Rate Monotonic Scheduler

- Classic hard real-time CPU scheduler
  - Preemption-based
  - Independent tasks
- Static scheduler: fixed priorities
  - No dynamic priority adjustments
  - Not a static schedule
RM Algorithm

- Main components
  - Periodic tasks, marked by start and end
  - Priority-based preemption
  - Shorter period tasks get higher priority

Crucial Zone Theorem

- Only need to check the first period
  - If tasks make their first deadline, then they can make the following deadlines
Discussion of RM

- Zhao’s evaluation criteria
  - CPU utilization: guaranteed theoretical WCAU limit of 0.693 for CPU
  - Robustness: OK if below the 0.693 limit
  - Timing fault: limited by preemption
  - Aperiodic jobs: high priority until the 0.693 WCAU limit
  - Run-time overhead: may be high

Another Discussion

- Sha’s list of advantages
  - Fixed priorities: easy management
  - Aperiodic tasks: sporadic server, etc
  - Synchronization: priority ceiling, etc
  - Imprecise computations
  - Ease of implementation
Transient Overload

- Variable task CPU consumption
  - Worst case may be too stringent
  - Guarantee a set of critical tasks for the worst case (WCAU)
  - Add other tasks at lower priorities
- During overload
  - Set of highest priority tasks run “normally” (continue to be guaranteed)

Period/Priority Mismatch

- Q: high priority task, but long period?
- A: period transformation (task slicing)
  - Divide the task into $k$ pieces
  - Run each piece in its own period, size $p/k$
  - Original task completed with the last piece
- Complications and issues
  - Context switch overhead ($k$ times)
  - Portability, synchronization
Aperiodic Tasks

- Two kinds of aperiodic tasks
  - High priority: emergency alerts
  - Low priority: background jobs
- Problem with background jobs
  - Neglected during transient overload
  - Long response time (example 3, p. 56)

High Priority Aperiodic

- Aperiodic server
  - Pretends to be a high priority periodic task
  - Sub-schedules its allotted time slices
  - Can preempt normal periodic tasks
- Constraints and issues
  - Pre-allocation counts towards WCAU limit
  - Independent replenishment and overload handling policies
Task Synchronization

- Priority inversion (example 4, p. 57)
  - Priorities combined with locking
  - T3 holding lock(A), is preempted by T1, which needs lock(A)
  - T1 waits for T3, and T3 waits for T2

- Examples of remedies
  - No preemption when holding lock(A) - also introduces priority inversion

Priority Ceiling Protocol

- Priority inheritance
  - If T1 waits for T3, then T3 gets T1 priority

- Dynamic rescheduling
  - Critical sections execute at highest priority
  - Waiting is outside critical sections
  - Highest priority waiting task runs next
  - Critical section becomes a subroutine of the highest priority waiting task
Advantages of PC

- Deadlock free (example 5, p. 57)
  - Critical section always runs
  - Waiting is outside critical sections
  - T2 prevents T1 from getting lock (fig. 2)
- Bounded priority inversion
  - High priority tasks “jump over” queues
- Good schedulability tests

Further Discussion of PC

- Priority inversion
  - Low priority tasks in critical sections become high priority
  - Tasks w/o critical sections may wait
- Execution overhead
  - Every critical section needs a priority queue
“Best Effort” Scheduling

- Minimum laxity first (MLF)
  - Laxity = time to latest feasible start time (when the job can still complete)
  - Run the job closest to failing first
  - Optimal in minimizing deadline failures
  - Earliest-Deadline First (EDF) also optimal
  - MLF = EDF when jobs have same length

Ada Case Study

- Designed for “safe” programming
  - Predictable program properties
  - Real-time a big application target (AF)
  - One way to do each task
- Some difficulties with real-time
  - Non-deterministic task scheduling
  -Prioritized tasks queued in FIFO order
  - No dynamic change of task priorities
“Fixing Ada for RT”

- Adopt priority ceiling protocol (p. 60)
  - Monitor task guards critical section (fig. 4)
  - Priority ceiling emulated by run-time
- Difficulties with Ada specification
  - Monitor task cannot suspend itself (no I/O)
  - Sporadic server for aperiodic tasks
  - Disallow Ada fixed priorities
  - Get around FIFO queues

RM: Average Case Analysis

- Stochastic characterization
  - Most likely (average) workload, described by cumulative distribution function (CDF)
  - Utilization determined by scaling workload
- Asymptotic approximation
  - Probabilistic density function of utilization calculated from CDF plus assumptions
  - U depends only on task period, not length