Operating Systems

Dr. J.M. Khalifeh

Unit-1

Multilevel Queue Scheduling

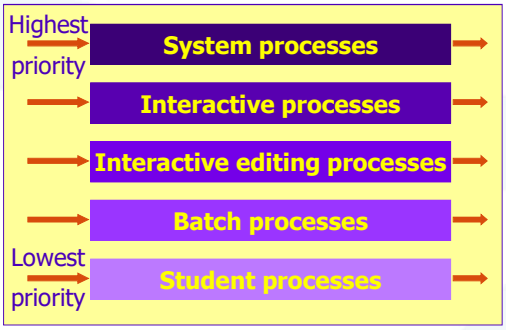
Multi-Processor Scheduling

Operating Systems

Dr. J.M. Khalifeh

Multilevel Queue Scheduling

Multilevel Queue Scheduling



- **Ready queue is partitioned into separate queues:**
  - foreground (interactive)
  - background (batch)
- **Each queue has its own scheduling algorithm**
  - foreground – RR
  - background – FCFS

- **Scheduling must be done between the queues.**
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS
- **Processes are permanently assigned to a queue on entry to the system. Processes do not move between queues**

Operating Systems Dr. J.M. Khalifeh

Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way.
- Multilevel-feedback-queue 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

Operating Systems Dr. J.M. Khalifeh

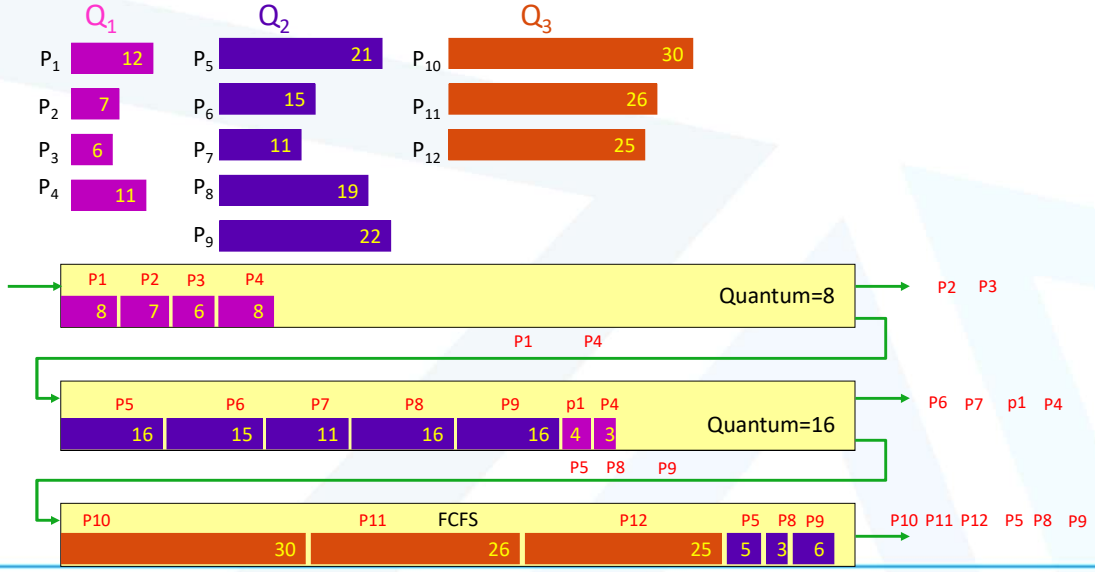
Operating Systems — Dr. J.M. Khalifeh

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – time quantum 8 milliseconds
  - Q_1 – 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.

Operating Systems — Dr. J.M. Khalifeh

Example of Multilevel Feedback Queue



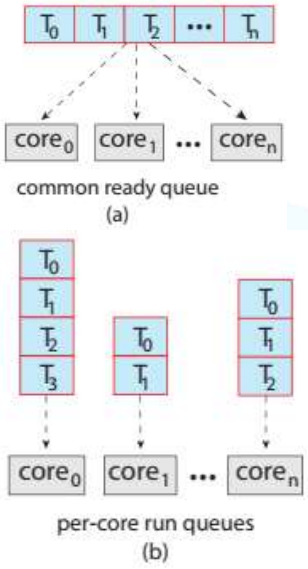
Operating Systems
Dr. J.M. Khalifeh

- CPU scheduling more complex when multiple CPUs are available
- Multiprocess may be any one of the following architectures:
  - Multicore CPUs
  - Multithreaded cores
  - (non-uniform memory access)NUMA systems
  - Heterogeneous multiprocessing

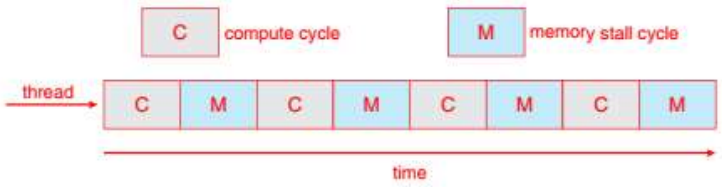
Operating Systems
Dr. J.M. Khalifeh

Approaches to 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

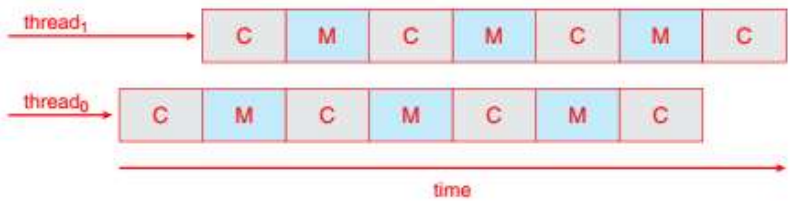


- Memory stall can occur because of a cache miss (accessing data that are not in cache memory. In this scenario, the processor can spend up to 50 percent of its time waiting for data to become available from memory
- Multiple threads per core also growing
  - Takes advantage of memory stall to make progress 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!



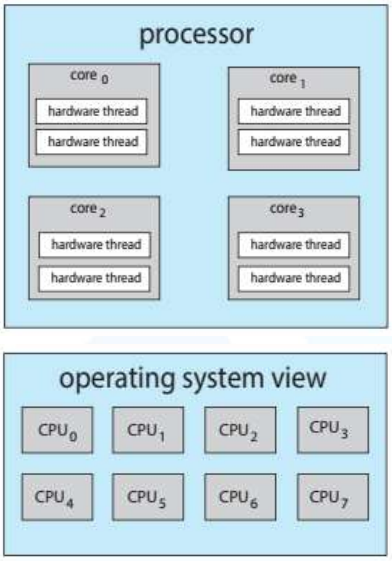
Multithreaded Multicore System

Operating Systems

Dr. J.M. Khalifeh

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

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



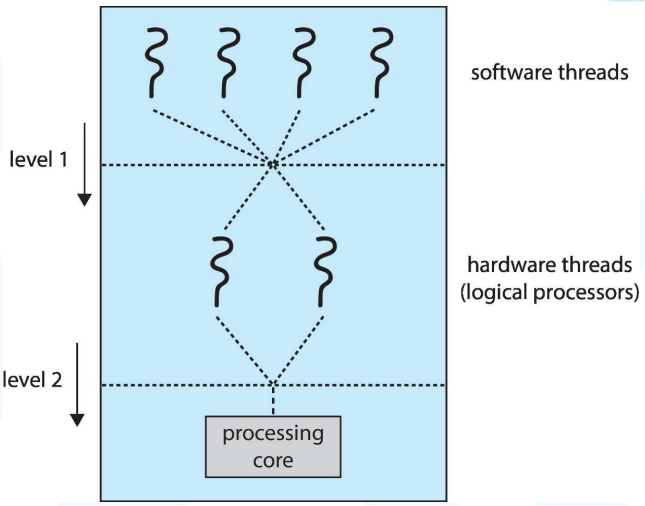
Multithreaded Multicore System

Operating Systems

Dr. J.M. Khalifeh

❖Two levels of scheduling:

1. The operating system deciding which software thread to run on a logical CPU
2. How each core decides which hardware thread to run on the physical core.



Multiple-Processor Scheduling – Load Balancing

Operating Systems
Dr. J.M. Khalifeh

- 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

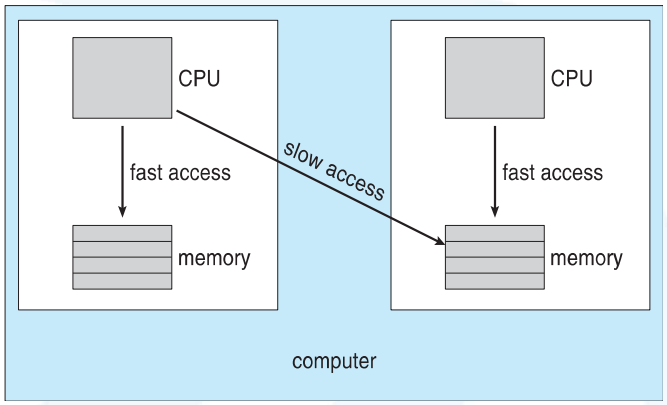
Operating Systems
Dr. J.M. Khalifeh

- 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 of.
- **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

Operating Systems
Dr. J.M. Khalifeh

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



Real-Time CPU Scheduling

Operating Systems
Dr. J.M. Khalifeh

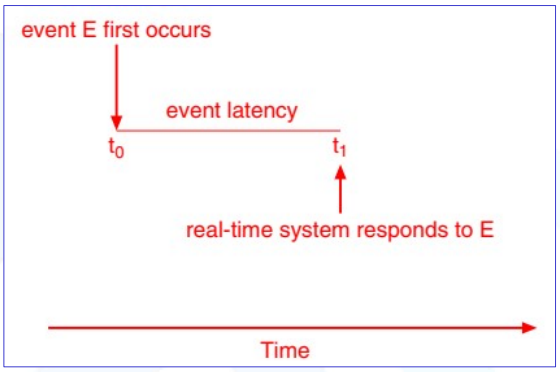
- The most important feature of a real-time operating system is to respond immediately to a real-time process as soon as that process requires the CPU.
- In general, we can distinguish between
  - **Soft real-time systems** – Critical real-time tasks have the highest priority, but no guarantee as to when tasks will be scheduled
  - **Hard real-time systems** – task must be serviced by its deadline
- Usually, different events have different latency requirements. Two types of latencies affect performance
  1. **Interrupt latency**
  2. **Dispatch latency**

Operating Systems

Dr. J.M. Khalifeh

Real-Time CPU Scheduling - Minimizing Latency

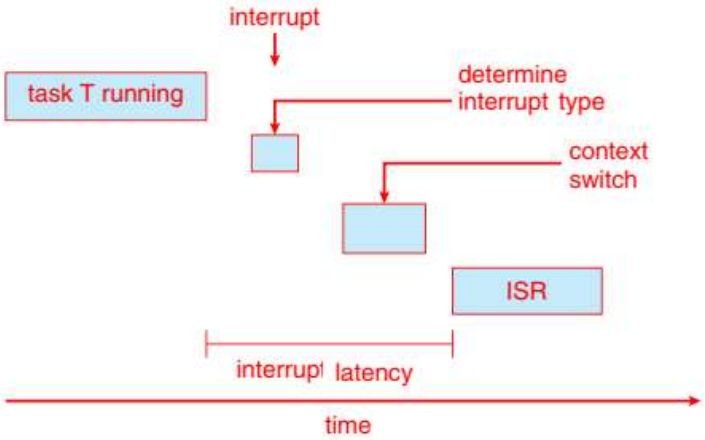
- The system is typically waiting for an event in real time to occur.
- Events may arise either in software or in hardware.
- When an event occurs, the system must respond to and service it as quickly as possible.
- **Event latency:** the amount of time that elapses from when an event occurs to when it is serviced.



Operating Systems

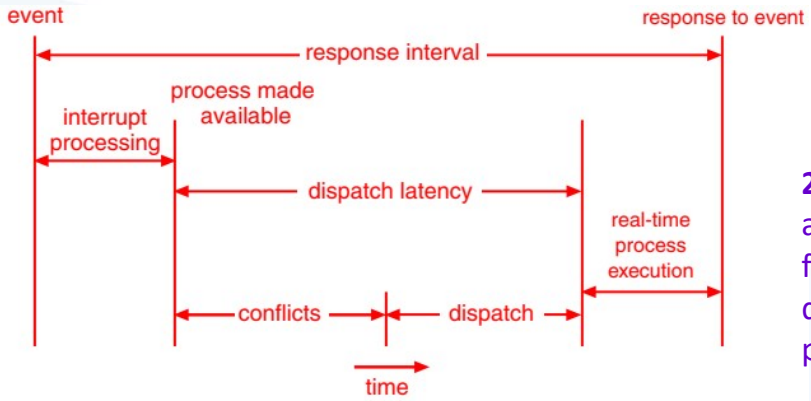
Dr. J.M. Khalifeh

Interrupt Latency



1.Interrupt latency – time from arrival of interrupt to start of routine that services interrupt

Dispatch Latency



2- Dispatch latency: The amount of time required for the scheduling dispatcher to stop one process and start another.

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

Operating Systems
Dr. J.M. Khalifeh

Priority-based Scheduling

- The scheduler for a real-time operating system must support a priority based algorithm with preemption.
- Priority-based scheduling algorithms assign each process a priority based on its importance; more important tasks are assigned higher priorities than those deemed less important.
- If the scheduler also supports preemption, a process currently running on the CPU will be preempted if a higher-priority process becomes available to run.

Operating Systems
Dr. J.M. Khalifeh

Priority-based Scheduling- Notes

Operating Systems

Dr. J.M. Khalifeh

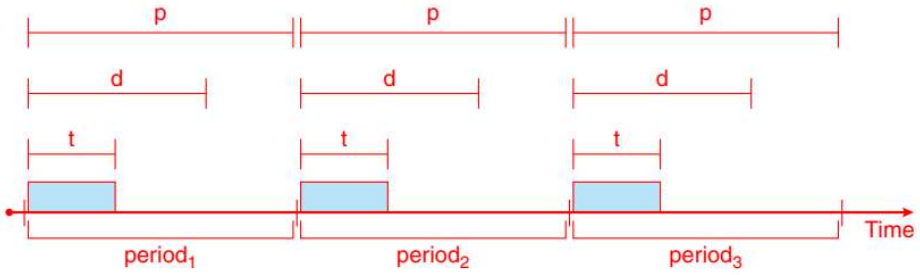
- Note that providing a preemptive, priority-based scheduler only guarantees soft real-time functionality.
- Hard real-time systems must further guarantee that real-time tasks will be serviced in accord with their deadline requirements, and making such guarantees requires additional scheduling features.

Priority-based Scheduling = define certain characteristics of the processes

Operating Systems

Dr. J.M. Khalifeh

- **Periodic process:**
  - ones require CPU at constant intervals
  - Has processing time *t*, deadline *d*, period *p*
  - $0 \le t \le d \le p$
  - **Rate** of periodic task is $1/p$



Priority-based Scheduling = define certain characteristics of the processes

- Schedulers can assign priorities according to a process's deadline or rate requirements.
- A process may have to announce its deadline requirements to the scheduler. Then, using a technique known as an **admission-control** algorithm.
- The scheduler does one of two things:
  - It either admits the process, guaranteeing that the process will complete on time,
  - or rejects the request as impossible if it cannot guarantee that the task will be serviced by its deadline.

Rate Montonic Scheduling

- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- P_1 is assigned a higher priority than P_2.

Operating Systems

Dr. J.M. Khalifeh

Rate Montonic Scheduling

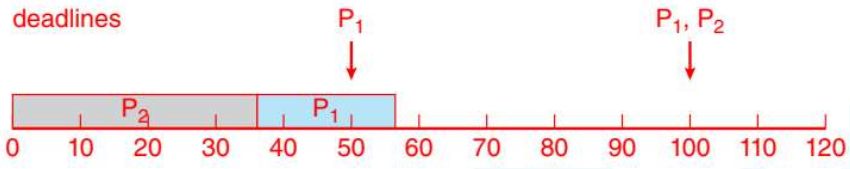
- p1 = 50 and p2 = 100.
- *t*1 = 20 for *P*1 and *t*2 = 35 for *P*2
- If we measure the CPU utilization of a process **Pi** as the ratio of its burst to its period—**ti/pi** —the CPU utilization of **P**1 is 20/50 = 0.40 and that of **P**2 is 35/100 = 0.35, for a total CPU utilization of 75 percent.
- Therefore, it seems we can schedule these tasks in such a way that both meet their deadlines and still leave the CPU with available cycles.

Operating Systems

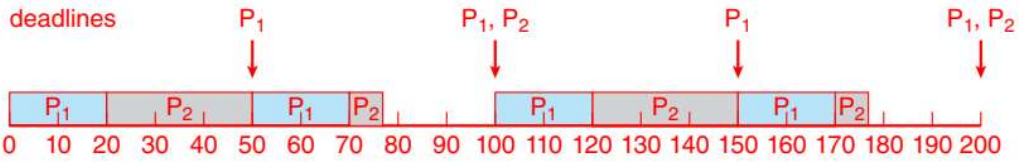
Dr. J.M. Khalifeh

Rate Montonic Scheduling

- When we assign **P**2 a higher priority than **P**1.



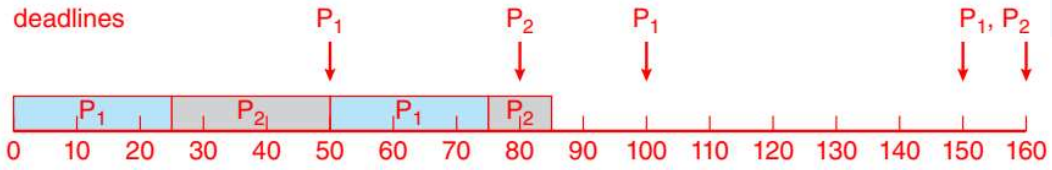
- When we assign **P**1 a higher priority than **P**2.



- if a set of processes cannot be scheduled by this algorithm, it cannot be scheduled by any other algorithm that assigns static priorities.

Missed Deadlines with Rate Monotonic Scheduling

- p1 = 50 and p2 = 80.
- *t*1 = 25 for *P*1 and *t*2 = 35 for *P*2
- The total CPU utilization of the two processes is (25⁄50) + (35⁄80) =0.94, and it therefore seems logical that the two processes could be scheduled and still leave the CPU with 6 percent available time.

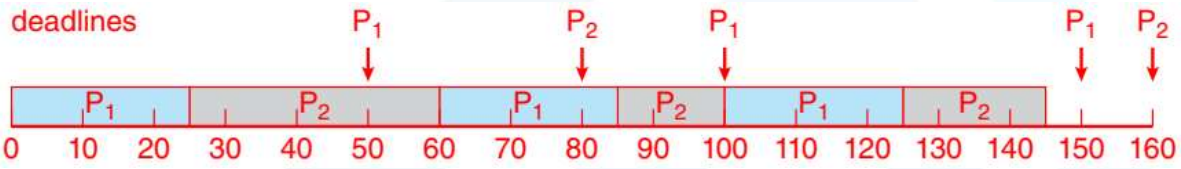


Process P2 misses finishing its deadline at time 80

Operating Systems — Dr. J.M. Khalifeh

Earliest Deadline First Scheduling (EDF)

- Priorities are assigned dynamically according to deadlines:
  - the earlier the deadline, the higher the priority;
  - the later the deadline, the lower the priority
- p1 = 50 and p2 = 80.
- *t*1 = 25 for *P*1 and *t*2 = 35 for *P*2
- Rrate-monotonic scheduling allows P1 to preempt P2 at the beginning of its next period at time 50, EDF scheduling allows process P2 to continue running. P2 now has a higher priority than P1 because its next deadline (at time 80) is earlier than that of P1 (at time 100).



Operating Systems — Dr. J.M. Khalifeh

Operating Systems

Dr. J.M. Khalifeh

Proportional Share Scheduling

- *T* shares are allocated among all processes in the system
- An application receives *N* shares where *N* < *T*
- This ensures each application will receive ***N*** / *T* of the total processor time