CSCI1600: Embedded and Real Time Software
Lecture 27: Real Time Scheduling III
Steven Reiss, Fall 2017
Relaxing Constraints

- Non-preemptability
- Resource Contention
- Self-suspension
- Context Switching time
- Sporadic and aperiodic jobs
- Job dependencies
- Multiple cores/threads/processors
Aperiodic Job Scheduling

- **Aperiodic Tasks**
  - Arise occasionally
  - Soft deadlines
  - Might be prioritized

- **Goal:** minimize worst/average case performance
Simple Aperiodic Scheduling

- Run aperiodic tasks in open slots
- How to schedule them
  - Round robin
  - Prioritized round robin
    - Run tasks with higher priorities first
    - Allocate CPU time according to priority
- This gives no guarantees of performance
Total Bandwidth Server

- Make an aperiodic job look like a sporadic one
  - Add to schedule using standard algorithm (e.g. EDF)
  - Can reject the sporadic job if necessary

- Key: set the parameters for the job so that
  - The result is schedulable (total $U \leq 1$ or $U \leq \ln(2)$)

- Recall $U = \frac{e}{D}$
  - $e$ is known and fixed for the job
  - This lets us vary $D$ to accommodate the jobs
Total Bandwidth Server

- Determine desired aperiodic load $U_s$
  - To still allow sporadic jobs and periodic jobs
  - Total $U \leq 1$
- TBS is configured with **budget** $Q_s$ and **period** $T_s$
  - $Q_s/T_s = U_s$
- Terminology
  - $e_s$ : budget for the server (its execution time)
  - $e_q$ : execution time of job at the head of the queue
  - $u_s$ : size of the server (max utilization)
Total Bandwidth Server

- Initially $e_s = 0$ and $d = 0$
- When a new job with execution time $e$ comes into an empty queue
  - Set $d = \max(d, t) + \frac{e}{u_s}$
  - Set $e_s = e$
- When the server completes the current job, remove the job from the queue
  - If there are more jobs,
    - Set $d = d + \frac{e_q}{u_s}$
    - Set $e_s = e_q$
Example

- Periodic: P1(4,1), P2(6,3)
- Aperiodic: A3: e=2 at time 1, A4: e= 1 at time 2
- First determine $U_s$
  - $U_p = \frac{1}{4} + \frac{3}{6} = 0.75$
  - $U_s = 0.25$ (could be less)
Example

At Start $P_1(4,1), P_2(6,3), e_s = 0, d = 0$

A3: $e=2$ at time 1

At time 1: $d = \max(0,1) + \frac{2}{0.25} = 9, e_s = 2$
Example

- $P_1(4,1)$, $P_2(6,3)$, $P_s(9,2)$
- At time 3: new aperiodic job $P_t$ ($A4: e=1$ at time 2), but $P_s$ not done
- At time 7: $P_s$ finishes, $P_2$ has deadline 12
At time 7: \( d = d + \frac{e_q}{U_s} = 9 + \frac{1}{0.25} = 13 \), \( e_s = 1 \)
- \( P_s(13,1) \)
Example

At time 7: \( d = d + e_q/U_s = 9 + 1/0.25 = 13, \) \( e_s = 1 \)

- \( P_s(13,1) \)
- P1 and P2 have deadlines of 12
Handling Task Dependencies

- Suppose tasks depend on one another
  - Represent this as a precedence relation
  - $T_i \rightarrow T_j$ if $T_i$ must precede $T_j$
- Why would this arise?
- How can we schedule the result
  - EDF doesn’t work directly
EDF* Scheduling

- Let \( D_i \) be the deadline for task \( i \)
- Let \( e_i \) be the execution time for task \( i \)
- Define \( D'_i \) as the modified deadline for task \( i \)
  - \( D'_i = \min(D_i, \text{MIN}(T_j \text{ dependent on } T_i)[D'_j - e_j]) \)
- Then use EDF on the modified deadlines
Jitter

- Periodic jobs are not perfectly periodic
- Compute maximum jitter and add to execution/deadline
Thread Scheduling

- All our schedules assume single core, single thread
  - This might not be realistic today
- Can we just use these techniques for multithreaded scheduling?
  - Using any available processor
  - What can go wrong?
Thread Scheduling

- Assume m processors, m+1 tasks
  - Tasks 1..m are (1,2e) for small e (2me < 1)
  - Task m+1 is (1+e,1)
  - Try: m = 4, 1-4 (10,2); 5: (11,10)
- What happens?
  - Using EDF/RM/DM
    - Tasks 1..m are all scheduled
    - But then m+1 can’t run
    - Even though there is lots of idle time
  - Other anomalies are possible (esp. w/ synchronization)
Thread Scheduling

- Can still use the basic algorithms
  - Assign each task to a particular thread
  - Ensure the utilization of that thread is $\leq 1$
- Then you can use EDF (RM/DM) to schedule the thread
- Optimal assignment is NP-hard
  - But approximations are pretty good
Practical Scheduling

- What should be done in practice?
- What should a general purpose OS do?
- Suppose we just assign priorities to jobs
  - Run the job with the highest priority at any time
  - What is the result?
Scheduling Exercise

- P1(8,4), P2(12,4), P3(20,4)
Adding Sound Samples

- Assume you have 2 sounds
  - You want to play them simultaneously
  - Sounds are 8 bit 8K samples
- In theory, you just add the waves
  - Result should be a new wave
Problems with Adding

- What are the 8 bit values (what is 0?)
  - 128 == 0, smaller values negative, larger positive
  - Thus sample = x+128
  - (x+128) + (y+128) = (x+y)+256 [wrong]
  - (x+128-128) + (y+128-128) + 128 = (x+y)+128

- What happens if sum >= 128 or < -128
  - Need to set result to 255 or 0
  - Clipping occurs (doesn’t sound that good)
  - Can scale the two sounds if needed
  - Original sounds shouldn’t use full range
Next Time

- Real Time Operating Systems