CS420: Operating Systems

Multiprocessor Scheduling

James Moscola
Department of Physical Sciences
York College of Pennsylvania

Multiple-Processor Scheduling

- **CPU scheduling more complex when multiple CPUs are available**
  - Most current general purpose processors are multiprocessors (i.e. multicore processors)
  - No single ‘best’ solution to multiple-processor scheduling

- **A multicore processor typically has two or more homogeneous processor cores**
  - Because the cores are all the same, any available processor can be allocated to any process in the system

Approaches to Multiple-Processor Scheduling

- **Asymmetric multiprocessing**
  - All scheduling decisions, I/O processing, and other system activities handled by a single processor
  - Only one processor accesses the system data structures, alleviating the need for data sharing

- **Symmetric multiprocessing (SMP)**
  - 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
  - Currently, most common approach to multiple-processor scheduling

Processor Affinity

- **Processor affinity – process has an affinity for processor on which it is currently running**
  - Leaving a process on a single processor prevents the need to invalidate and repopulate caches
  - Best to avoid migrating processes from one processor to another

- **Different variations of processor affinity**
  - Soft affinity - OS will attempt to keep a process running on the same processor, however there is no guarantee
  - Hard affinity - OS will force a process to continue running on the same processor

Load Balancing

- **In SMP systems, it is necessary to keep the workload balanced among all processors to fully utilize the available processors**
  - Typically only necessary in systems where each processor has its own queue
  - In systems with a single queue, any processor can pull tasks from a common ready queue
    - Most modern operating systems assign a private queue to each processor (DOH!)
- **Two approaches to load balancing**
  - Push migration - a specific task periodically checks the load on each processor and rebalances the load if necessary
  - Pull migration - an idle processor can pull waiting tasks from a busy processor
- **Load balancing may negate the benefits of processor affinity**

Multicore Processors

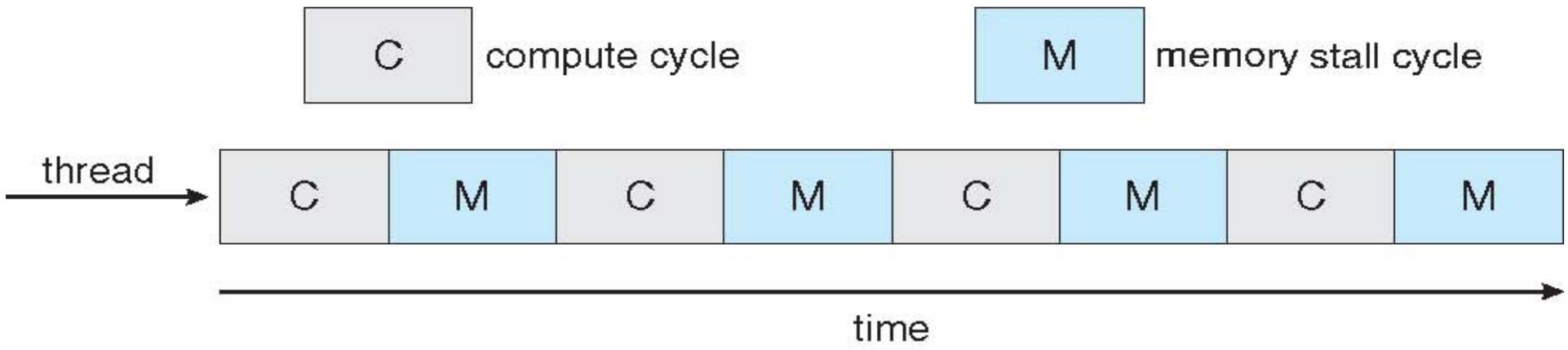
- **Recently, the architectural trend is to place multiple processor cores on same physical chip (this will likely be true for the foreseeable future)**
  - Faster and consume less power than system where each processor has its own physical chip

- **Multiple threads per core also growing (Intel Hyper-Threading)**
  - Duplicate thread state at the architectural level, but not execution resources
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens

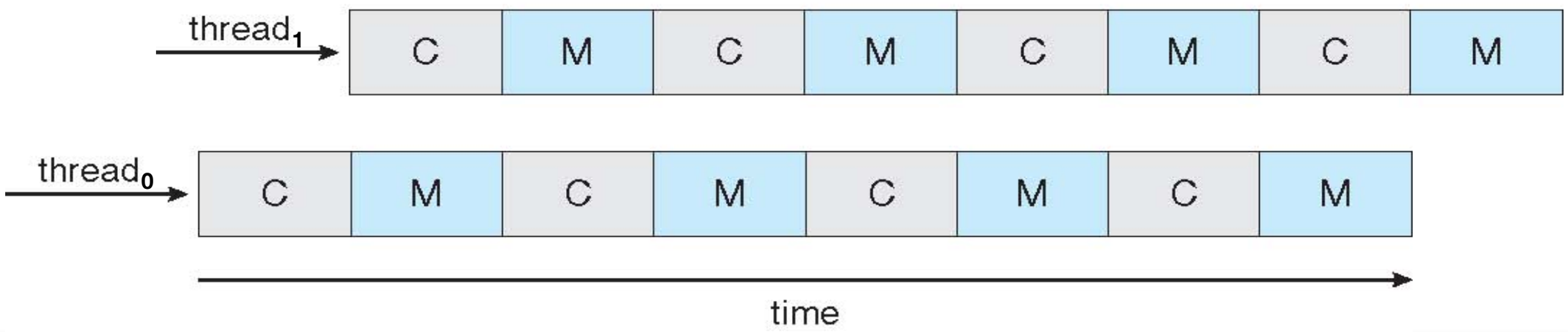
- **A quad-core processor with two threads per core appears as 8 processors to the operating system**
  - 4 physical processors x 2 hardware threads = 8 logical processors

Multithreaded Multicore System

- **A processor core with a single hardware thread doesn't accomplish any work during a memory stall**



- **A processor core with multiple hardware threads can work on another thread while waiting for a memory read to complete**



Scheduling Multithreaded Multicore Systems

- **Two levels of scheduling must take place**
  - (1) Operating system is still scheduling tasks based on its scheduling algorithms
  - (2) Second level of scheduling decides which hardware threads to run
    - Coarse-grained multithreading - a thread executes on a processor until a long-latency event occurs (i.e. reading from memory)
    - Fine-grained multithreading - May swap between hardware threads between instructions

Algorithm Evaluation

- **How to select CPU-scheduling algorithm for an OS?**

- **Determine criteria, then evaluate algorithms**
  - Not all systems will have the same criteria

- **A variety of approaches, each with its own pros and cons**
  (1) Deterministic Model
  (2) Queueing Models
  (3) Simulations
  (4) Implementation

(1) Deterministic Modeling

- **Takes a particular predetermined workload and defines the performance of each algorithm for that workload**
- **Type of analytic evaluation**
- **Pros**
  - Simple and fast
  - Provides exact numbers to enable comparison between algorithms
- **Cons**
  - Requires exact numbers for input and results only apply to those exact input data

(2) Queueing Models

- **Describes the arrival of processes, and CPU and I/O bursts probabilistically**
  - Distributions can be measured from the real system
  - Can computes average throughput, utilization, waiting time, etc.
- **Computer system described as network of servers, each with queue of waiting processes**
  - Knowing arrival rates and service rates, can compute utilization, average queue length, average wait time, etc.
- **Pros**
  - Useful in comparing scheduling algorithms
- **Cons**
  - To simplify analysis, distributions are often defined in simplified, but unrealistic ways
  - Accuracy of results may be questionable

(3) Simulations

- **Simulations can provide accurate evaluation of scheduling algorithms**
  - Programmed model of computer system
  - Data to drive simulation gathered via
    - Random number generator according to probabilities
    - Distributions defined mathematically or empirically
    - Trace tapes record sequences of real events in real systems
- **Pros**
  - Can provide a very accurate representation of how various algorithms perform (especially when using trace tapes)
- **Cons**
  - Can be very time consuming

(4) Implementation

- **Forgo simulations and estimations, just implement the new scheduler and test it in real systems**

- **Pros**
  - The most accurate approach for comparing algorithms
- **Cons**
  - High cost
  - Environments vary