Today’s topic:
REAL TIME SCHEDULING (BASICS)

Question
How to schedule the Tasks such that given timing constraints are satisfied?

Task models
- Non periodic/Aperiodic (three parameters)
  - A: arriving time
  - C: computing time
  - D: deadline (relative deadline)

Overall Structure of Real Time Systems

Task 1  ...  Task n
RTOS/Run-Time System
Hardware

So far, we have talked about
- Programming Languages to implement the Tasks
- Run-Time/Operating Systems to run the Tasks
Constraints on task sets
- Timing constraints: deadline for each task, relative to arriving time or absolute deadline
- Other constraints
  - Precedence constraints
    - Precedence graphs imposed e.g. by input/output relation
  - Resource constraints: mutual exclusion
  - Resource access protocols

Scheduling Problems
Given a set of tasks (ready queue)
1. Check if the set is schedulable
2. If yes, construct a schedule to meet all deadlines
3. If yes, construct an optimal schedule e.g. minimizing response times

Tasks with the same arrival time
Assume a list of tasks
(A,C1, D1)(A,C2, D2) ...(A,Cn,Dn)
that arrive at the same time i.e. A
- How to find a feasible schedule?
- (OBS: there may be many feasible schedules)

Early Due Date first (EDD) [Jackson 1955]
- EDD: order tasks with nondecreasing deadlines.
  - Simple form of EDF (earliest deadline first)
- Example: (1,10)(2,3)(3,5)
  - Schedule: (2,3)(3,5)(1,10)
- FACT: EDD is optimal
  - If EDF can’t find a feasible schedule for a task set, then no other algorithm can, i.e. The task set is non-schedulable.

EDD: Schedulability test
- If C1+C2+...+Ck <=Dk for all k<=n for the schedule with nondecreasing ordering of deadlines, then the task set is schedulable
- Response time for task i, Ri =C1+...+Ci
- Prove that EDD is optimal?

EDD: Examples
- (2, 4)(1,5)(6,10) is schedulable:
  - Feasible schedule: (2,4)(1,5)(6,10)
  - Note that (1,5)(2,4)(6,10) is also feasible
- (1,10)(3,3)(2,5) is schedulable
  - The feasible schedule: (3,3)(2,5)(1,10)
  - Why not shortest task first?
- (4,6)(1,10)(3,5) is not schedulable
  - (3,5)(4,6)(1,10) is not feasible: 3+4 > 6!
Assume that \( R_i \) is the finishing time (relative to the release time) of task \( i \). Note that \( R \) means response time. Let \( L_i = R_i - D_i \) (the lateness for task \( i \)).

**FACT:** EDD is optimal with respect to minimizing the maximum lateness \( L_{\text{max}} = \max(L_i) \) (the general form of optimality of EDD).

Note that even a task set is non-schedulable, EDD may minimize the maximal lateness (minimizes loss for soft tasks?)

**EDD: Exercises**
- Prove: EDD is optimal in finding a feasible schedule
- Program the schedulability test for EDD

**Tasks with different arrival times**
- Assume a list of tasks
  - \( S = (A_1,C_1,D_1)(A_2,C_2,D_2) \ldots (A_n,C_n,D_n) \)

**Preemptive EDF [Horn 1974]:**
- Whenever new tasks arrive, sort the ready queue according to earliest deadlines first at the moment
- Run the first task of the queue if it is nonempty

**FACT:** Preemptive EDF is optimal [Dertouzos 1974] in finding feasible schedules.

**Preemptive EDF: Schedulability test**
- At time \( A_i \), if the list ordered according to EDF (\( A'_1,C'_1,D'_1)(A'_2,C'_2,D'_2) \ldots (A'_i,C'_i,D'_i) \) satisfies \( C'_1 + \ldots + C'_k \leq D'_k \) for all \( k = 1,2,\ldots,i \), then \( S \) is schedulable at time \( A_i \).
- If \( S \) is schedulable at all \( A_i \)'s, \( S \) is schedulable.

**Preemptive EDF: Example**
- Consider \( (1, 5, 11)(2,1,3)(3, 4,8) \)
- Deadlines are relative to arrival times
- At 1, (5,11)
- At 2, (1,3)(4,10)
- At 3, (4,8)(4,9)

**Preemptive EDF: Response time calculation**
- Complicated
- But possible
Preemptive EDF: Exercises

- Write a program to calculate the response times for (non)preemptive EDF.

Preemptive EDF: Optimality

- Assume that $R_i$ is the finishing time (relative to the release time) of task $i$. Note that $R$ means response time. Let $L_i = R_i - D_i$ (the lateness for task $i$).

- FACT: preemptive EDF is optimal with respect to minimizing the maximum lateness $L_{max} = \max(L_i)$ (the general form of optimality of preemptive EDF).

Non preemptive EDF (on-line version)

- Alternative 1: Run a task until it’s finished and then sort the queue according to EDF.
  - The algorithm may be run on-line, easy to implement, less overhead (no more context switch than necessary).
  - However it is not optimal, it may not find the feasible schedule even if it exists e.g., $(0,5,20)(1,1,3)(6,7,30)$: the second task misses its deadline. Note that the feasible schedule: $(1,1,3)(0,5,20)(6,7,30)$.

On-line non preemptive EDF: example

- On-line EDF Schedule:

```
<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival</th>
<th>Event</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>0</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>4</td>
<td>101</td>
</tr>
</tbody>
</table>
```

- Feasible Schedule:

```
<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival</th>
<th>Event</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>0</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>T2</td>
<td>0</td>
<td>4</td>
<td>101</td>
</tr>
<tr>
<td>T3</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
```

On-line non preemptive EDF: Optimal?

- If we only consider non-idle algorithms (CPU waiting only no task to run), is EDF is optimal?
  - Unfortunately no!

- Example:
  - $T_1 = (0, 10, 100)$
  - $T_2 = (0, 1, 101)$
  - $T_3 = (1, 4, 4)$
  - Run $T_1,T_3,T_2$: the 3rd task will miss its deadline
  - Run $T_2,T_3,T_1$: it is a feasible schedule.

Off-line Non preemptive EDF (complete search)

- Alternative 2: the decision should be made according to all the parameters in the whole list of tasks.
  - Consider the example: $(0,5,20)(1,1,3)(6,7,30)$. 

CPU idling
Off-line Non preemptive EDF (NP-hard)

- Unfortunately, to find a feasible non-preemptive schedule for task set with different arrival times is not easy
- The worst case is to test all possible combinations of \( n \) tasks (NP-hard, difficult for large \( n \))

Practical methods: Bratley’s algorithm

- Search until a non-schedulable situation occur, then backtrack [Bratley’s algorithm]
  - simple and easy to implement but may not find a schedule if \( n \) is too big (worst case)

Example (Bratley’s alg.)

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

Heuristic methods: Spring algorithm

- Similar to Bratley’s alg. But
  - Use heuristic function \( H \) to guide the search until a feasible schedule is found, otherwise backtrack: add a new node in the search tree if the node has smallest value according to \( H \) e.g \( H(\text{task } i) = C_i, A_i, D_i \) etc [Spring alg.]
  - However it may be difficult to find the right \( H \)

Example Heuristics

- \( H(T_i) = A_i \) FIFO
- \( H(T_i) = C_i \) SJF
- \( H(T_i) = D_i \) EDF
- \( H(T_i) = D_i + w* C_i \) EDF+SJF
- ...

EDF: + and –

- Simple (+)
- Preemptive EDF, Optimal (+)
- No need for computing times (+)
- On-line and off-line (+)
- Preemptive schedule easy to find (+)
- But preemptive EDF is “difficult” to implement efficiently (-)
  - Must use a list of “timers”, one per task
- Non-preemptive (feasible) schedule difficult to find (-)
  - But minimal context switch (+)
  - And the only way to schedule non-preemptive tasks
Other scheduling algorithms

- Classical ones
  - HPF (priorities = degrees of importance of tasks)
  - Weighted Round Robin
- LRT (Latest Release Time or reverse EDF)
- LST (Least Slack Time first)

Latest Release Time (reversed EDF)

- Release time = arrival time
- Idea: no advantage to completing any hard task sooner than necessary. We may want to postpone the execution of hard tasks e.g. to improve response times for soft tasks.
- LRT: Schedule tasks from the latest deadline to earliest deadline. Treat deadlines as ‘release times’ and arrival times as ‘deadlines’. The latest ‘deadline’ first
  - FACT: LRT is optimal in finding feasible schedule (for preemptive tasks)

LRT: Example

Reverse time: we get the schedule:

T1(9,11)T2(11,13)T3(13,17)T2(17,18)T1(18,20)

OBS: from 0 to 9, soft tasks may be running!

LRT: + and -

- It needs Arrival times (-)
- It got to be an off-line algorithm (-)
- Only for preemptive tasks (-)
- It could optimize Response times for soft tasks (+)

Least slack time first (LST)

- Let $S_i = D_i - C_i$ (the Slack time for task i)
  - $S_i$ is the maximal (tolerable) time that task $i$ can be delayed
- Idea: there is no point to complete a task earlier than its deadline. Other (soft) tasks may be executed first
  - Slack stealing
- LST: order the queue with nondecreasing slack times
- FACT: preemptive LST is optimal in finding feasible schedules

LST: Example

Comment: a task should run until a Slack reaches 0 (to avoid context switch)
And if more than one 0-slack: nonschedulable
LST: + and –

- It needs Computing times (-)
- Only for preemptive tasks (-)
- Not easy to implement! (-)
- But it can run on-line (+)
- and it may improve response times?

Independent tasks

- OBS! we have assumed that tasks are independent!
  - meaning that we can compute them in arbitrary orderings
  - only if the orderings (schedules) are feasible

- All algorithms we have studied so far are applicable
  - only to independent tasks

Summary: scheduling independent tasks

<table>
<thead>
<tr>
<th>Task types</th>
<th>Same arrival times</th>
<th>Preemptive different arrival times</th>
<th>Non preemptive different arrival times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td>For Independent tasks</td>
<td>EDF, Horn 74; O(n) log n, optimal</td>
<td>LST, LRT optimal; EDF, Horn 74; O(n^2), optimal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>For Non Independent tasks</th>
<th>EDF, Horn 74; O(n^2), optimal</th>
</tr>
</thead>
</table>

Dependent tasks

- In practice, tasks are dependent. We often have conditions or constraints e.g.
  - A must be computed before B
  - B must be computed before C and D

- Such conditions are called precedence constraints which can be represented as Directed Acyclic Graphs (DAG) known as Precedence graphs
  - Such graphs are also known as “Task Graph”

Dependent tasks: Examples

- Input/output relation
  - Some task is waiting for output of the others, data flow diagrams

- Synchronization
  - Some task must be finished before the others e.g. It is holding a shared resource

- Other dependence relations (e.g priority-orderings?)

Precedence graph: Example

- A must be computed before B
- B must be computed before C and D
**Precedence graph: Examples**

- A  B  C  D
- A  B  C  D
- Not a precedence graph!

Conjunct and Disjunct join: We will only consider conjunct join!

**AND/OR-precedence graphs**
- AND-node, all incoming edges must be finished first
- OR-node: some of the incoming edges must be finished

**Scheduling under Timing and Precedence constraints**
- Feasible schedules should meet
  - Timing constraints: deadlines and also
  - Precedence constraints: Precedence graphs
- Overlapping area of blue and red is what we need
- Precedence constraints restrict the search area (Guiding!)

Schedules satisfying Precedence constraints

All possible schedules (feasible/infeasible)

Schedules Satisfying Timing constraints

**Dependent tasks with the same arrival times**
- Assume a list of tasks: 
  \((A,C1,D1)(A,C2,D2)\) ...\((A,Cn,Dn)\)
- In addition to the deadlines \(D1...Dn\), the tasks are also constrained by a DAG
- Solution: Latest Deadline First (LDF), Lawler 1973
- FACT: LDF is optimal (in finding feasible schedules)

**Latest Deadline First (LDF)**
- It constructs a schedule from tail to head using a queue:
  - Pick up a task from the current DAG, that
    - Has the latest deadline and
    - Does not precede any other tasks (a leaf)
  - Remove the selected task from the DAG and put it to the queue
- Repeat the two steps until the DAG contains no more tasks.
  Then the queue is a potentially feasible schedule. The last task selected should be run first.
- Note that this is similar to LRT

**LDF: Example**

<table>
<thead>
<tr>
<th>Task</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
LDF: Example

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

LDF: T6,T5,T3,T4,T2

Earliest Deadline First (EDF)

- It is a variant of LDF, but start with the root of the DAG:
  - Pick up a task with earliest deadline among all nodes that have no fathers (the roots)
  - Remove the selected task from the DAG and put it to the queue
- Repeat the two steps until the DAG contains no more tasks. Then the queue is a feasible schedule.

- Unfortunately, EDF is not optimal (see the following example)
EDF: T1

LDF: T1, T3

EDF: T1, T3, T2

LDF: T1, T3, T2, T4, T5, T6

Dependent tasks with different arrival times

Assume a list of tasks:
\[ S = (A_1, C_1, D_1)(A_2, C_2, D_2)...(A_n, C_n, D_n) \]

In addition to the deadlines \( D_1...D_n \), the tasks are also constrained by a DAG

Solution: The Complete Search guided by the DAG
- The Bratley’s algorithm
- The Spring algorithm
**Better algorithms?**

- Assume a list of tasks:
  \[ S = (A_1, C_1, D_1)(A_2, C_2, D_2)...(A_n, C_n, D_n) \]
- In addition to the deadlines \( D_1...D_n \), the tasks are also constrained by a DAG

- **Idea:**
  - Transform the task set \( S \) (constrained by the DAG) to an independent task set \( S^* \) such that \( S \) is schedulable under DAG iff \( S^* \) is schedulable

**Idea: how to transform \( S \) to \( S^* \)?**

- **Idea:**
  - If \( T_i \rightarrow T_j \) is in the DAG i.e. \( T_i \) must be executed before \( T_j \), we replace the arrival time for \( T_j \) and deadline for \( T_i \) with:
    - \( A_j^* = \max(A_j, A_i + C_i) \)
    - \( D_i^* = \min(D_i, D_j - C_j) \)
  - \( T_i \) should be finished early enough to meet the deadline for \( T_j \)

**Algorithm (EDF\(^*\)): transform \( S \) to \( S^* \)**

- Let arrival times and deadlines be *absolute times*
- **Step 1:** Transform the arrival times from roots to leaves
  - For all initial (root) nodes \( T_i \), let \( A_i^* = A_i \)
  - **Repeat:**
    - Pick a node \( T_j \) whose fathers arrival times have been modified. If no such node, stop. Otherwise:
      - Let \( A_j^* = \max(A_j, \max(A_i^* + C_i: T_i \rightarrow T_j)) \)
  - **Step 2:** Transform the deadlines from leaves to roots
    - For all terminal (leafs) nodes \( T_j \), let \( D_j^* = D_j \)
    - **Repeat:**
      - Pick a node \( T_i \) all whose sons deadlines have been modified. If no such node, stop. Otherwise:
        - Let \( D_i^* = \min(D_i, \min(D_j^* - C_j: T_i \rightarrow T_j)) \)
  - **Step 3:** use EDF to schedule \( S^* = (A_1^*, C_1, D_1^*)...(A_n^*, C_n, D_n^*) \)

**EDF\(^*\): optimality**

- **FACT:**
  - \( S \) is schedulable under a DAG iff \( S^* \) is schedulable
  - EDF\(^*\) is optimal in finding a feasible schedule

**Example**

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**EDF\(^*\): Example(1)**

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Step 1:** Modifying the arrival times (top-down)
Step 1: Modifying the arrival times (top-down)

Step 2: Modifying the deadlines (bottom-up)

Step 3: now we don’t need the DAG any more!

Step 3: schedule S* using EDF

Finally we have a schedule: T1,T2,T4,T3,T5,T6
### Summary: scheduling aperiodic tasks

<table>
<thead>
<tr>
<th>Task types</th>
<th>Same arrival times</th>
<th>Preemptive</th>
<th>Non-preemptive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Different arrival times</td>
<td>Different arrival times</td>
</tr>
<tr>
<td>Algorithms for independent tasks</td>
<td>EDF, Jackson 55, $O(n \log n)$</td>
<td>Optimal</td>
<td>(O(n^{**2})) Heuristic</td>
</tr>
<tr>
<td></td>
<td>EDF, Horn 74, $O(n^{**2})$</td>
<td>Optimal</td>
<td>(O(n^{**2})) Heuristic</td>
</tr>
<tr>
<td></td>
<td>Tree search, Bradley 71</td>
<td>(O(n \log n)), optimal</td>
<td>(O(n^{**2})) Heuristic</td>
</tr>
<tr>
<td></td>
<td>LRT, optimal</td>
<td>(O(n^{**2})) Heuristic</td>
<td>(O(n^{**2})) Heuristic</td>
</tr>
<tr>
<td>Algorithms for dependent tasks</td>
<td>EDF, Lowier 73, $O(n^{**2})$</td>
<td>Optimal</td>
<td>(O(n^{**2})) Heuristic</td>
</tr>
<tr>
<td></td>
<td>EDF, Chetto et al 90, $O(n^{**2})$</td>
<td>Optimal</td>
<td>(O(n^{**2})) Heuristic</td>
</tr>
<tr>
<td></td>
<td>Spring</td>
<td>As above</td>
<td>As above</td>
</tr>
</tbody>
</table>