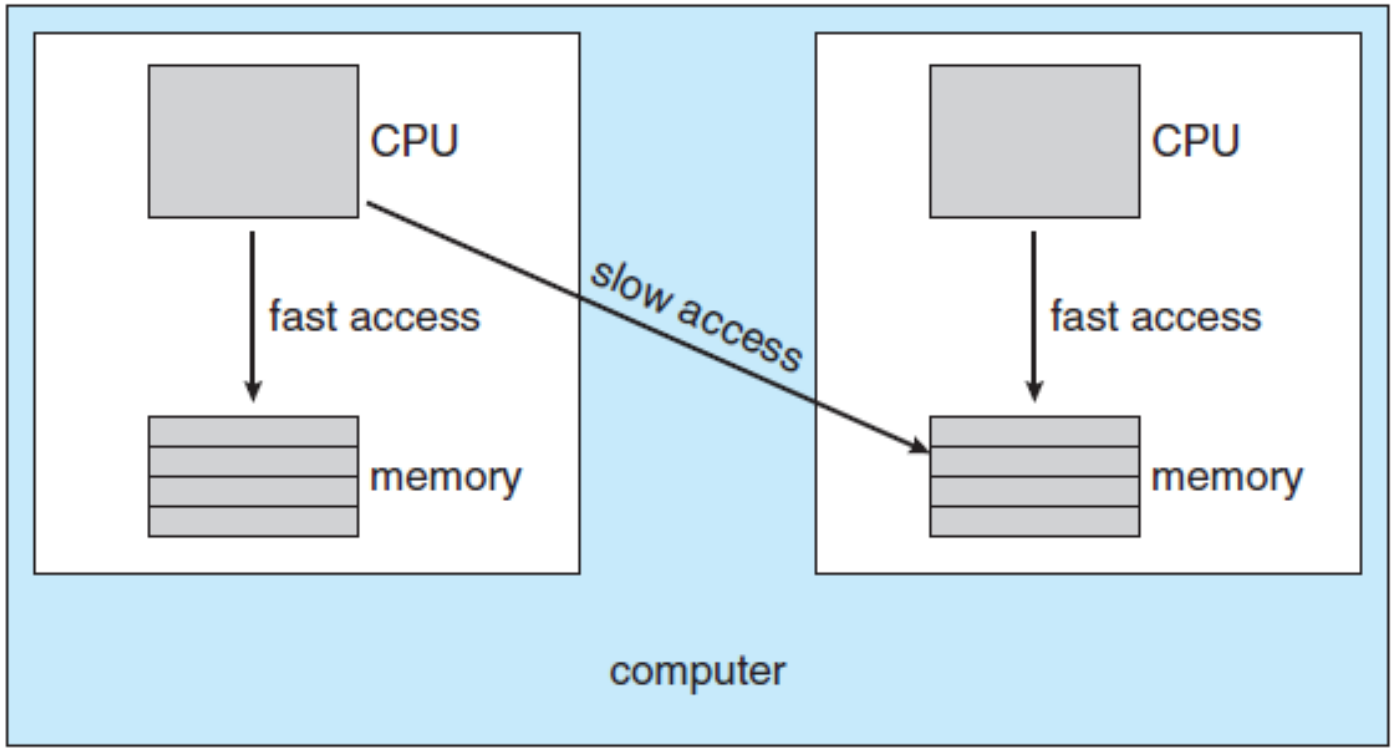

Figure 6.9 NUMA and CPU scheduling.

Load Balancing

- **Load balancing** attempts to keep the **workload evenly distributed across all processors** in an SMP system.
- Otherwise, one or more processors may sit idle while other processors have high workloads, along with lists of processes awaiting the CPU.

- There are two general approaches to load balancing:
  - **push migration**
  - **pull migration**

With **push migration**, a specific task periodically checks the load on each processor and—if it finds an imbalance—evenly distributes the load **by moving (or pushing) processes from overloaded** to idle or less-busy processors.

Pull migration occurs when an **idle processor pulls a waiting task** from a busy processor.

Push and pull migration need not be mutually exclusive and are in fact often implemented in parallel on load-balancing systems.

Interestingly, **load balancing often counteracts the benefits of processor affinity**

Multicore Processors

Multiple processor cores placed on the same physical chip, resulting in a **multicore processor.**

memory stall

- when a processor accesses memory, it spends a significant **amount of time waiting** for the data to become available. This situation, known as a **memory stall**, may occur for various reasons, such as a cache miss (accessing data that are not in cache memory).

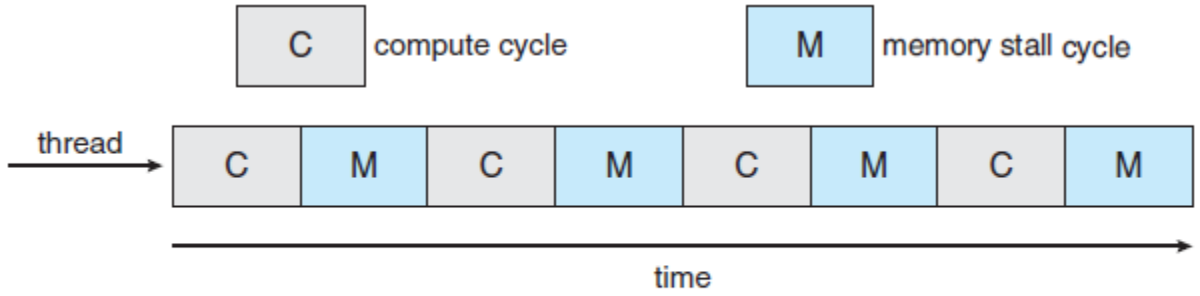

Figure 6.10 Memory stall.

- To remedy this situation, many recent hardware designs have implemented multithreaded processor cores in which **two (or more) hardware threads are assigned to each core**.
- That way, if one thread stalls while waiting for memory, the core can switch to another thread.

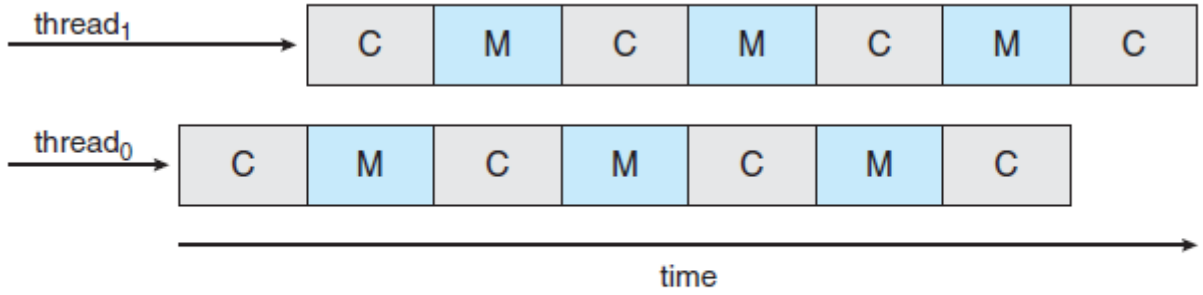

Figure 6.11 Multithreaded multicore system.

There are two ways to multithread a processing core:

coarse grained

fine-grained multithreading

- With **coarse-grained multithreading**, a thread executes on a processor until a **long-latency event** such as a **memory stall occurs**. Because of the delay caused by the long-latency event, the processor must switch to another thread to begin execution.
- However, the **cost of switching between threads is high**, since the instruction pipeline must be flushed before the other thread can begin execution on the processor core.
- Once this new thread begins execution, it begins filling the pipeline with its instructions.

- **Fine-grained (or interleaved) multithreading** switches between threads at a much **finer level of granularity**—typically at the **boundary of an instruction cycle**.
- However, the architectural design of fine-grained systems includes logic for thread switching. As a result, the **cost of switching between threads is small**.

Two different levels of scheduling

- On one level are the scheduling decisions that must be made by the operating system as it chooses **which software thread to run on each hardware thread** (logical processor).
- A second level of scheduling specifies how each core decides **which hardware thread to run**.

REFERENCES

TEXT BOOKS:

T1 Silberschatz, Galvin, and Gagne, “Operating System Concepts”, Ninth Edition, Wiley India Pvt Ltd, 2009.)

T2. Andrew S. Tanenbaum, “Modern Operating Systems”, Fourth Edition, Pearson Education, 2010

REFERENCES:

R1 Gary Nutt, “Operating Systems”, Third Edition, Pearson Education, 2004.

R2 Harvey M. Deitel, “Operating Systems”, Third Edition, Pearson Education, 2004.

R3 Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, “Operating System Concepts”, 9th Edition, John Wiley and Sons Inc., 2012.

R4. William Stallings, “Operating Systems – Internals and Design Principles”, 7th Edition, Prentice Hall, 2011


THANK
YOU!