CPU Scheduling(cont'd)

Course Code: CSC 2209 Course Title: Operating Systems

Dept. of Computer Science
Faculty of Science and Technology

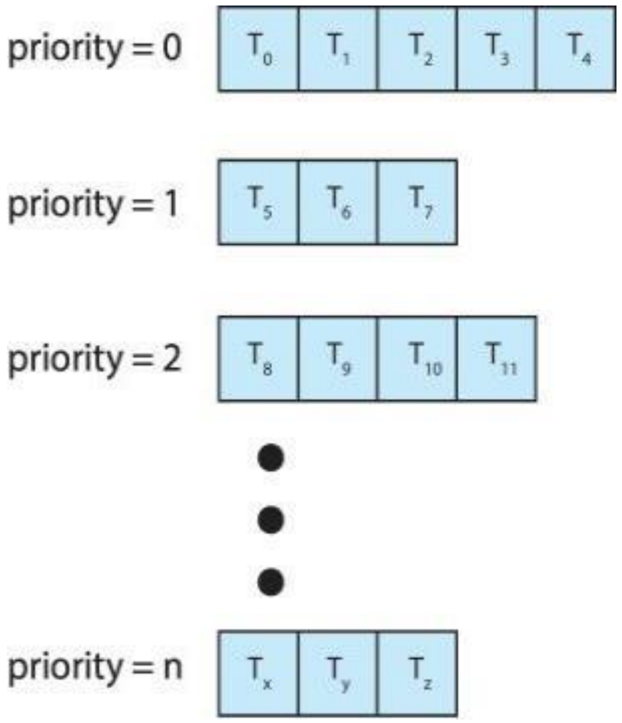
Lecturer No:	**07**	**Week No:**	**07**	**Semester:**	
Lecturer:	*Name & email*				

Lecture Outline

1. Thread Scheduling
2. Multi-Processor Scheduling
3. Real-Time CPU Scheduling
4. Operating Systems Examples
5. Algorithm Evaluation

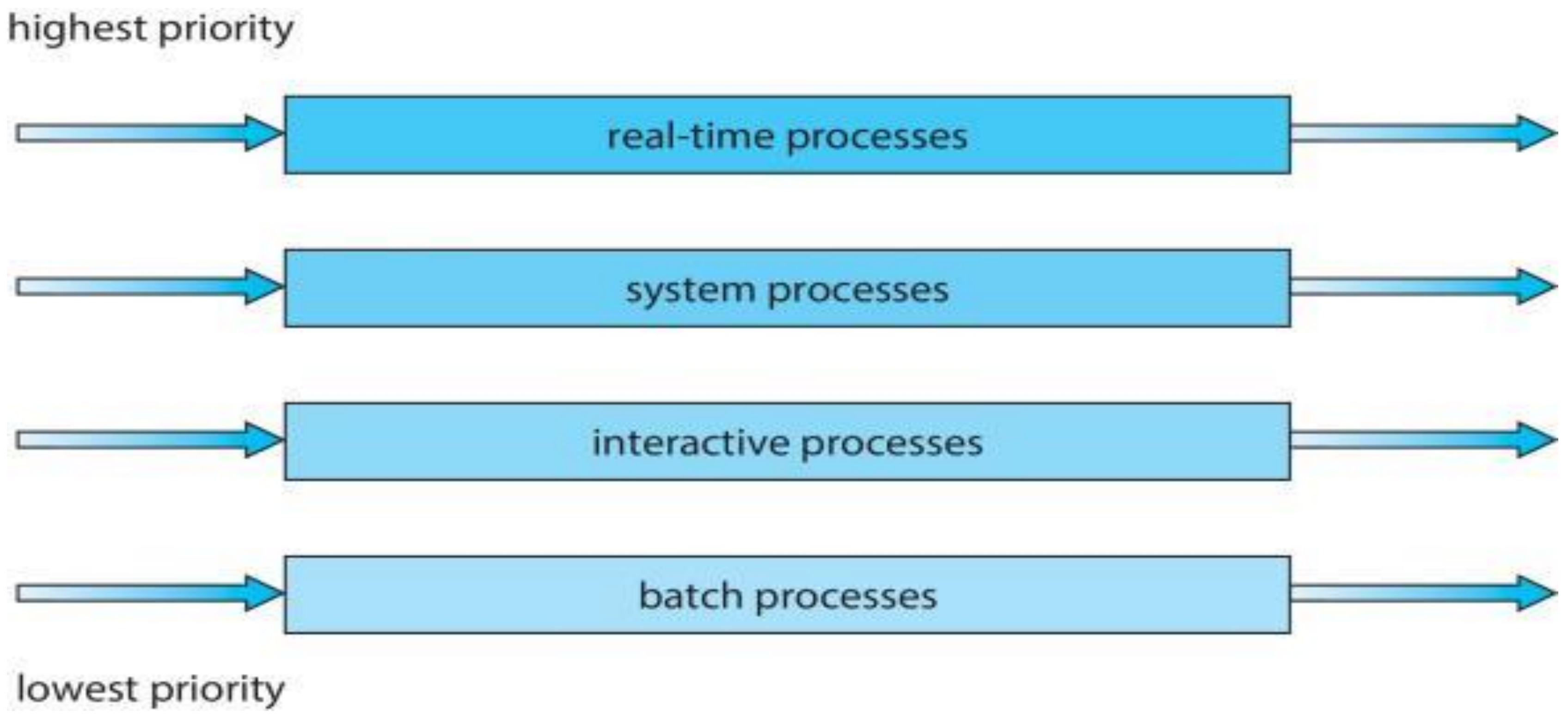
Multilevel Queue

- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!
- Priority 0 **ready queue** = RR
- Priority 1 ready queue = FCFS
- CPU Scheduling algo queue & process



Multilevel Queue

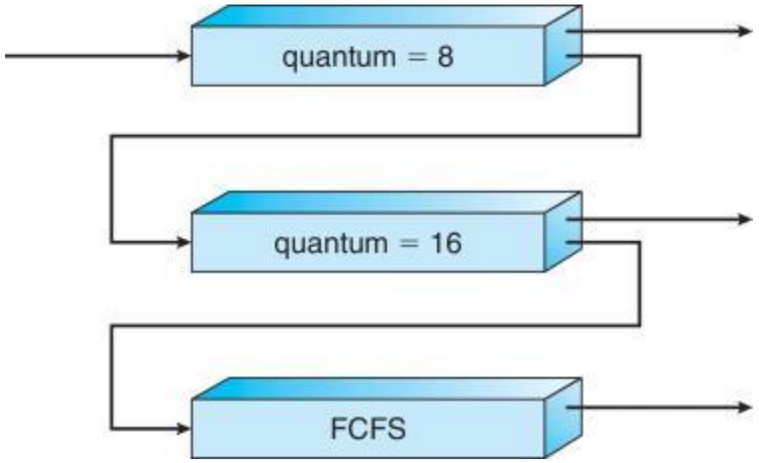
- Prioritization based upon process type



Multilevel Feedback Queue (MLFQ)

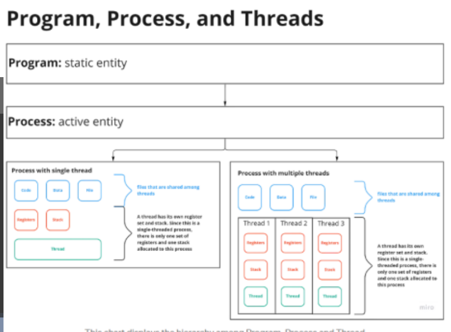
- A **process** can move between the various queues; **aging** can be implemented this way
- MLFQ scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue



- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0 which is served RR
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q_1
  - At Q_1 job is again served RR and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2 served FCFS.

Thread Scheduling


Program, Process, and Threads
Program: static entity
Process: active entity
Process with single thread
Code
Data
File
Registers
Stack
Thread
files that are shared among threads
A thread has its own register set and stack. Since this is a single-threaded process, there is only one set of registers and one stack allocated to this process
Process with multiple threads
Code
Data
File
Thread 1
Thread 2
Thread 3
Registers
Stack
Thread
files that are shared among threads
A thread has its own register set and stack. Since this is a single-threaded process, there is only one set of registers and one stack allocated to this process
miro

This chart displays the hierarchy among Program, Process and Thread

- Distinction between **user-level** and **kernel-level** threads
- When threads supported, threads scheduled, not processes
- Many-to-one and many-to-many models, thread library schedules **user-level threads** to run on **Light Weight Process** (LWP)
  - Known as **process-contention scope** (**PCS**) since scheduling competition is within the process
  - Typically done via priority set by programmer
- **Kernel thread** scheduled onto available CPU is **system-contention scope** (**SCS**) – competition among all threads in system

Pthread Scheduling

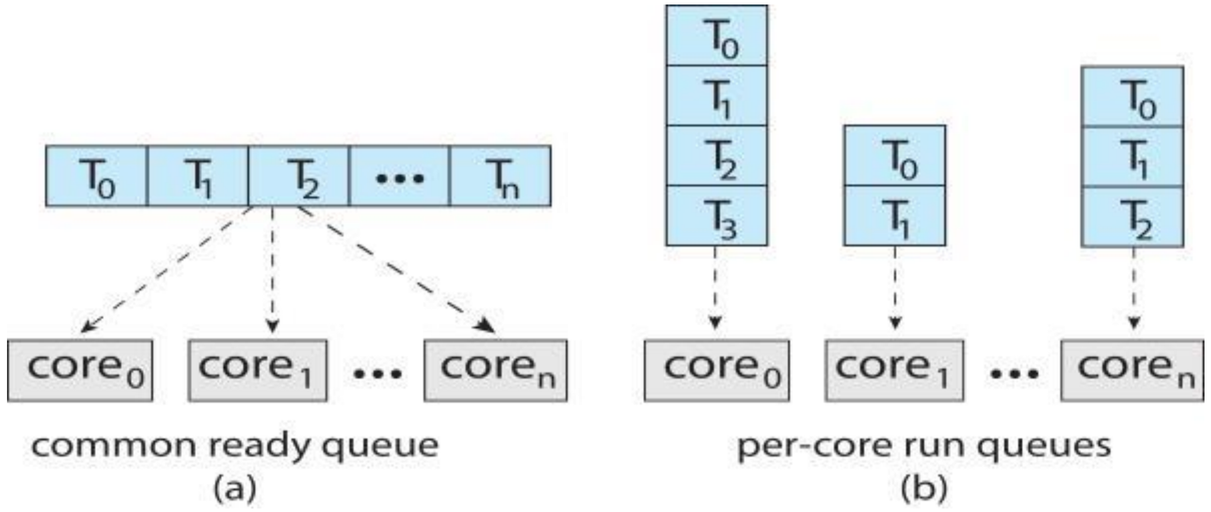
- API allows specifying either PCS or SCS during thread creation
  - PTHREAD_SCOPE_PROCESS schedules threads using PCS scheduling
  - PTHREAD_SCOPE_SYSTEM schedules threads using SCS scheduling
- Can be limited by OS – Linux and macOS only allow PTHREAD_SCOPE_SYSTEM

Multiple-Processor Scheduling

- CPU scheduling more complex when **multiple CPUs** are available
- Multiprocessing may be any one of the following **architectures**:
  - Multicore CPUs
  - Multithreaded cores
  - NUMA systems (Non-Uniform Memory Access Systems)
  - Heterogeneous multiprocessing

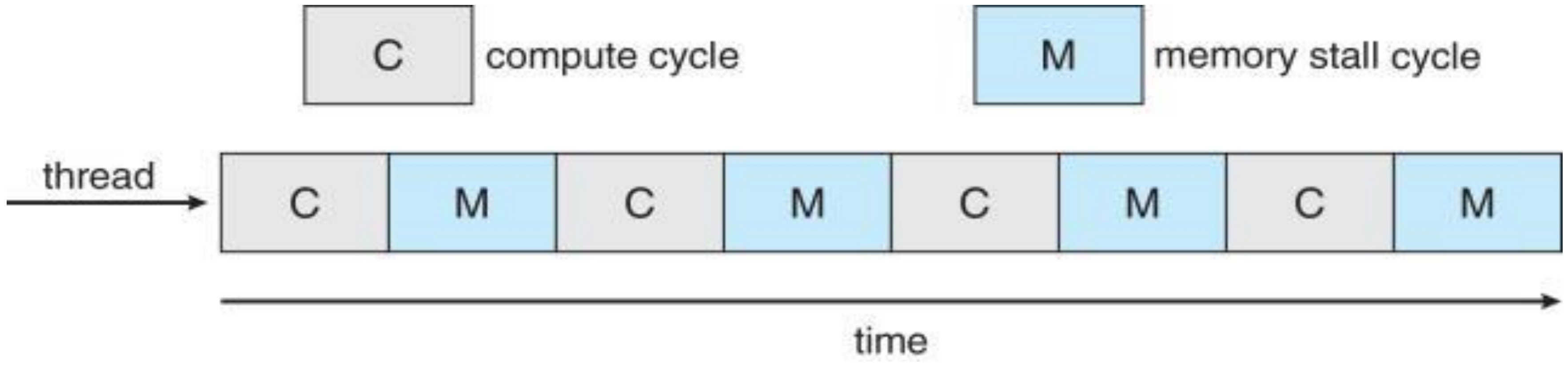
Multiple-Processor Scheduling

- **Symmetric multiprocessing** (SMP) is where each processor is self scheduling.
- All threads may be in a common ready queue (a)
- Each processor may have its own private queue of threads (b)



Multicore Processors

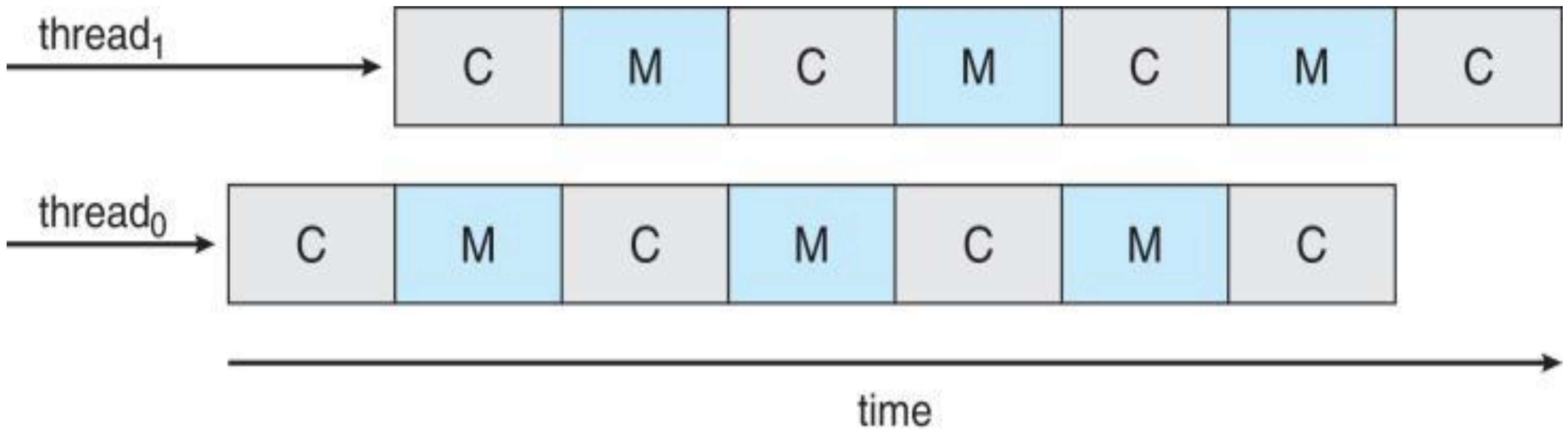
- Recent trend to place **multiple processor cores on same physical chip**
- Faster and consumes less power
- Multiple threads per core also growing
  - Takes advantage of **memory stall** to make progress on another thread while memory retrieve happens



Multithreaded Multicore System

Each core has > 1 **hardware threads**.

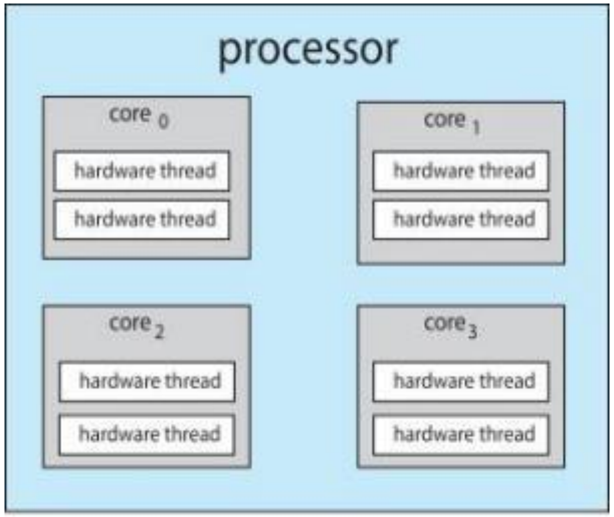
If one thread has a memory stall, switch to another thread!

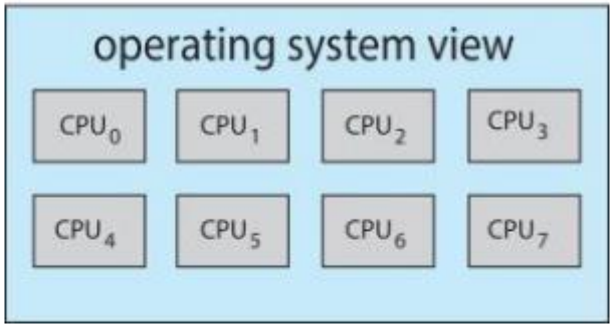


Multithreaded Multicore System

- **Chip-multithreading** (CMT) assigns each core multiple hardware threads. (Intel refers to this as **hyper-threading**.)

- On a **quad-core system** with 2 hardware threads per core, the operating system sees 8 logical processors.



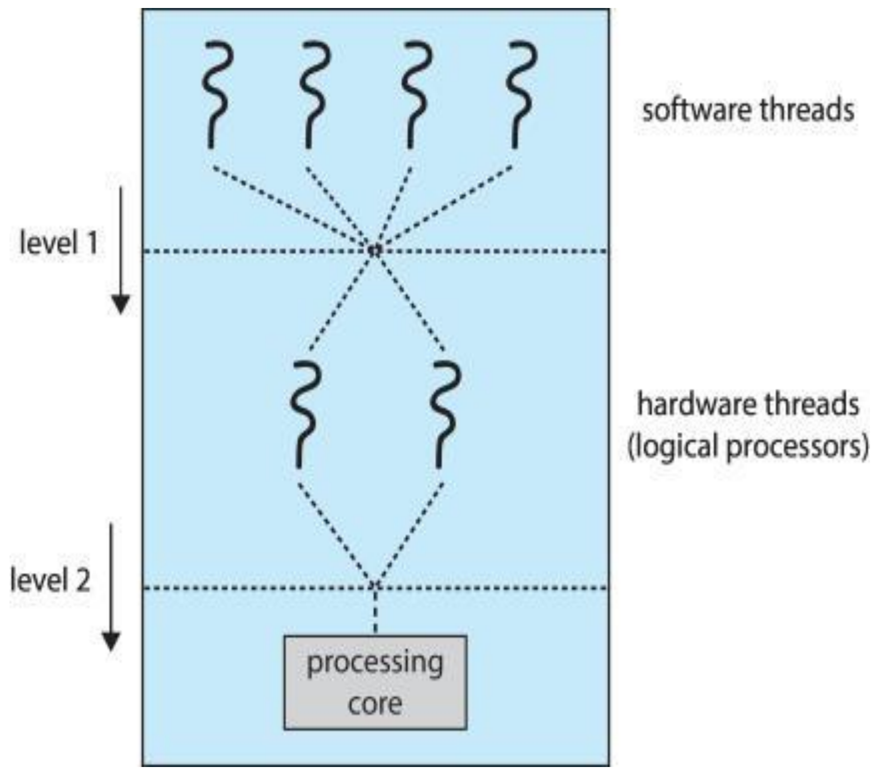


Multithreaded Multicore System

- **Two levels** of scheduling:

1. The operating system deciding which software thread to run on a logical CPU

2. Each core decides which hardware thread to run on the physical core.



Multiple-Processor Scheduling – Load Balancing

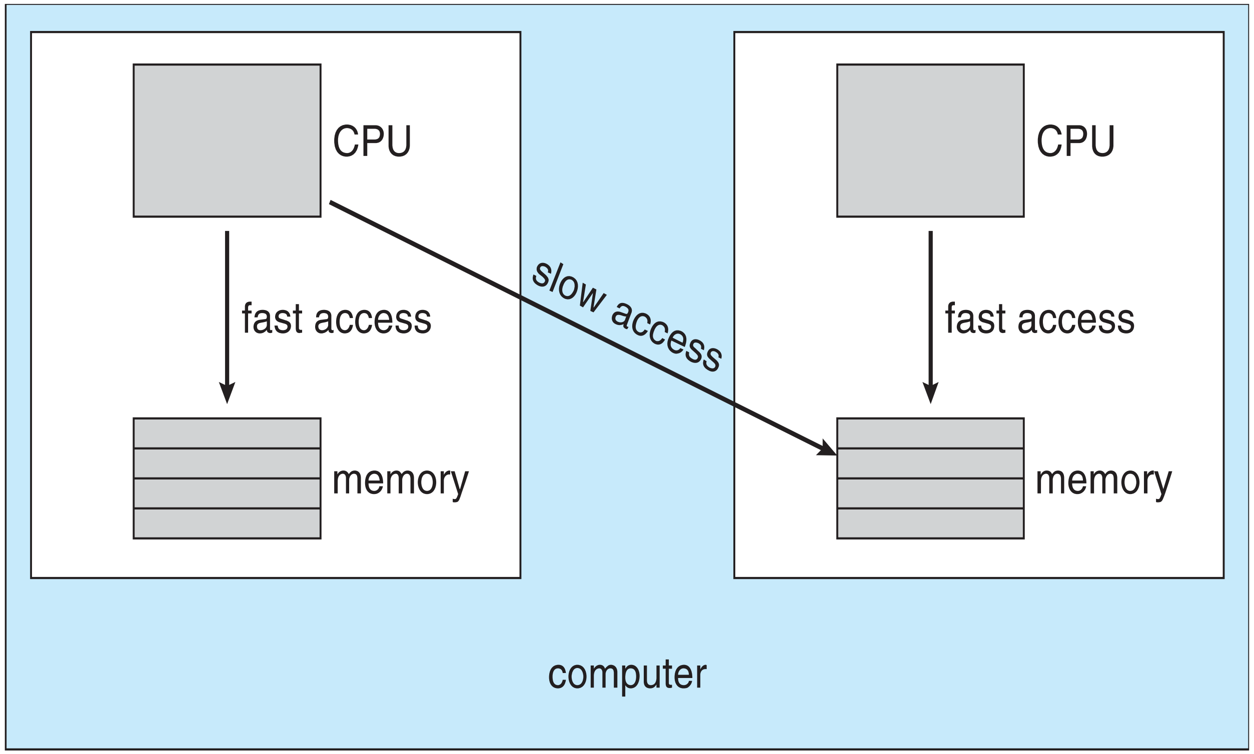
- If SMP, need to keep all CPUs loaded for efficiency
- **Load balancing** attempts to keep workload evenly distributed
  - **Push migration** – periodic task checks load on each processor, and if found, pushes task from overloaded CPU to other CPUs
  - **Pull migration** – idle processors pulls waiting task from busy processor

Multiple-Processor Scheduling – Processor Affinity

- ❑ When a thread has been running on one processor, the cache contents of that processor stores the memory accesses by that thread.
- ❑ We refer to this as a thread having affinity for a processor (i.e. "**processor affinity**")
- ❑ Load balancing may affect processor affinity as a thread may be moved from one processor to another to balance loads, yet that thread loses the contents of what it had in the cache of the processor it was moved off.
- ❑ **Soft affinity** – the operating system attempts to keep a thread running on the same processor, but no guarantees.
- ❑ **Hard affinity** – allows a process to specify a set of processors it may run on.

NUMA and CPU Scheduling

If the operating system is **NUMA-aware**, it will assign memory closes to the CPU the thread is running on.

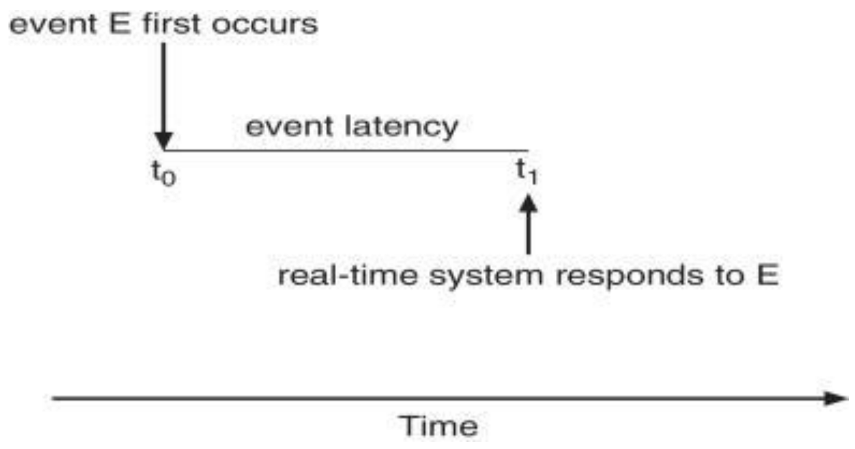


Real-Time CPU Scheduling

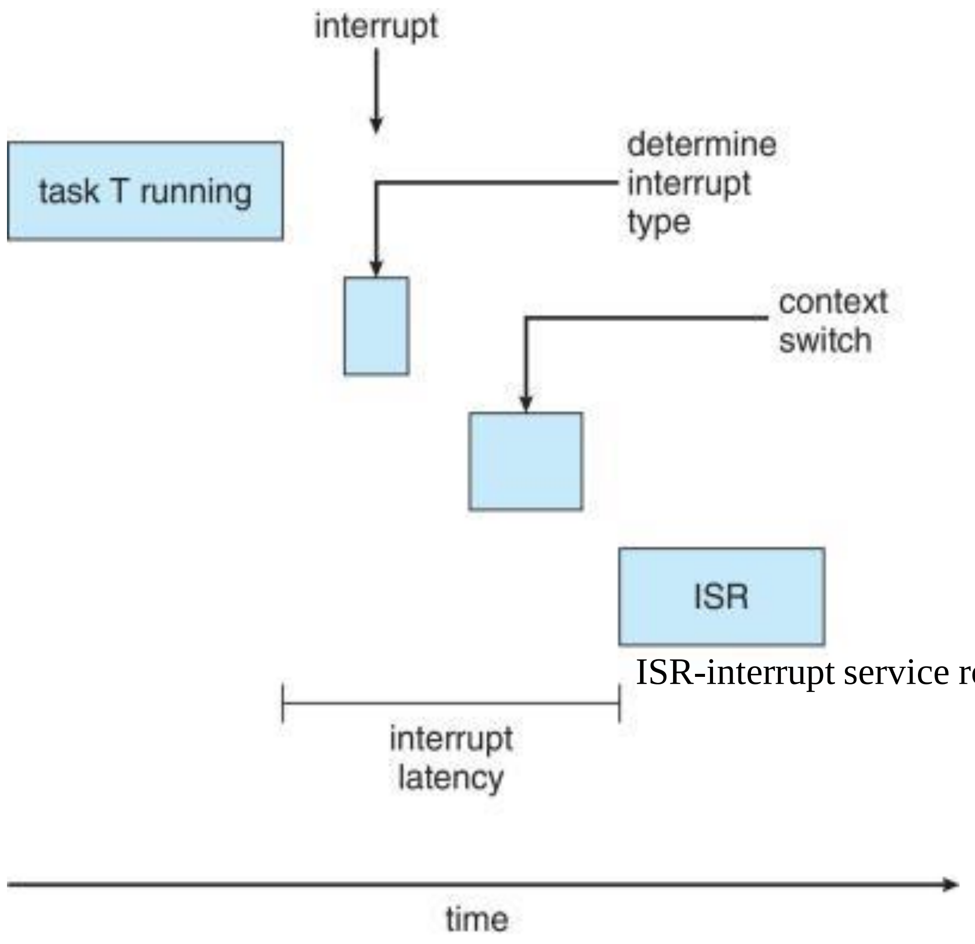
- Can present obvious challenges
- **Soft real-time systems** – Critical real-time tasks have the highest priority, but no guarantee as to when tasks will be scheduled
- **Hard real-time systems** – task must be serviced by its deadline

Real-Time CPU Scheduling

- **Event latency** – the amount of time that elapses from when an event occurs to when it is serviced.
- Two types of latencies affect performance
  - **Interrupt latency** – time from arrival of interrupt to start of routine that services interrupt
  - **Dispatch latency** – time for schedule to take current process off CPU and switch to another



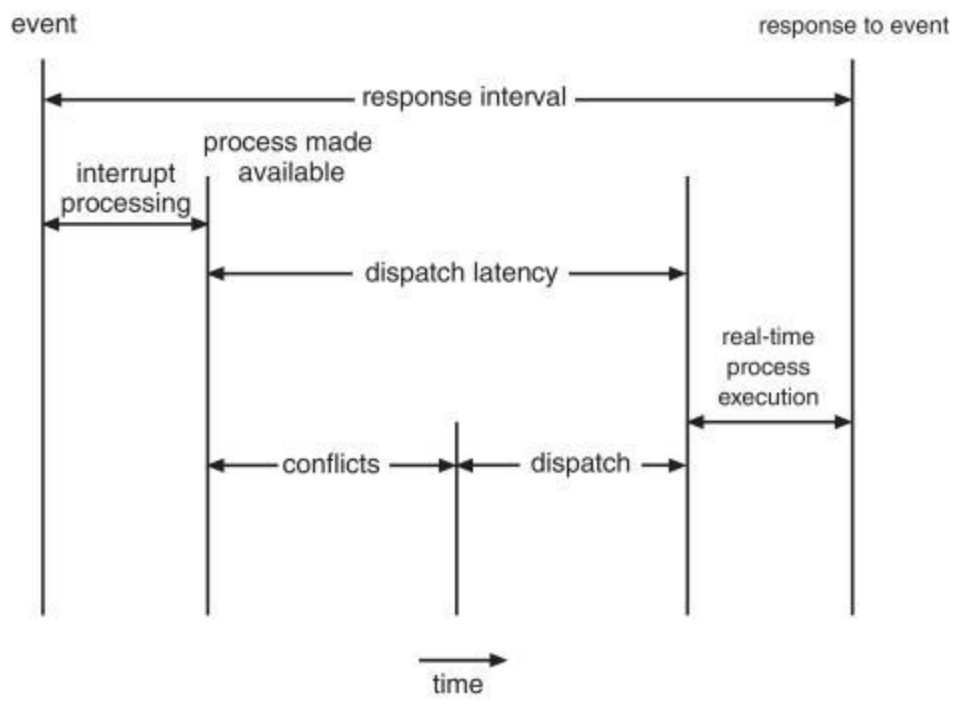
Interrupt Latency



ISR-interrupt service routine

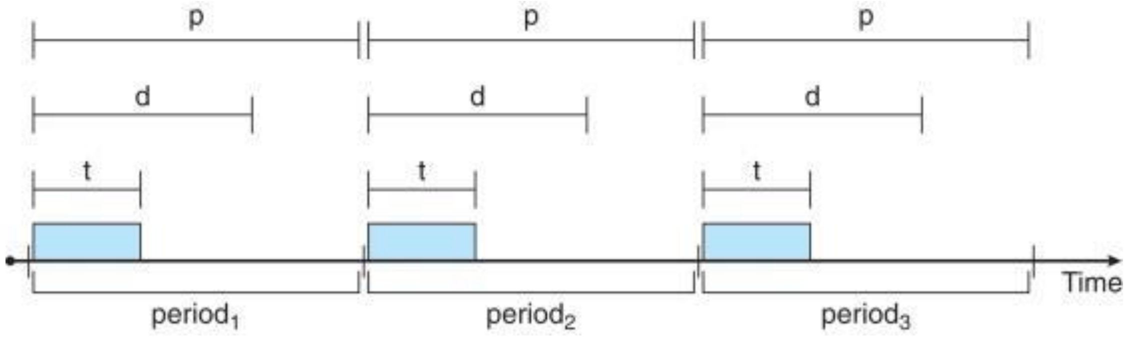
Dispatch Latency

- Conflict phase of dispatch latency:
  - Preemption of any process running in kernel mode
  - Release by low-priority process of resources needed by high-priority processes



Priority-based Scheduling

- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
  - But only guarantees soft real-time
- For hard real-time must also provide ability to meet deadlines
- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - Has processing time *t*, deadline *d*, period *p*
  - $0 \leq t \leq d \leq p$
  - **Rate** of periodic task is $1/p$

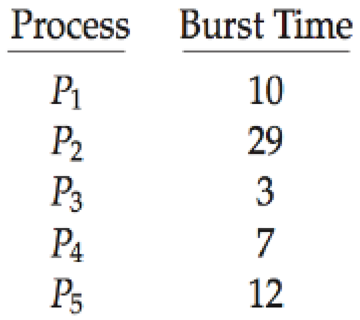


Operating System Examples

- Linux scheduling
- Windows scheduling
- Solaris scheduling

Algorithm Evaluation

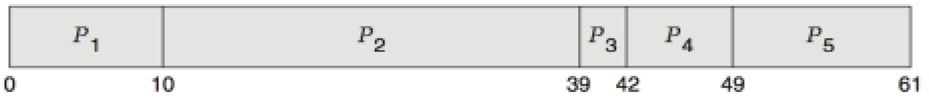
- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- **Deterministic modeling**
  - Type of **analytic evaluation**
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- **Consider 5 processes arriving at time 0**:



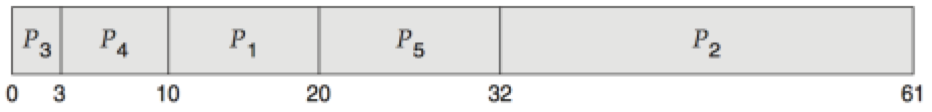
Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

Deterministic Evaluation

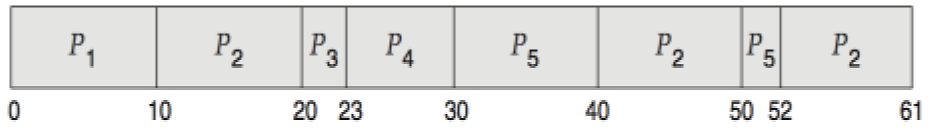
- For **each algorithm, calculate minimum average waiting time**
- Simple and fast, but requires exact numbers for input, applies only to those inputs
  - FCFS is 28ms:



  - Non-preemptive SFJ is 13ms:



  - RR is 23ms:



Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

Queueing Models

- Describes the arrival of processes, and CPU and I/O bursts probabilistically
  - Commonly exponential, and described by mean
  - 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
  - Computes utilization, average queue length, average wait time, etc

Little's Formula

- n = average queue length
- W = average waiting time in queue
- λ = average arrival rate into queue
- Little's law – in steady state, processes leaving queue must equal processes arriving, thus:

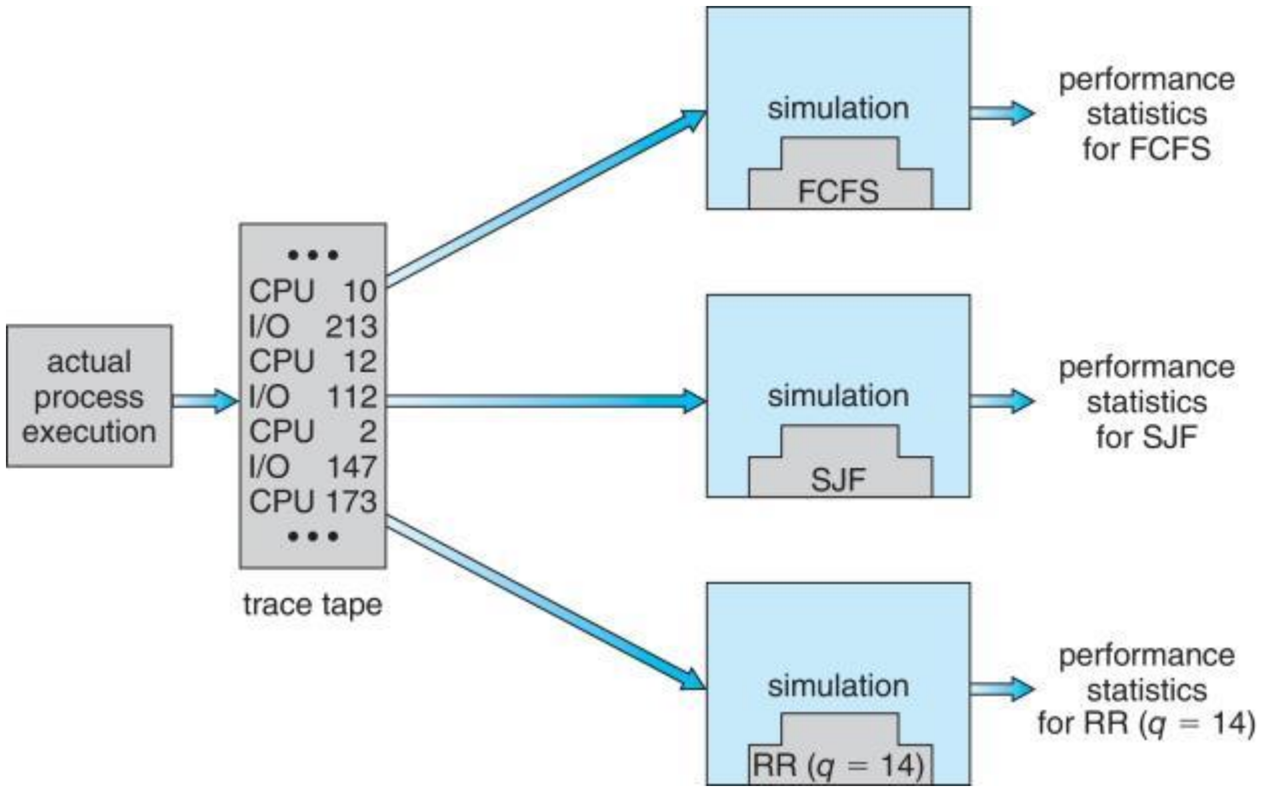
  $$n = \lambda \times W$$

  - Valid for any scheduling algorithm and arrival distribution
- For example, if on average 7 processes arrive per second, and normally 14 processes in queue, then average wait time per process = 2 seconds

Simulations

- Queuing models have limited accuracy
- **Simulations** more accurate
  - Programmed model of computer system
  - Clock is a variable
  - Gather statistics indicating algorithm performance
  - 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

Evaluation of CPU Schedulers by Simulation



Implementation

- **Even simulations have limited accuracy**
- Just implement new scheduler and test in real systems
  - High cost, high risk
  - Environments vary
- Most flexible schedulers can be modified per-site or per-system
- Or APIs to modify priorities
- But again environments vary

Books

- Operating Systems Concept
  - Written by Galvin and Silberschatz
  - Edition: 9th

References

- Operating Systems Concept
  - Written by Galvin and Silberschatz
  - Edition: 9th