Embedded Systems Summary

Nicolas Trüssel

July 31, 2016

1 Miscellaneous

Dependable means

- Reliable (continue working correctly when it worked correctly before)
- Maintainable (recover from errors)
- Available (probability that it's working)
- Safety
- Security

Efficient in terms of:

- Energy
- Code size
- Memory consumption
- Run time
- Weight
- Cost

2 Generic Time Triggered Cyclic Executive Scheduler

Let f denote the frame length, P the full period, $D(k)$ the relative deadline of task k, $C(k)$ its execution time, and $p(k)$ the period of task k (how often it occurs). Then the following conditions have to be satisfied:

- $\forall k. f \leq p(k)$ (at most one execution within a frame)
- $P = \operatorname{lcm}_k (p(k))$
- $\forall k. f \geq C(k)$ (processes start and complete within single frame)
- $\forall k. 2f - \gcd(p(k), f) \leq D(k)$ (between release time and deadline of every task there is at least one frame boundary)

3 Aperiodic Scheduling

	Equal arrival, non-preemptive	Arbitrary arrival, preemptive
Independent Tasks	EDD	EDF
Dependent Tasks	LDF	EDF*

3.1 Definitions

- $J = \{j_1, j_2, \ldots, j_n\}$ is a set of tasks.
- a_i or r_i is the arrival / release time of task i.
- d_i is the deadline of task i.
- C_i is the total computation time of task i.
- $c_i(t)$ is the the remaining execution time of task i at time t.
- s_i is the start time of task i.
- f_i is the finish time of task i.
- $L_i = f_i - d_i$ is the lateness of task i.
- $E_i = \max(0, L_i)$ is the exceeding time or tardyness of task i.
- $X_i = d_i - a_i - C_i$ is the laxity or slack of task i.

3.2 EDD

Schedule the tasks in order of non-decreasing deadlines. This minimizes the maximal lateness.

3.3 EDF

Always execute the task with the earliest absolute deadline. Schedulability test:

$$\forall i \in [n].t + \sum_{k=1}^{i} c_k(t) \leq d_i$$

3.4 LDF

Among all tasks without successors select the task with the latest deadline. Put it in a stack. Repeat until no more tasks. Now execute tasks as they are on the stack.

3.5 EDF*

Modify arrival and deadline of each task and use EDF on modified tasks.

$$r_j^* = \max_j \left(r_j, \max_i \left(r_i^* + C_i | J_i \to J_j \right) \right)$$

$$d_i^* = \min_i \left(d_i, \min_j \left(d_j^* - C_j | J_i \to J_j \right) \right)$$

4 Periodic Scheduling

Periodic scheduling is always preemptive.

	Deadline = Period	Deadline ≤ Period
Static Priority	Rate Monotonic	Deadline Monotonic
Dynamic Priority	EDF	EDF*

4.1 Definitions

- $\Gamma = \{\tau_1, \tau_2, \dots \tau_n\}$ is set of periodic tasks.
- $\tau_{i,j}$ is the j-th instance of task τ_i.
- $a_{i,j}, r_{i,j}, d_{i,j}, s_{i,j}, f_{i,j}$ are the same as for aperiodic tasks.
- Φ_i is the phase of task i (release time of the first instance).
- D_i is the relative deadline of task i.
- T_i is the period of the task (time between 2 releases).

The following hypotheses are assumed

- $r_{i,j} = \Phi_i + (j-1)T_i$
- C_i is constant.
- $d_{i,j} = \Phi_i + (j-1)T_i + D_i$
- The tasks are independent

4.2 Rate Monotonic

Always schedule the task that has the shortest period. Sufficient (but not necessary) schedulability test:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \left(2^{\frac{1}{n}} - 1\right)$$

4.3 Deadline Monotonic

Always schedule the task that has the shortest relative deadline. Sufficient (but not necessary) schedulability test:

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n \left(2^{\frac{1}{n}} - 1\right)$$

4.4 Necessary and sufficient schedulability test

We define the interference I_i for task i as

$$I_i(t) = \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

where the tasks are ordered such that $m < n \iff D_m < D_n$

4.4.1 Algorithm

```
foreach (τ_i ∈ Γ) {
        I = 0;
        do {
                R = I + C_i;
                if (R > D_i) return false; // unschedulable
                I = Σ_{j=1}^{i-1} ⌈R / T_j⌉ C_j;
        } while (I + C_i > R);
}
return true;
```

4.5 EDF

Always schedule the task with the earliest deadline. Schedulable with EDF iff $\sum_{i=1}^{n} \frac{C_i}{T_i} = U \leq 1$.

5 Mixed Task Sets

A simple solution would be to schedule asynchronous tasks/events whenever there is time available. However this might lead to starvation.

5.1 Polling Server (RM)

A polling server is an additional periodic task which is used to serve aperiodic requests. An aperiodic task is schedulable if (sufficient but not necessary):

$$\left(1 + \left\lceil \frac{C_a}{C_s} \right\rceil \right) T_s \leq D_a$$

Where C_a, C_s, D_a, T_s are the total computation time of the aperiodic request, the computation time of one server task, the deadline of the aperiodic request, and the period of the server task. It is assumed that one aperiodic request is served before another task arrives.

5.2 Total Bandwidth Server (EDF)

A certain cpu utilization (bandwidth) is reserved for the server. When the kth aperiodic request arrives it receives the deadline

$$d_k = \max(r_k, d_{k-1}) + \frac{C_k}{U_s}$$

where C_k is the total execution time of the request and U_s is the utilization factor of the server. The request is then treated as a normal periodic task.
The schedule is feasible iff $U_p + U_s \leq 1$. Where U_p is the processor utilitation of the periodic tasks.

6 Priority Inheritance Protocol

- Tasks have both nominal priority P_i and active priority $p_i \geq P_i$. The tasks are sorted by nominal priority, highest first.
- Tasks are scheduled by their active priority using FCFS for ties.
- When J_i is blocked, it transmits its active priority p_i to the blocking task, $p_k = p_i$
- When J_k exits the critical section, the highest priority job blocked by J_k is awakened. If no other jobs are blocked by J_k, p_k is set to P_k otherwise to the highest priority of the jobs blocked by J_k.
- Priority inheritance is transitive.

7 Communication

7.1 Random Access

Best sending rate is $p = \frac{1}{n}$ (using maximization of $p\,(1-p)^{n-1}$) leads to utilization of $\frac{1}{e}$.

7.2 TDMA

Statically allocated slots, synchronization done by master.

7.3 CSMA/CD

Detect collitions, do exponential backoff in case of collisions.

7.4 Token Ring

Pass token around, attach to token.

7.5 CSMA/CA

IMAGE HERE

7.6 CSMA/CR

Before message is sent, arbitration happens. During arbitration every client can detect which one is allowed to send. During arbitration, every node sends its unique id and checks whether the result on the bus matches the id. Low bits override high bits, hence the one with the lowest id wins.

7.7 FlexRay

FlexRay uses a combination of TDMA and something close to CSMA/CA alternating, first a static segment using TDMA, then a dynamic segment

7.8 Bluetooth

Frequency hopping in range $2402 + k$ MHz, $0 \leq k \leq 78$, 1600hops per second. Organized in piconets with 1 master and at most 7 slaves, which can be combined into scatternets. A node is a master in at most one piconet and can be a slave in multiple piconets.
The master can only send in even time slots. After each slot, the frequency is changed using a pseudo-random sequence based on the master's device address (unique) and the phase of the master's system clock. Packets are either 1, 3, or 5 slots long, the frequency is only changed after the full packet is sent, however it changes 1, 3, or 5 steps, depending on the packet length.
The initial setup between master and slave is done by the inquiry step, where the master sends it s ID using a special channel sequence, the slave listens and replies with its ID, clock and other information.
Then the connection is started with in the page mode. The master sends its and the slaves address using a special channel sequence, the slave listens whether it hears its address and answers with its own address. The master then sends a frequency hop synchronization packet (FHS) to the slave which allows them to synchronize and connnect.
A slave can be in multiple states, active, hold (not processing date packtes), sniff (awaken in regular time intervals), and park (no connection but still synchronized).