Last Time

- Real-time scheduling using cyclic executives

Today

- Real-time scheduling using priorities
  - How to assign priorities?
  - Will the assigned priorities work?
  - What can we say in general about the scheduling algorithms?

Real-Time Review 1

- Motivation
  - Your car’s engine control CPU overloads → BAD
  - Airplane doesn’t update flaps on time → BAD
- System contains n periodic tasks T₁, …, Tₙ
- Tᵢ is specified by (Pᵢ, Cᵢ, Dᵢ)
  - P is period
  - C is execution cost (also called E)
  - D is relative deadline
- Task Tᵢ is “released” at start of period, executes for Cᵢ time units, must finish before Dᵢ time units have passed
  - Often Pᵢ=Dᵢ, and in this case we omit Dᵢ

Real-Time Review 2

- Given:
  - A set of real-time tasks
  - A scheduling algorithm
- Is the task set schedulable?
  - Yes → all deadlines met, forever
  - No → at some point a deadline might be missed
- Ways to schedule
  - Cyclic executive
  - Static priorities
  - Dynamic priorities
  - …

Cyclic Exec. Vs. Priorities

- Priorities are more flexible but less predictable
- Priorities may be fixed at design time or computed at runtime

Today’s Assumptions

- Tasks are running on an RTOS
  - Each task runs in its own preemptive thread
  - Scheduled using priorities
- Uniprocessor embedded system
  - If system has multiple processors we analyze them separately
  - This works unless we want tasks to migrate between processors
- Tasks don’t synchronize using locks
  - Later we’ll see how to avoid this assumption
- No OS overhead
  - Later we’ll see how to avoid this assumption
How to assign priorities?
- Rate monotonic (RM)
  - Shorter period tasks get higher priority
- Deadline monotonic (DM)
  - Tasks with shorter relative deadlines get higher priority
- Both RM and DM...
  - Have good theoretical properties
  - Work well in practice
- Other considerations
  - Criticality
  - Output jitter requirement

Example
- System with 4 tasks:
  - $T_1 = (4, 1)$, $T_2 = (5, 1.8)$, $T_3 = (20, 1)$, $T_4 = (20, 2)$
- What is the RM priority assignment?
- What is the DM priority assignment?
- Will these priority assignments work?
  - Remember: “work” means no deadlines missed, ever

Utilization
- Utilization of a task: $C / P$
- Utilization of a task set: Sum of task utilizations
- Schedulable utilization of a scheduling algorithm:
  - Every set of periodic tasks with utilization less or equal than the schedulable utilization of an algorithm can be feasibly scheduled by that algorithm
- Higher schedulable utilization is better
- Schedulable utilization is always $\geq 0.0$ and $\leq 1.0$
- Question: What is the schedulable utilization of...
  - FIFO scheduling?
  - EDF scheduling?
  - Generic fixed priority scheduling?
  - RM scheduling?

How about dynamic priorities?
- Dynamic priority means that priorities are not fixed at design time – the system can keep changing them as it runs
- Example algorithms
  - Earliest deadline first (EDF)
  - Least slack time first (LST)
  - First-in first-out (FIFO)
  - Last-in first-out (LIFO)
- Which of these work, for the example from the previous slide?

FIFO Schedulable Utilization
- $U_{\text{FIFO}} = 0.0$
  - Oops!
- Proof
  - Pick a utilization $u$
  - Pick an arbitrary period $p$
  - Create a task set with two tasks
    - Task 1 has $C = p \times u/2$, $P = p$ (utilization = $u/2$)
    - Task 2 has $C = p$, $P = p \times 2/u$ (utilization = $u/2$)
  - This task set has utilization $u$ and is not schedulable

EDF Schedulable Utilization
- $U_{\text{EDF}} = 1.0$
  - As long as we ignore synchronization between tasks
  - We’ll return to this result later
**Fixed Priority Schedulable Utilization**

- $U_{FP} = 0$
- $U_{RM} = ?$
- $U_{RM} \neq 1$

**Simply Periodic Case**

- A set of tasks is simply periodic if, for every pair of tasks, one period is multiple of other period.
- Result: A system of simply periodic, independent, preemptible tasks whose relative deadlines are equal to their periods is schedulable according to RM iff their total utilization does not exceed 1.0.
- Proof:
  - Assume $T_i$ misses deadline at time $t$
  - $t$ is integer multiple of $P_i$ and $P_{ik} < P_i$
  - Then, total time to complete jobs with deadline $t$ is:
    \[
    \sum_{i} U_i \leq 1
    \]
  - $T_i$ can only miss deadline if $U > 1.0$

**General RM Case**

- Theorem
  - $n$ independent, preemptible, periodic tasks with $D_i P_i$ can be feasibly scheduled by RM if its total utilization $U$ is less or equal to $m(2^n - 1)$.
  - For $n=1$, $U = 1.0$
  - For $n=2$, $U = 0.83$
  - For $n=\infty$, $U = 0.69$

**RM Proof Sketch**

- General idea
  - Find the most-difficult-to-schedule system of $n$ tasks among all difficult-to-schedule systems of $n$ tasks.
- Difficult-to-schedule
  - Fully utilizes processor for some time interval
  - Any increase in execution time would make system unschedulable
- Most-difficult-to-schedule
  - System with lowest utilization among difficult-to-schedule systems.
  - Difficult-to-schedule situations happen when all tasks are released at once.
    - First prove that this is the most difficult case.
    - Then prove that in this case, the system is schedulable.

**Summary**

- Fixed priority scheduling
- Not optimal – So why do we care?
  - Simple
  - Efficient
  - Easy to implement on standard RTOSs
  - Predictable – During overload low-priority jobs lose
- Fixed priority scheduling is heavily used in real embedded systems.