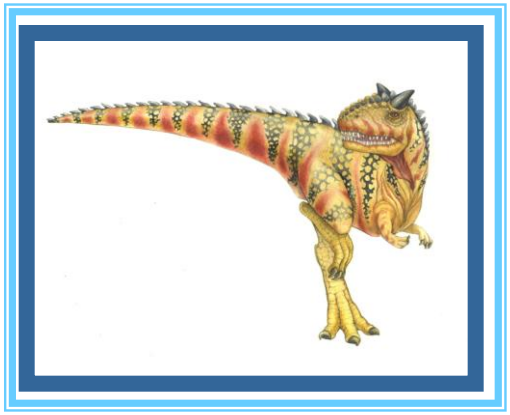
Chapter 5: Process Scheduling



Chapter 5: Process Scheduling

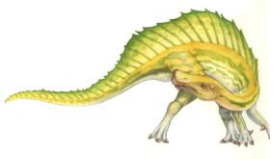
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling

Thread Scheduling

- Distinction between user-level and kernel-level threads
- Contention Scope:
  - **Process-Contention Scope (PCS) :** thread library schedules user-level threads to run on an available LWP,
    - the scheduling competition for the CPU takes place among threads within the same process.
    - In this scope the thread is not actually running on a CPU
    - It is schedule according to priority – the scheduler selects the runnable thread with the highest priority to run.
  - **System-Contention Scope (SCS) :** Kernel thread scheduled onto available physical CPU
    - The scheduling competition among all threads in system

Thread Scheduling (cont.)

- On systems implementing the *many-to-one* and *many-to-many* models, schedule threads using **PCS** and **SCS**
- On systems implementing the *one-to-one* model, schedule threads using only **SCS**



Pthread Scheduling

- API allows specifying either PCS or SCS during thread creation.
- Pthreads identifies the following contention scope values:
  - PTHREAD_SCOPE_PROCESS schedules threads using PCS scheduling
  - PTHREAD SCOPE SYSTEM schedules threads using SCS scheduling.

Multiple-Processor Scheduling

- **Load sharing** become possible when multiple CPUs are available and CPU scheduling become more complex

- On Multiprocessing System which the processor are identical **(Homogeneous processors),** we can use any available processor to run any process in the queue. (but there is some limitation)

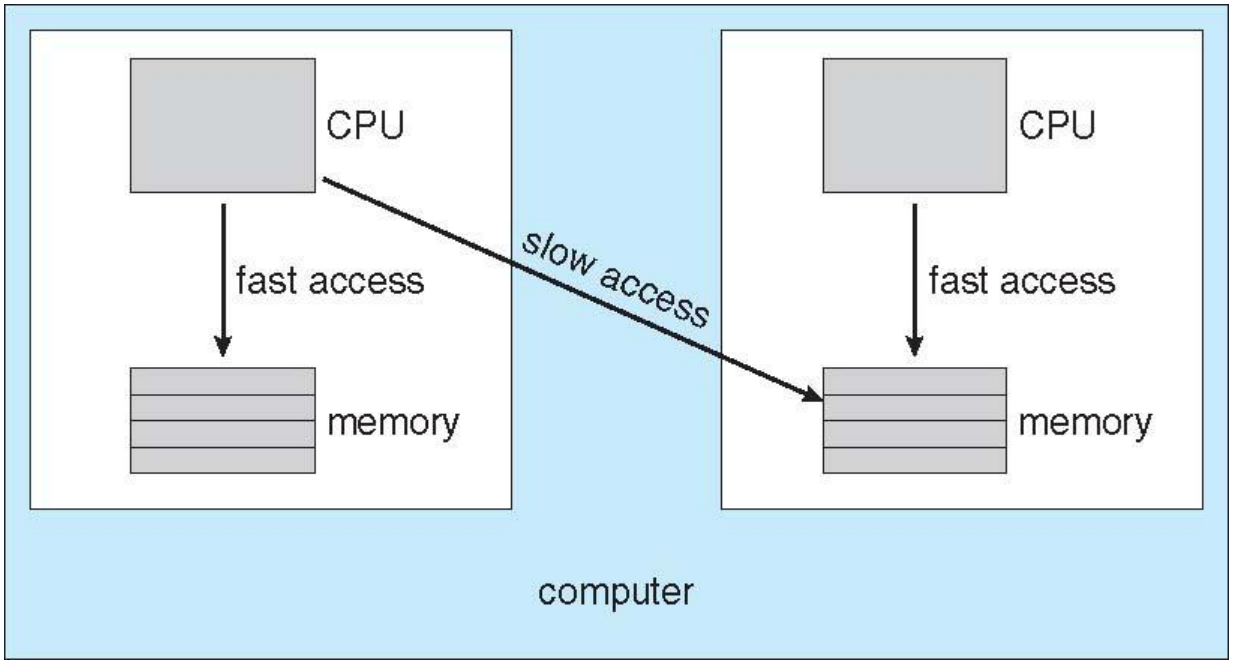
Multiple-Processor Scheduling (cont.)

- Approaches to Multiple-Processor Scheduling
  - **Asymmetric multiprocessing** – only one processor accesses the system data structures, alleviating the need for data sharing
  - **Symmetric multiprocessing (SMP)** – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
    - All modern operating system support SMP, including Windows XP, Windows 2000, Solaris, Linux, and Mac OS X.

Multiple-Processor Scheduling (cont.)

- **Processor affinity** – process has affinity for processor on which it is currently running
  - **Soft affinity**: it is possible for a process to migrate between processors. ex : Solaris
  - **Hard affinity:** not allowing a process to migrate to other processors. ex : Linux

NUMA and CPU Scheduling



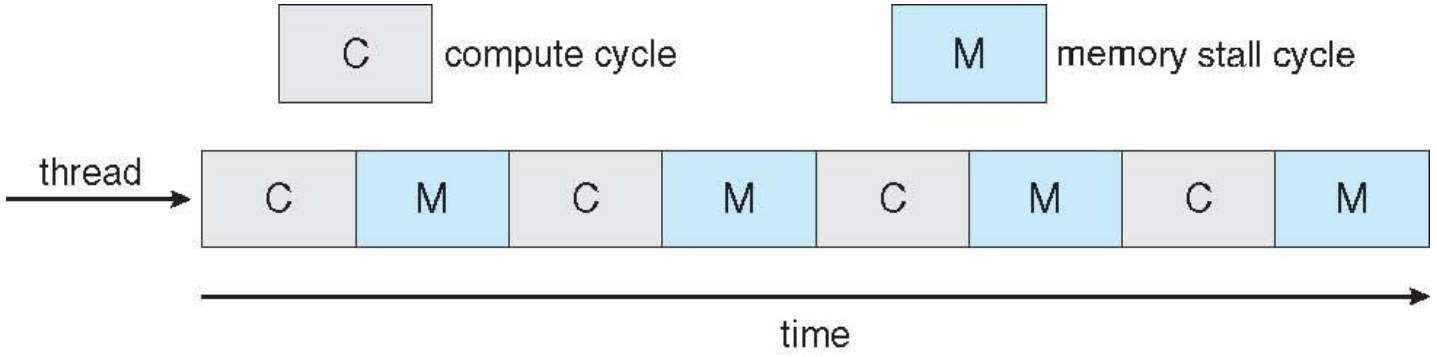
Load Balancing

- Load balancing attempts to keep the workload evenly distributed across all processors in an SMP system.
- It is necessary on systems where each processor has its own private queue of processes to execute.
- The load balancing Approaches are:
  - **Push migration** : 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** : an idle processor pulls a waiting task from a busy processor.
  - ➢ The push and pull migration are often implemented in parallel on load-balancing systems.

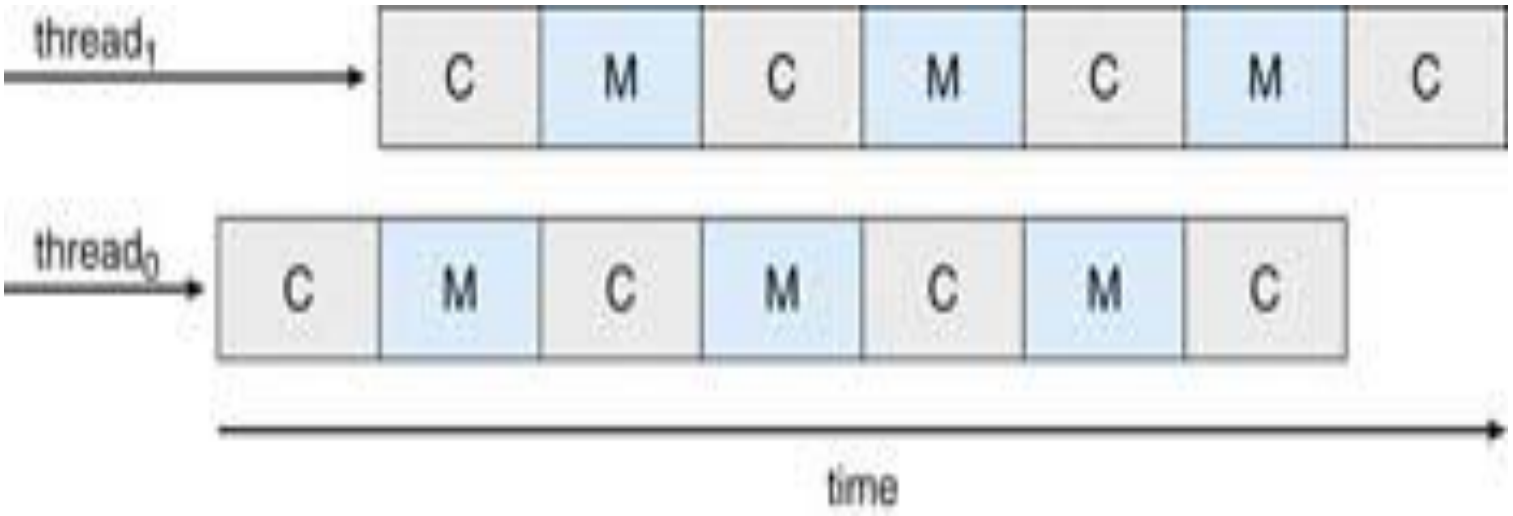
Multicore Processors

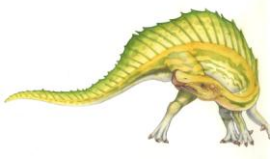
- Recent trend to place multiple processor cores on same physical chip
- Faster and consume less power
- Multiple threads per core also growing
  - When a processor accesses memory, it spends a significant amount of time waiting for the data to become available. This situation known **memory stall**
  - To remedy this situation, many hardware designs have implemented multithreaded processor cores in which two (or more) hardware threads are assigned to each core.
  - So, if one thread stalls while waiting for memory, the core can switch to another thread. (see figure 5.11 in text book)

Memory stall



Multithreaded Multicore System





End of Chapter 5