Today’s topic:
REAL TIME SCHEDULING

Overall Structure of Real Time Systems

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

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)

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 all deadlines can be met (schedulability check)
2. If yes, construct a “feasible” 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)
**Earliest 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, which means that the task set is non schedulable.

---

**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!

---

**Tasks with different arrival times**

- Assume a list of tasks
  
  $S = (A_1, C_1, D_1)(A_2, C_2, D_2) ... (A_n, C_n, D_n)$

- Preemptive EDF [Horn 1974]:
  
  - Whenever new tasks arrive, sort the ready queue according to earliest deadlines
  - Run the first task of the queue

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

---

**EDD: Schedulability test**

- If $C_1+C_2+...+C_k <= D_k$ for all $k<=n$ for the schedule with nondecreasing ordering of deadlines, then the task set is schedulable

- Response time for task $i$, $R_i = C_1+...+C_i$

- Prove that EDD is optimal?

---

**EDD: optimality**

- Assume that $R_i$ is the finishing time of task $i$, i.e. response time. Let $L_i = R_i-D_i$ (the lateness for task $i$)

- **FACT**: EDD is optimal, minimizing the maximum lateness $L_{max} = \text{MAX}(L_i)$

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

---

**Preemptive EDF: Schedulability test**

- At any time, order the tasks according to EDF $(A'_1, C'_1, D'_1) ... (A'_i, C'_i, D'_i)$

- If $C'_1+...+C'_k <= D'_k$ for all $k=1,2...i$, then the task set is schedulable at the moment

- If $S$ is schedulable at all time points at which tasks arrive, $S$ 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: Optimality

- Assume that $R_i$ is the finishing time/response time of task $i$. Let $L_i = R_i - D_i$ (the lateness for task $i$)

- **FACT**: preemptive EDF is optimal in minimizing the maximum lateness $L_{max} = \max_i(L_i)$

On-line non-preemptive EDF

- 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 it exists.

On-line non-preemptive EDF: Example

<table>
<thead>
<tr>
<th>A</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

On-line non-preemptive EDF: Optimal?

- If we only consider non-idle algorithms (CPU waiting only if no tasks 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)

- The decision should be made according to all the parameters in the whole list of tasks
Off-line Non preemptive EDF (complete search, NP-hard)

- The decision should be made according to all the parameters in the whole list of tasks
- 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 occurs, then backtrack [Bratley’s algorithm]
  - simple and easy to implement but may not find a schedule if $n$ is too big (worst case)

Heuristic methods

- 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 (Bratley’s alg.)

<table>
<thead>
<tr>
<th>Task</th>
<th>$A_i$</th>
<th>$C_i$</th>
<th>$D_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>T3</td>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>T4</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Example Heuristics

- $H(T_i) = A_i$ FIFO
- $H(T_i) = C_i$ SJF
- $H(T_i) = D_i$ EDF
- $H(T_i) = D_i + w \cdot C_i$ 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 (-)
    - Need a list of “timers”, one per task,
    - Overheads for context switch
  - Nonpreemptive schedule difficult to find (-)
    - But minimal context switch (+)
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

![Example of LRT scheduling](image)

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 (+)

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>EDF, Horn '74 O(n**2), Optimal LRT, LRT optimal</td>
</tr>
</tbody>
</table>

Dependent tasks

- We often have conditions or constraints on tasks 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

Precedence graph: Examples

- A must be computed before B
- B must be computed before C and D

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

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!)

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:
  1. Pick up a task from the current DAG, that
     - Has the latest deadline and
     - Does not precede any other tasks (a leaf)
  2. 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>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

LDF: T6

LDF: T6, T5

---

LDF: Example

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

LDF: T6

LDF: T6, T5

---

LDF: Example

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

LDF: T6, T5
Earliest Deadline First (EDF)

- It is a variant of LDF, but start with the root of the DAG:
  1. Pick up a task with earliest deadline among all nodes that have no fathers (the roots)
  2. 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)
**Better algorithms?**

- Assume a list of tasks: $S = (A_1, C_1, D_1)(A_2, C_2, D_2)...(A_3, C_n, D_n)$
  
- 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

**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 up a node $T_j$ whose fathers’ arrival times have been modified. If no such node, stop. Otherwise:
    - $A_j^* = \max(A_j, \max(A_i^* + C_i : T_i \rightarrow T_j))$
- **Step 2:** Transform the deadlines from leaves to roots
  - For all terminal (leaf) nodes $T_j$, let $D_j^* = D_j$
  - **REPEAT:**
    - Pick up a node $T_i$ all whose sons deadlines have been modified. If no such node, stop. Otherwise:
    - $D_i^* = \min(D_i, \min(D_j^* - C_j : T_i \rightarrow T_j))$
- **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
**EDF*: Example (1)**

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

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

**EDF*: Example (2)**

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>A*</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Step 2: Modifying the deadlines (bottom-up)

**EDF*: Example (3)**

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>D*</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>A*</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

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>Different arrival times</th>
<th>Non preemptive</th>
<th>Different arrival times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms for</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Independent tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDF, Jackson 55</td>
<td>$O(n \log n)$, optimal</td>
<td>EDF, Horn 74</td>
<td>$O(n^2)$, optimal</td>
<td>Tree search, Bratley'71</td>
<td>$O(n^2)$, optimal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LST, optimal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LRT, optimal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithms for</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependent tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDF, Lawler 73</td>
<td>$O(n^2)$</td>
<td>LDF*</td>
<td></td>
<td>Spring</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chetto et al 90</td>
<td>$O(n^2)$, optimal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>As above</td>
</tr>
</tbody>
</table>