Preview

- Multi-Processor Scheduling
  - Approaches to Multiple Processor Scheduling
  - Scheduling for Multicore processors
    - Fine Grained multithreading
    - Coarse-Grained multithreading
    - Load Balancing

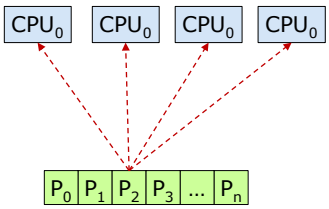
Multi-Processor Scheduling

- Traditional term of a multiprocessor systems: A system with multiple physical processors with single-core CPU.
- Evolved term of multiprocessor systems:
  - Multicore CPUs - A integrated circuit with two or more separate processing units, called cores, each of which reads and executes program instructions.
  - Multithreaded cores (a core with two or more hardware threads)
  - NUMA(Non-Uniform-Memory-Access) systems
  - Heterogeneous multiprocessing
- If multiple CPUs are available (multiple threads may run in parallel), scheduling issues become correspondingly more complex.

Multi-Processor Scheduling
(Approaches to Multiple Processor Scheduling)

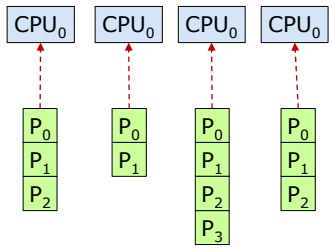
- Asymmetric multiprocessing
  - OS run on a special CPU (master server CPU) and other processors execute user code.
  - Since master server need control all jobs run on different general CPU's, there might be possible bottleneck where overall system performance may be reduced.
- Symmetric Multiprocessing (SMP)
  - Each CPU is self-scheduling.
  - When a CPU become available, CPU scheduler check the ready queue and select a process (thread) to run.
  - Two possible strategies
    - One common ready queue for all CPU's
    - Ready queue for each CUP

Multi-Processor Scheduling
(Approaches to Multiple Processor Scheduling)



One common ready queue for all CPU cores

Possible race condition:
- if more than one CPU are available, a process (or thread) might run on more than one CPU cores



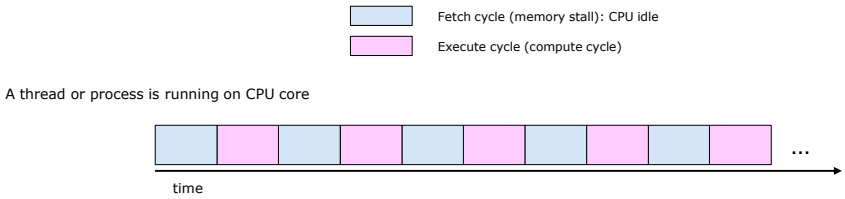
A ready queue for each CPU core

Consideration:
- Balancing work loads between CPU cores

Multi-Processor Scheduling
(Multicore Processors)

- Instead of multiple CUPs in a system, most contemporary computer use multicore processor.
- Each core maintains its state and appears to OS to be separate logical CPU to run processes (or threads) in parallel.
- SMP (Symmetric Multiprocessing) systems that use multicore processors are faster and consume less power than system with independent CPUs with single core.
- Multicore processors may complicate scheduling issues.
- A processor need wait a significant amount time for the data from Memory (memory stall). Memory stall can be alleviated by using cache memory.
  - Memory Stall – processors operate much faster than memory.
  - Von Newmann Bottleneck – processor operate much faster than fetch cycle

Multi-Processor Scheduling
(Multicore Processors)

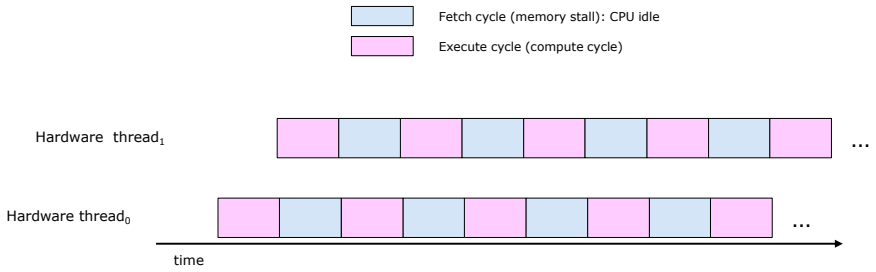


Memory Stall

Multi-Processor Scheduling
(Multicore Processors)

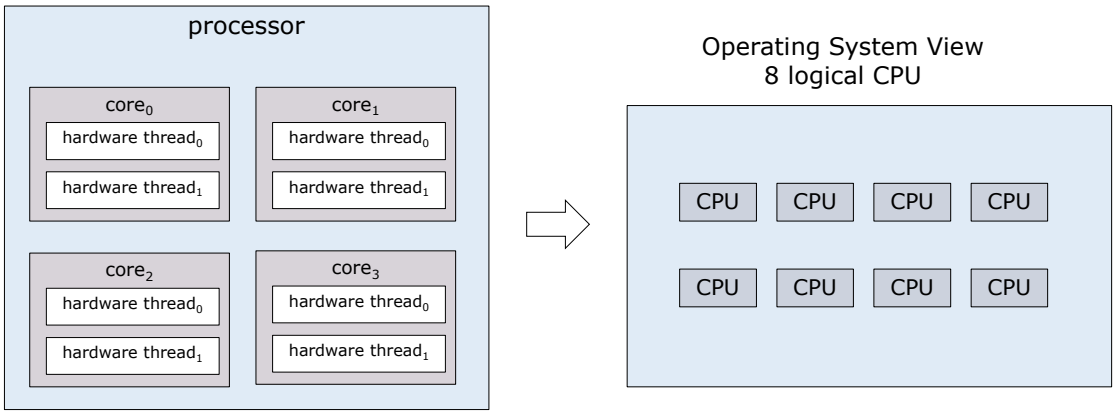
- Solution for memory stall : Multithreaded Processing Cores
  - A hardware designs have implemented multithreaded processing cores where two (or more) hardware threads are assigned to each core. If one hardware thread is waiting for memory, the core can switch to another hardware thread.
  - On a dual-threaded processing core, execution of hardware thread 0 and thread 1 are interleaved (next page figure).
  - Each hardware thread maintains its architectural state, such as program counter and register set, and thus appears as a logical CPU that is available to run a software thread – known as chip multithreading (CMT)

Multi-Processor Scheduling
(Multicore Processors)



Multithreaded Multicore System

Multi-Processor Scheduling
(Multicore Processors)



Chip Multithreading:
4 computing cores with two hardware threads for each core

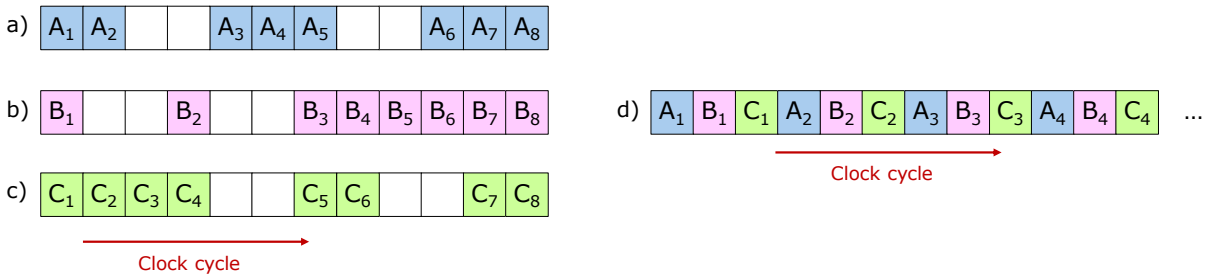
Multi-Processor Scheduling
(Multicore Processors)

- Intel processors use the term hyper-threading (also known as simultaneous multithreading or SMT) to describe assigning multiple hardware threads to a single processing core.
- Example of multithreaded multicore system
  - Intel i7 (has 4 or 6 cores): assign two hardware threads to a single core (support 8 or 12 logical CPUs).
  - Oracle Sparc M7 (8 cores): assign eight hardware threads to a single core (support 64 logical CPUs).
- Two Types of Multithread processing core
  - Coarse-grained multithreading
  - Fine-Grained Multithreading

Multi-Processor Scheduling
(Multicore Processors: Fine-grained multithreading)

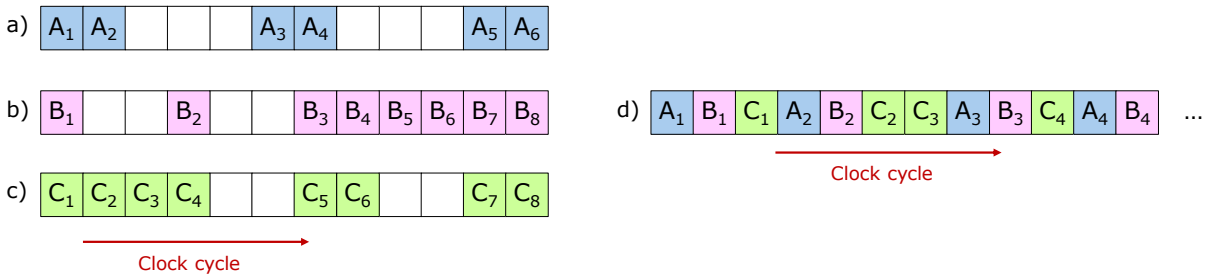
- Fine-grained multithreading: switching among threads happens at each instruction, independently from the fact that the thread instruction has caused a cache miss.
  - Instructions "scheduling" among threads obeys a round robin policy, and the CPU must carry out the switch with no overhead, since overhead cannot be tolerated.
  - If there is a sufficient number of threads, it is likely that at least one is active (not stalled), and the CPU can be kept running

Multi-Processor Scheduling
(Multicore Processors: Fine-grained multithreading)



- (a)-(c) three threads and associated stalls (empty slots).
- (d) Fine-grained multithreading.
- Each slot is a clock cycle, and we assume for simplicity that each instruction can be completed in a clock cycle, unless a stall happens

Multi-Processor Scheduling
(Multicore Processors: Fine-grained multithreading)



- (a)-(c) three threads and associated stalls (empty slots).
- (d) Fine-grained multithreading.
- Each slot is a clock cycle, and we assume for simplicity that each instruction can be completed in a clock cycle, unless a stall happens

Multi-Processor Scheduling
(Multicore Processors: Fine-grained multithreading)

- With fine-grained multithreading in a pipelined Architecture, if:
  - the pipeline has k stages,
  - there are at least k threads to be executed,
  - and the CPU can execute a thread switch at each clock cycle
- then there can never be more than a single instruction per thread in the pipeline at any instant, so there cannot be hazards due to dependencies, and the pipeline never stalls

Pipeline Operation
(Microprocessor Process)

- There are many different pipeline implementations and each can have a different number of pipeline stages.
- There are 5 pipeline stages in a RISC processor.
  - Instruction Fetch (IF)
  - Instruction Decode (ID)
  - Execute (EX)
  - Memory Access (MEM)
  - Writeback (WB)
- Each stage is executed by its own dedicated CPU functional unit and each takes one clock cycle to execute.
- Pipeline design can improve the performance of instructions.

Pipeline Operation
(Microprocessor Process)

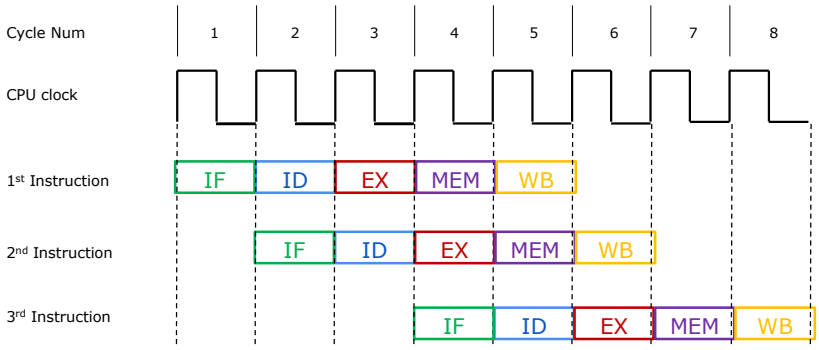
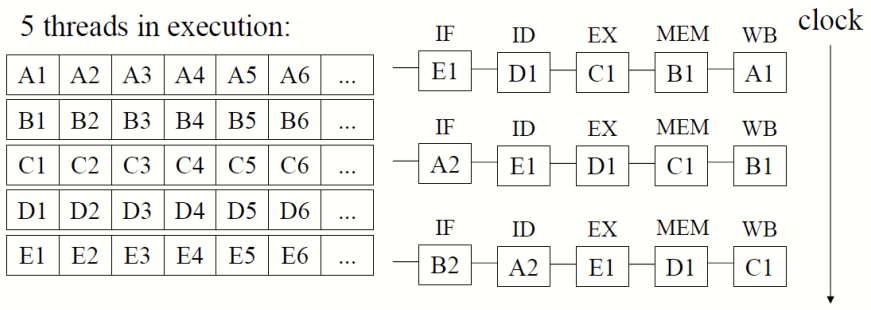


Fig. 3 Microprocessor instruction pipeline

Multi-Processor Scheduling

(Multicore Processors: Fine-grained multithreading)

- Fine-grained multithreading in a CPU with a 5-stage pipeline: there are never two instructions of the same thread concurrently active in the pipeline.



Multi-Processor Scheduling

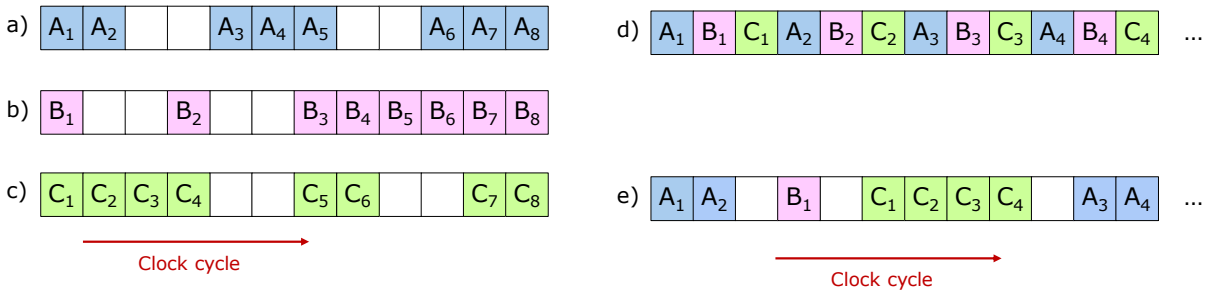
(Multicore Processors: Fine-grained multithreading)

- Besides requiring an efficient context switch among threads, threads fine-grained scheduling at each instruction slows down a thread even when the thread could go on since it is not causing a stall.
- Furthermore, there might be fewer threads than stages in the pipeline (actually, this is the usual case), so keeping the CPU busy is no easy matter.

Multi-Processor Scheduling
(Multicore Processors: Coarse-grained multithreading)

- Coarse-grained multithreading
  - A hardware thread executes on a core until a long-latency event (memory stall) occurs.
  - For long-latency event, The core must switch to another hardware thread to begin execution.
  - 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.

Multi-Processor Scheduling
(Multicore Processors: Coarse-grained multithreading)



- (a)-(c) three threads and associated stalls (empty slots).
- (d) Fine-grained multithreading.
- (e) Coarse-grained multithreading.
- Each slot is a clock cycle, and we assume for simplicity that each instruction can be completed in a clock cycle, unless a stall happens

Multi-Processor Scheduling

(Multicore Processors: Fine vs Coarse-grained multithreading)

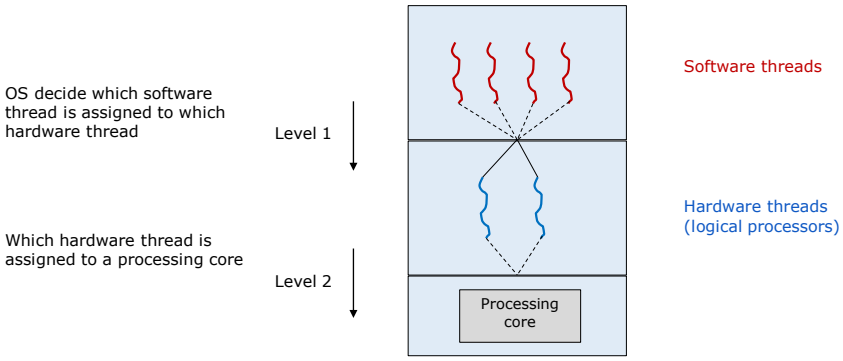
- In the previous figure, fine-grained multithreading seems to work better, but this is not always the case.
- Specifically, a switch among threads cannot be carried out without any waste in clock cycles.
- So, if the instructions of the threads do not cause stalls frequently, a coarse-grained scheduling can be more convenient than a fine-grained one, where the context switch overhead is paid at each clock cycle (this overhead is very small, but never null).

Multi-Processor Scheduling

(Multicore Processors)

- To share resources of the physical core (such as caches and pipelines) between hardware threads in a multithreaded multicore processor, the processor requires two levels of scheduling.
- On one level, operating system decide which software thread to run on which hardware thread (logical CPU). The operating system may choose any scheduling algorithm for decision.
- A second level of scheduling specifies how each core decides which hardware thread to run.
  - Round-robin – used in UltraSPARC T3
  - Dynamic urgency value – each hardware thread is assigned with a urgency value (0 ~7). Select a hardware thread based on the value. - Intel Itanisum dual-core with two hardware threads per core.

Multi-Processor Scheduling
(Multicore Processors)



Two levels of scheduling

Multi-Processor Scheduling
(Multicore Processors: Load Balancing)

- On Symmetric multiprocessor (SMP) systems, it is important to keep the workload balanced among all processors to fully utilize the benefits of having more than one processor.
  - On systems with a common ready queue, load balancing is unnecessary, because once a processor becomes available, it immediately extracts a runnable thread from the common ready queue.
  - On systems with a ready queue for each CPU core, need consider load balancing
- Two general approaches to load balancing
  - Push migration – a specific task checks the load on each processor and if it finds an unbalance, by moving threads from overloaded to ideal or less busy processors.
  - Pull migration - occurs when an idle processor pulls a waiting task from a busy processor.
  - Push and Pull migration can be implemented in parallel on load-balancing system

Multi-Processor Scheduling
(Multicore Processors: Processor Affinity)

- Consider a cache memory on a processor. When a thread has been running on a specific processor, the data most recently accessed populated the cache to use successive memory accesses.
- Now consider the thread migrates to another processor due to load balancing. 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 operating systems with SMP attempt to keep a thread running on the same processor and take advantage of a warm cache. (Processor affinity)

Multi-Processor Scheduling
(Multicore Processors: Processor Affinity)

- On systems with a common ready queue –
  - A thread may be selected for execution by any processor.
  - Thus, if a thread is scheduled on a new processor, that processor's cache must be repopulated.
- On systems with a ready queue for each CPU core –
  - A thread is always scheduled on the same processor and can therefore benefit from the contents of a warm cache (processor affinity free)