Real-Time Operating Systems M

10. Real-Time: Aperiodic Task Scheduling
Notice

The course material includes slides downloaded from:

http://codex.cs.yale.edu/avi/os-book/


and

http://retis.sssup.it/~giorgio/rts-MECS.html


which has been edited to suit the needs of this course.

The slides are authorized for personal use only.

Any other use, redistribution, and any for profit sale of the slides (in any form) requires the consent of the copyright owners.
Introduction

- A variety of algorithms for scheduling real-time aperiodic tasks on single processor

Definitions to keep in mind:

- **Schedule**: an assignment of tasks to the processor, so that each task is executed until completion. Represented as a step function (time slices).
  - A schedule $\sigma$ is said to be **feasible** if *all* the tasks can complete according to a set of specified constraints.
  - A set of tasks $\Gamma$ is said to be **schedulable** if there exists at least one algorithm that can produce a feasible schedule for it.

- **Aperiodic task**: A type of task that consists of a sequence of identical jobs (instances), activated at irregular intervals.

Facts

- The general scheduling problem is NP hard
- Polynomial time algorithms can be found under particular conditions
Graham’s Notation

\[ \alpha \mid \beta \mid \gamma \]

- **\alpha**: machine environment
  - e.g., number of processors; parallel architecture; …
- **\beta**: constraint on tasks
  - e.g., preemption, limited resources, precedence,…
- **\gamma**: optimality criterion
  - \( \varphi_{\text{max}} \) or \( \sum \varphi_j \); e.g., \( \varphi = \) response time \( R \), lateness \( L \), tardiness, …

- **Examples**:
  - \( 1 \mid \text{sync} \mid L_{\text{max}} \)
  - \( 2 \mid \text{prec, sync} \mid \sum f_j \)
  - \( 1 \mid \text{no_preem} \mid \text{feasible} \)
Lateness

\[ L_i = f_i - d_i \]

\[ L_{\text{max}} = \max_i (L_i) \]

- If \( (L_{\text{max}} < 0) \) then no task misses its deadline
Classical Scheduling Policies

- First Come First Served
- Shortest Job First
- Round Robin
- Priority Scheduling

- Not suited for real-time systems
  - Why?
Why is FCFS not suitable for RTOS?

- Very unpredictable
  - Response time strongly depends on task arrivals

\[
\begin{array}{cccc}
\tau_1 & \tau_2 & \tau_3 \\
R_1 = 20 & R_2 = 26 & R_3 = 26 \\
\end{array}
\]

\[
\begin{array}{cccc}
\tau_3 & \tau_2 & \tau_1 \\
R_1 = 26 & R_2 = 8 & R_3 = 2 \\
\end{array}
\]
Why is SJF not suitable for RTOS?

- Minimizes the average response time
- Not optimal in the sense of feasibility
Round Robin?

- Interactive systems
- Time sharing
  - Each task runs as it was executing alone on a virtual processor \( n \) times slower than the real one
  - \( R_i \sim nC_i \)

- If \( Q > \max(C_i) \) \( \Rightarrow \) RR = FCFS
- If \( Q \sim \) context switch time \( \Rightarrow \) \( R_i \sim 2nC_i \)

- Very unpredictable
  - Response time strongly depends on quantum, task arrivals, workload
Priority Scheduling?

- Each task is assigned a priority
- The task with highest priority is executed
- Tasks with the same priority are served FCFS

Problem: starvation

- Solution: aging
  - $p_i \sim 1/C_i \rightarrow$ SJF
  - $p_i \sim 1/r_i \rightarrow$ FCFS

- No guarantee
Multi-level scheduling

- High priority: system tasks
  - Priority

- Medium priority: interactive task
  - RR

- Low priority: batch task
  - FCFS

CPU
Real-Time Algorithms

- **Static** algorithms: consider relative deadlines $D_i$
- **Dynamic** algorithms: consider absolute deadlines $d_i$
Earliest Due Date

- Consider the following problem:
  - single processor machine
  - n independent tasks (consisting of a single job)
  - synchronous arrival times
  - performance measure: achieve minimum lateness

\[ 1 \mid \text{sync} \mid L_{max} \]

- **Jackson's algorithm (EDD):**
  
  "select the task with the earliest relative deadline"
Let’s Have a Look at Lateness…

\[ \sigma \]

\[ \sigma' \]

\[ r_0 \quad f_a' < f_b \quad f_b' = f_a \quad d_a \quad d_b \]

EDD Optimality

is the minimum value achievable by any algorithm
EDD Optimality

- $\sigma \rightarrow \sigma' \rightarrow \sigma'' \rightarrow \ldots \rightarrow \sigma^*$
- $L_{\text{max}}(\sigma) \rightarrow L_{\text{max}}(\sigma') \rightarrow L_{\text{max}}(\sigma'') \rightarrow \ldots \rightarrow L_{\text{max}}(\sigma^*)$
- $\sigma^* = \sigma_{\text{EDD}}$
- $L_{\text{max}}(\sigma_{\text{EDD}})$ is the minimum value achievable by any algorithm

- Given a set of $n$ independent tasks, all with the same arrival time, any algorithm that executes the tasks in order of nondecreasing deadlines is optimal with respect to minimizing the maximum lateness

(Jackson’s theorem)
## Exercise

<table>
<thead>
<tr>
<th></th>
<th>J₁</th>
<th>J₂</th>
<th>J₃</th>
<th>J₄</th>
<th>J₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$d_i$</td>
<td>3</td>
<td>10</td>
<td>7</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>
Exercise

<table>
<thead>
<tr>
<th>Cᵢ</th>
<th>J₁</th>
<th>J₂</th>
<th>J₃</th>
<th>J₄</th>
<th>J₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dᵢ</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
Guarantee test for EDD (off line)

- Need to show that, in the worst case, all tasks can complete before their deadlines
  - for all $i$, $f_i \leq d_i$
- Assume tasks listed by decreasing deadlines
  - $f_i = ?$
  - $n$ conditions (one per task)...

- Complexity of building EDD schedule = sorting, $O(n \log n)$
- Complexity of guarantee test: $O(n)$
Exercise

- Write an algorithm for finding the maximum Lateness $L_{\text{max}}$ of a task set scheduled by the EDD algorithm

  - assume the task set is $J[n]$, with $J[i].C$ and $J[i].D$ representing worst-case computation time and relative deadline of the i-th scheduled task
Quizzes

- True or False?
  1. A real-time application that is feasible on a given processor can become infeasible when running on a slower processor.
  2. A real-time application that is feasible on a given processor can become infeasible when running on a faster processor.
  3. A real-time application that is feasible with $L_{\text{max}} = -X$ can become infeasible when introducing a delay $\Delta < X$.
  4. A heuristic scheduling algorithm guarantees that, if there is a feasible schedule, the algorithm will find it.
  5. Heuristic algorithms have greater complexity than optimal algorithms.
  6. EDD is a scheduling algorithm defined for a single-processor machine.
  7. EDD is a heuristic algorithm.
  8. EDD is a guaranteed algorithm.
Quizzes

True or False?

Consider a set of independent jobs $J = \{J_1, J_2, \ldots, J_n\}$, with identical arrival time, to be scheduled on a single-processor machine:

1. There is always a feasible schedule
2. There is always a schedule (not necessarily feasible)
3. There may be an unfeasible EDD schedule
4. It is possible to improve any given schedule in terms of $L_{\text{max}}$
5. It is possible to improve any non-EDD schedule in terms of $L_{\text{max}}$
6. It is possible to find a schedule with minimum $L_{\text{max}}$ using a procedure with complexity below $n^2$
Earliest Deadline First

- What if tasks are not synchronous, but can have arbitrary times?
- Preemption?

\[
1 \mid \text{preem} \mid L_{\text{max}}
\]

- **Horn's algorithm (EDF):**
  
  “select the task with the earliest absolute deadline”

- Assume
  - arbitrary arrival times
  - dynamic priority \((d_i \text{ depends on arrival})\)
  - full preemption
What can we say about EDF?

- Consider time slices of 1 unit of time each
- Notation:
  - $\sigma(t)$: the task executing in the slice $[t, t+1)$
  - $E(t)$: the ready task that at $t$ has earliest deadline
  - $t_E(t)$: the time ($\geq t$) at which the next slice of $E(t)$ begins execution

![Diagram of task scheduling]

Dertouzos transformation algorithm preserves schedulability, in fact:

- This is obvious for the advanced slice!
- For the postponed slice, the slack cannot decrease:
Dertouzos Transformation

\[
\text{for } (t = 0 \text{ to } D_{\text{max}} - 1) \text{ if } (\sigma(t) \neq E(t)) \{ \\
\quad \sigma(t_E) = \sigma(t) \\
\quad \sigma(t) = E(t) \\
\}
\]

- Each transposition preserves schedulability

- Complexity of building EDF schedule using transpositions: \(O(n^2)\)
EDF Optimality

- $\sigma \rightarrow \sigma' \rightarrow \sigma'' \rightarrow \ldots \rightarrow \sigma^*$
- $L_{\text{max}}(\sigma) \rightarrow L_{\text{max}}(\sigma') \rightarrow L_{\text{max}}(\sigma'') \rightarrow \ldots \rightarrow L_{\text{max}}(\sigma^*)$
- $\sigma^* = \sigma_{\text{EDF}}$
- $L_{\text{max}}(\sigma_{\text{EDF}})$ is the minimum value achievable by any algorithm

- Given a set of $n$ independent tasks with arbitrary arrival times, any algorithm that at any instant executes the tasks with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness

(Horn’s theorem)
About Optimality and Lateness

- An algorithm A is **optimal in the sense of feasibility** if it generates a feasible schedule, if there exists one.
  - If an optimal algorithm (in the sense of feasibility) produces an unfeasible schedule, then no algorithm can produce a feasible schedule.
- If an algorithm A minimizes $L_{\text{max}}$ then A is also optimal in the sense of feasibility. The opposite is **not** true.
# Exercise

<table>
<thead>
<tr>
<th></th>
<th>J₁</th>
<th>J₂</th>
<th>J₃</th>
<th>J₄</th>
<th>J₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>aᵢ</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Cᵢ</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>dᵢ</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>10</td>
<td>9</td>
</tr>
</tbody>
</table>
Guarantee test for EDF (on line)

- Need to show that, in the worst case, all tasks can complete before their deadlines
  - for all \( i \), \( f_i \leq d_i \)
- Assume all tasks listed by increasing deadlines
- Let \( c_i(t) \) be the remaining worst-case execution time of task \( J_i \)
  - \( f_i(t) = ? \)
  - \( n \) conditions (one per task)…
Non Preemptive Scheduling

- Under non preemptive execution, EDF is not optimal

Feasible schedule

EDF

- Should need to be *clairvoyant* to be optimal!
- Still, NP-EDF is optimal among non-idle scheduling algorithms
  - I.e., if we forbid to leave CPU idle when there are ready tasks
Non Preemptive Scheduling

- Finding a feasible schedule is an \textbf{NP-hard} problem
- Treated off-line with \textbf{tree search algorithms}
  - \textbf{Branch-and-bound} techniques to reduce search space
  - \textbf{Pruning}

\[ \text{depth} = n \]
\[ \# \text{ leaves} = n! \]
\[ \text{complexity: } O(n \ n!) \]
Bratley’s algorithm

- Reduces the **average** complexity by pruning techniques
  - Does that mean it’s ok for *online* usage in RT systems??

1 | no-preem | feasible

- **Pruning:**
  
  “abandon a branch if the addition of any node to the current path causes a missed deadline”

(a branch is abandoned unless it is **strongly feasible**: remains feasible after adding any one of the remaining nodes)

- Bratley’s algorithm can be used to produce a **task activation list**
## Exercise

<table>
<thead>
<tr>
<th></th>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$J_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$C_i$</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$d_i$</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>
Spring Algorithm

- Used in Spring kernel (Stankovic & Ramamritham)
- Example of heuristic algorithm
  - Goal: find a feasible schedule when tasks have different types of constraints (precedence, resource, arbitrary arrivals, non-preemptive properties, and importance levels).
  - Idea: apply function $H$ to each remaining task, choose smallest $H$
**Spring Algorithm**

- **H** could be many things
  - FCFS?
  - SJF?
  - EDF?
  - ESTF?
  - EDF+SJF?
  - EDF+ESTF?

- **Earliest Available Time (EAT)**
  - EAT determines earliest time a job can start execution without blocking on shared resources
  - \( T_{est}(i) = \max [a_i, \max_k \text{EAT}(i,k)] \)
  - Uses binary array of resources for each job: \([R_1, R_2, \ldots, R_r]\)
    - \( R_j = 1 \) if job uses \( R_j \) in exclusive mode
**Spring Algorithm**

- **Eligibility**: $E_i$ to handle precedences:
  - $E_i = 1$ only if all ancestors have finished
  - $E_i = \infty$ otherwise

- Sample heuristic functions:
  - $H = E_i(w_1 r_i + w_2 D_i)$
  - $H = E_i(w_1 C_i + w_2 d_i)$

- To reduce worst-case complexity: limit number of evaluations for $H$
  - Exhaustive search: $O(n \times n!)$
  - Heuristic search: $O(n^2)$
  - Heuristic with *max k backtracks*: $kn^2 \rightarrow O(n^2)$
Exercise

Consider a single-processor machine.

1. Find the path(s) produced by Spring with \( H = a_i + C_i + D_i \) and \( k=2 \)

2. Is there a heuristic function that produces a feasible schedule with \( k=1 \) (no backtracking)?
Exercise

Consider a single-processor machine.

1. Find the path produced by Spring with \( H = E(a_i + C_i + D_i) \) and \( k=4 \)

2. What is the minimum \( k \) to find a feasible solution with this heuristic function?
Exercise

Consider a 2-processor machine, 2 shared resources, $R = [R_1, R_2]$, and no task/resource preemption.

1. Show, as a Gantt chart, the output produced by Spring with $H = \text{ESTF}$, and $k=1$

<table>
<thead>
<tr>
<th></th>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$J_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>$C_i$</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>$d_i$</td>
<td>18</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>$R_i$</td>
<td>$[0,1]$</td>
<td>$[1,1]$</td>
<td>$[1,0]$</td>
<td>$[0,0]$</td>
</tr>
</tbody>
</table>
Scheduling with Precedence Constraints

- Finding optimal schedule with precedence relations is in general **NP-hard**
- There are **polynomial time algorithms** under either assumption:
  - synchronous arrival
  - preemptive scheduling

- Latest Deadline First (LDF, Lawler)
  \[ 1 | \text{prec, sync} | L_{\text{max}} \]

- EDF with Precedence Constraints (EDF*, Chetto²)
  \[ 1 | \text{prec, preem} | L_{\text{max}} \]
Latest Deadline First (LDF)

1 | prec, sync | \( L_{\text{max}} \)

- Constructs the schedule \textit{from the tail} of the precedence graph:
  - select the task with the \textit{latest deadline} among the nodes with \textit{no successors}, or whose successors have \textit{all been selected}
- Example (assume duration 1 for all tasks)

\[ \text{Graph:} \]

- EDF vs LDF?
LDF Optimality

- Is $\sigma$ an LDF schedule?
- $\Gamma$: tasks without successors
  - no precedence relation within
  - what if $\ldots \sigma \rightarrow \sigma^*$?

- Need to visit graph to determine $\Gamma$ after each allocation
  $\rightarrow$ complexity is $O(n^2)$
EDF* (with Precedence Constraints)

- Scheduling $n$ tasks with precedence constraints and dynamic activations can be (optimally) done in polynomial time only if tasks are preemptable.

$$1 \mid \text{prec,preem} \mid L_{\text{max}}$$

- Idea:
  - Assume arrival times to be known
  - Transform precedence into timing constraints (DAG $\rightarrow$ deadlines)
  - Apply EDF
**EDF* Transformations**

- Postpone the arrival time of a successor:

```
\[ r_A^* = r_A + C_A \]
```

- Advance the deadline of a predecessor

```
\[ d_A^* = d_A - C_B \]
```

The idea is:

1. For all root nodes, set \( r_i^* = r_i \).
2. Select a task \( W_i \) such that all its immediate predecessors have been modified, else exit.
3. Set \( r_i^* = \max \{ r_i, \max (r_k^* + C_k) \} \).
4. Repeat from line 2.
EDF* Transformations

Postpone the arrival time of a successor:

1. For all leaves, set \( d^*_i = d_i \).
2. Select a task \( \tau_i \) such that all its immediate successors have been modified, else exit.
3. Set \( d^*_i = \min \{ d_i, \min (d^*_k - C_k) \} \).
4. Repeat from line 2.

Advance the deadline of a predecessor

1. For all root nodes, set \( r^*_i = r_i \).
2. Select a task \( \tau_i \) such that all its immediate predecessors have been modified, else exit.
3. Set \( r^*_i = \max \{ r_i, \max (r^*_k + C_k) \} \).
4. Repeat from line 2.
EDF* Optimality

- Does EDF* preserve feasibility?
- Does EDF* satisfy precedence constraints?
## Exercise

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_i)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(C_i)</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>(d_i)</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>prec</td>
<td>A→C</td>
<td>B→C</td>
<td>B→D</td>
<td>C→E</td>
<td>C→F</td>
<td>D→F</td>
<td>D→G</td>
</tr>
</tbody>
</table>
## Synopsis: Aperiodic Task Scheduling

<table>
<thead>
<tr>
<th></th>
<th>sync</th>
<th>preem</th>
<th>no_preem</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDD (Jackson)</td>
<td>$O(n \log n)$</td>
<td>$O(n^2)$</td>
<td>NP-EDF (Jeffay)</td>
</tr>
<tr>
<td></td>
<td>$L_{max}$</td>
<td>$L_{max}$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>non optimal</td>
</tr>
<tr>
<td>LDF (Lawler)</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>Tree search (Bratley)</td>
</tr>
<tr>
<td></td>
<td>$L_{max}$</td>
<td>$L_{max}$</td>
<td>$O(n \cdot n!)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>feasible</td>
</tr>
<tr>
<td>EDF* (Chetto)</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>Spring (Stankovic &amp; Ramamritham)</td>
</tr>
<tr>
<td></td>
<td>$L_{max}$</td>
<td>$L_{max}$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Heuristic</td>
</tr>
</tbody>
</table>