CSCI 4730
Special Class on Real-Time Systems

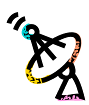
October 6, 2009

Real-time vs. general operating system

- In addition to requiring logical correctness, real-time systems require temporal correctness
  - Logical correctness: Given an input, the system must create the correct output
  - Temporal correctness: The correct output must be created at the correct time

Real-time applications

- Real-time systems are used in a variety of applications
  - Safety critical systems
    - Airplane autopilot, power plant controllers
  - Expensive systems
    - Satellite controllers, Mars rovers
  - Other time critical applications
    - Radar system, Sensor networks
  - Consumer and embedded devices
    - Cell phones



Types of Real-Time Systems

- Hard real-time systems
  - Deadlines must be met
  - Missed deadline = system failure
- Soft real-time systems
  - Some deadline misses OK
  - Many missed deadlines = lower quality of service
- Mixed systems
  - Real-time and non-real-time jobs execute together
  - Scheduling must ensure all real-time jobs meet their deadlines

Aspects of real-time research

- There are many active areas of research real-time systems
  - Scheduling algorithms
  - Schedulability tests
  - Strategies for reducing power consumption
  - Real-time operating systems
  - Real-time programming languages
  - Specific real-time applications
  - Hardware for real-time systems

Properties of Real-Time Schedulers

- Priority based
  - Jobs are assigned priorities
  - Scheduler always executes jobs with the highest priority
- Preemptive
  - When a higher priority job arrives, it interrupts currently executing job
    - Preemption is often allowed, but not always
    - Sometimes preemption may be allowed only at certain points within a job

More properties

- Event driven
  - Some external events can change the system configuration
    - Add jobs
    - Remove jobs
    - Change job priorities
  - Example: Power plant temperature exceeds certain safety threshold
- Low event latency
  - When such an event occurs, the system must respond in a timely manner
    - Latency = system_response_time – event_time

Standard real-time system model

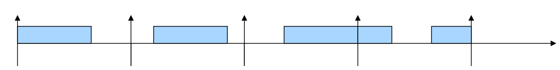
- Periodic and sporadic tasks: A mechanism for executing a job repeatedly at regular time intervals
- Simplified model T = (p,e)
- Periodic tasks invoke a new job every p time units
- Sporadic tasks invoke jobs at least p time units apart

Task notation

- T = (φ,p,e,D)
  - φ = phase
    - Periodic: start time of first job
    - Sporadic: first jobs starts no earlier than φ
  - e = execution requirement
  - p = period
    - Periodic: exact time between job releases
    - Sporadic: minimum time between job releases
  - D = relative deadline
    - Amount of time job has to execute
- Simplified model T = (p,e)
  - φ = 0
  - D = p

Example

- T = (5,3)
  - This task generates a new job every 5 time units
  - Each job will require at most 3 time units to execute
  - The deadline of each job is equal to the arrival time of the next job



Task utilization

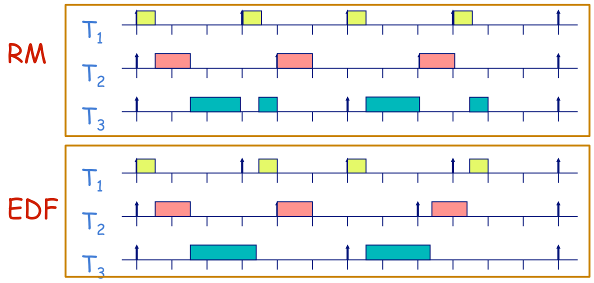
- Given a periodic task $T_i = (p_i, e_i)$, the utilization of T_i is $u_i = e_i/p_i$
  - Proportion of processing time this task will require on average
- Given a set of n periodic or sporadic tasks $\tau = T_1, T_2, \ldots, T_n$, $U(\tau)$ is the total utilization of all tasks
  - $U(\tau) = \Sigma_{1 \le i \le n} u_i$
- Many schedulability tests are based on task utilization

Two common scheduling algorithms

- Earliest Deadline First (EDF)
  - Jobs with earlier deadlines are given higher priority
- Rate Monotonic (RM)
  - Jobs generated by tasks with shorter periods are given higher priority
- Both algorithms have preemptive and non-preemptive versions

Example Preemptive RM and EDF schedules

Three tasks $T_1 = (3, 0.5)$, $T_2 = (4, 1)$, $T_3 = (6, 2)$



EDF will meet all deadlines if it is possible to do so

We say EDF is optimal on uniprocessors

Utilization-based EDF test

- Given a set of periodic tasks

  $\tau = \{T_1 = (e_1, p_1), T_2 = (e_2, p_2), \ldots, T_n = (e_n, p_n)\}$
- If $U(\tau) \leq 1$, then τ can be successfully scheduled using preemptive EDF
  - No jobs will miss their deadlines

Utilization-based RM test

- Given a set of periodic tasks

  $\tau = \{T_1 = (e_1, p_1), T_2 = (e_2, p_2), \ldots, T_n = (e_n, p_n)\}$
- If $U(\tau) \leq n(2^{1/n} - 1)$, then τ can be successfully scheduled using preemptive RM
  - Note: $n(2^{1/n} - 1)$ is a decreasing function that approaches ln 2 (approx. 69%) as n increases
- Why use RM?
  - Many real-time operating systems can only schedule tasks with fixed priority
    - All jobs generated by the same task must have the same priority

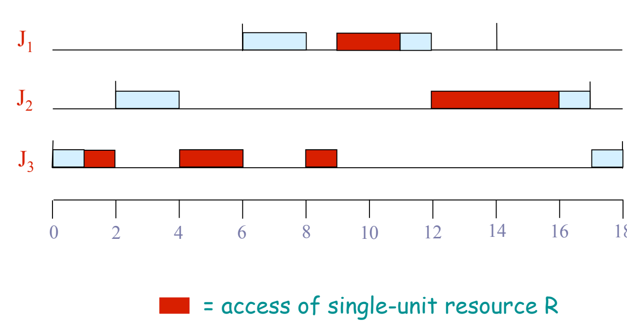
Shortcomings

- The model provided assumes all tasks are independent
  - Jobs may share resources
    - In this case, one job may block another job
  - One job may generate data that will be used by another job
    - In this case, we would want to impose a precedence constraint on these jobs

Scheduling jobs with dependiencies

- Both blocking and precedence constraints can cause priority inversion and timing anomolies
  - Priority inversion: A higher priority job may be forced to wait while a lower priority job executes
  - Timing anomolies: Reducing the execution of one job may cause another job finish execution at a later time

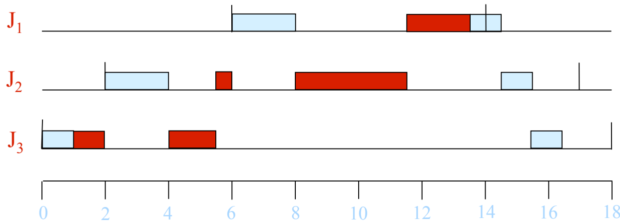
Priority inversion



Timing anomalies

When tasks share resources, there may be timing anomalies

Example: Reducing J_3's critical section from 4 time units to 2.5 causes J_1 to miss its deadline!



Multiprocessor scheduling

- Scheduling analysis is much more difficult on multiprocessors
- Many tests can only guarantee feasibility when the utilization is approximately m/2, where m is the number of processors
  - Things get even more complicated when there is resource sharing or precedence constraints

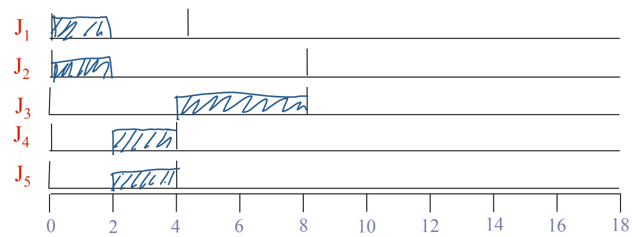
Optimal multiprocessor scheduling

- Hong and Leung used the following example to prove that no online scheduling algorithm can be optimal when deadlines are not all equal
  - $J_1 = J_2 = (0; 2; 4)$; $J_3 = (0; 4; 8)$
  - Later arrival times as follows
    - $J_4 = J_5 = (2; 2; 4)$, and
    - $J_4 = J_5 = (4; 4; 8)$.

Example Part 1

- $J_1 = J_2 = (0, 2, 4)$; $J_3 = (0, 4, 8)$
- Later arrival times as follows
  - $J_4 = J_5 = (2, 2, 4)$, and
  - $J_4' = J_5' = (4, 4, 8)$.

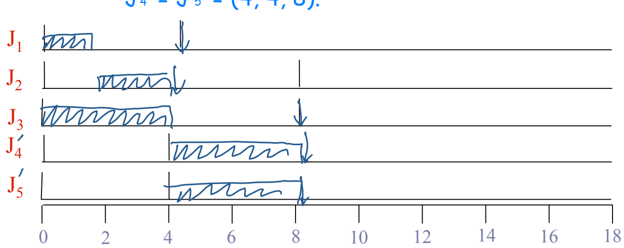
J_3 cannot execute in interval [0,2]



Example Part 2

- $J_1 = J_2 = (0, 2, 4)$; $J_3 = (0, 4, 8)$
- Later arrival times as follows
  - $J_4 = J_5 = (2, 2, 4)$, and
  - $J_4' = J_5' = (4, 4, 8)$.

J_3 must execute in interval [0,2]



Multiprocessor utilization test

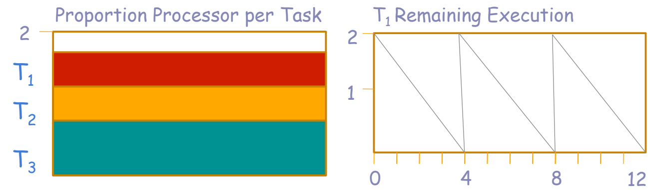
- Any task set τ is feasible on m processors provided
  - $\max\{u_i\} \leq 1$
  - $U(\tau) \leq m$
- Knowing some schedule exists is not the same as having a schedule that meets all deadlines!

Multiprocessor scheduling of PTs

- There are optimal online algorithms for scheduling periodic tasks on multiprocessors
  - Pfair, LLREF
- These tasks make decisions to emulate the ideal schedule

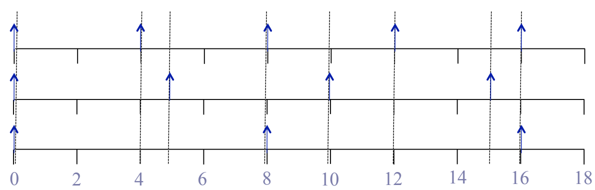
Ideal (but impractical) schedule

- Ideally, we would execute all tasks at a constant rate
  - Example T_1 = (4,2), T_2 = (6,3), and T_3 = (8,6)



New scheduling algorithm: NQ-Wrap

- The timeline is broken into time slices
  - Dividing points are determined by task deadlines
  - Scheduling within a TL plane $[t_{i-1}, t_i]$ ensures tasks have executed at their ideal amount at by time t_i



- Example T_1 = (4,2), T_2 = (5,3), and T_3 = (8,6)

Local execution and utilization

- Within each time slice $[t_{j-1}, t_j)$, each task is assigned a local workload and utilization
- $\ell_{i,t}$ = remaining work for T_i within time slice
- $r_{i,t}$ = local utilization within time slice
- $r_{i,t} = \ell_{i,t} / (t_j - t)$
- At start of each slice $r_i = u_i$
  - i.e., $\ell_{i,t(j-1)} = u_i \times (t_j - t_{j-1})$

Schedulers

- NQ-Wrap has two schedulers
  - Global scheduler makes decisions for all processors
  - Local scheduler schedules tasks on single processor
- In NQ-Wrap, the global scheduler executes at time slice boundaries only
  - Determines schedule for entire time slice and sends schedules to processors

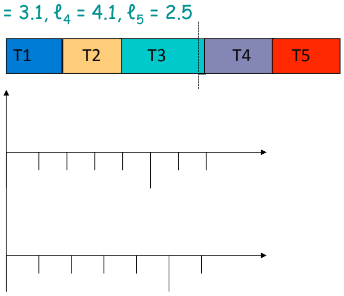
Global scheduler

- At the beginning of each time slice $[t_{j-1}, t_j)$, the global scheduler performs the following tasks
  - Determine ℓ_i for each task T_i
  - Considers these execution times in a long sequence
  - Cuts this sequence every $(t_j - t_{j-1})$ time units
  - Sends one sequence per processor until all sequences are assigned

Example

T_1=(7,2), T_2= (10,3), T_3= (9,4), T_4 = (12,7), T_5 = (14,5)

$\ell_1 = 2$, $\ell_2 = 2.1$, $\ell_3 = 3.1$, $\ell_4 = 4.1$, $\ell_5 = 2.5$



Why I like researching real-time systems

- Analyzing real-time systems is like solving puzzles
  - Analysis is visual
  - Small changes in assumptions can have large impact in analysis
- If this seemed interesting to you, please feel free to contact me regarding research projects or directed study!!!