Lecture 13
Task Assignment & Scheduling

Feng-Li Lian
NTU-EE
Feb10 – Jun10

The Question:
• Will my real-time application really meet its timing constraints or requirements?

The Problem:
• Given a set of tasks, precedence constraints, resource requirements, their execution times, release times, and deadlines, and one or more processing systems

• Assign tasks to different processing systems
• Design a feasible/optimal allocation/scheduling on the processing system

Shin & Ramanathan 94

Krishna & Shin 97
### Definitions:

- **Tasks:**
  - Consume resources (e.g., processor time, memory, input data), and
  - Put out one or more results

- **Precedence Constraints:**
  - Specify if any task(s) needs to precede other tasks
  - Represented by the means of a precedence graph

- **Resource Requirements:**
  - All tasks require
    - some execution time on a processor,
    - a certain amount of memory or
    - access to a bus (network)
  - Exclusive or non-exclusive

---

### Definitions:

- **Release Time:**
  - The time at which all the data that are required to begin executing the task are available

- **Deadline:**
  - The time by which the task must complete its execution
  - Hard or soft, depending on the nature of the corresponding task

- **Relative Deadline:**
  - The absolute deadline minus the release time

---

### Definitions:

- **Periodic:**
  - The task is released periodically
  - Only to run exactly once every period; not required for being run exactly one period apart

- **Sporadic:**
  - Not periodic, but at irregular intervals
  - Characterized by an upper bound on the rate at which the tasks may be invoked

- **Aperiodic:**
  - Same as sporadic, OR
  - For not periodic and w/o upper bound on the invocation time

---

### Definitions:

- **Feasible:**
  - A task assignment/schedule is said to be feasible if all tasks start after their release times and complete before their deadlines

- **A-Feasible:**
  - If an assignment/schedule algorithm A results in a feasible schedule

- **Offline or Online Scheduling:**
  - Schedule in advance
  - Schedule as the tasks arrive
Definitions:

- **Priority:**
  - A function of the nature of the tasks themselves and the current state of the controlled process
- **Static- & Dynamic-Priority Algorithms:**
  - Task priority does not change within a mode
  - Task priority can change with time
- **Preemptive & Non-preemptive Schedule:**
  - Tasks can be interrupted by other tasks (and then resumed)
  - Task schedule must be run to completion or until it gets blocked over a resource
  - Causing anomalies

Objective in Scheduling:

- For **non-real-time** applications
  - Minimize the total time required to execute all the tasks in the application
- For **real-time** applications
  - Meet the timing constraints of the individual tasks

Characteristics in RT Scheduling Algorithms:

- Uniprocessor or multiprocessor
  - For multi-processors, shared memory or message-passing system
- Periodic or aperiodic
- Preemptible or non-preemptible
- Criticality
- Independence
- Resource
- Placement constraints
- Strictness of deadlines

Terminologies:

- Feasibility
- Optimality
- Lateness
- Absolute/relative/effective deadlines
- Absolute/effective release times
- Periodic, sporadic, aperiodic
Components of Task Model:

- **Precedence relation:** \( T' \)
  - Set of tasks that must be completed before task \( T \) can begin its execution
- **Resource requirements:**
  - Processor, memory, bus, disk, etc.
  - Exclusive
  - Shared (read-only, read-write)
- **Schedule \( S \):**
  - \( \{ \text{set of processors} \} \times \{ \text{time} \} \rightarrow \{ \text{set of tasks} \} \)
  - Off-line or online
  - Static or dynamic priority algorithm
  - Preemptive or non-preemptive
  - Uniprocessor or multiprocessor

Commonly Used RT Scheduling Approaches:

- **Time-driven:**
  - Determines when to execute which job
  - All parameters of hard RT jobs are fixed and known
  - A schedule is computed off-line and stored for use at runtime
- **Weighted round-robin:**
  - For high-speed networks, where length of a round = sum of all weights
- **Priority-driven:**
  - Assigns priorities to jobs and executes jobs in priority order
  - Static priority assignment:
    - Rate or Deadline Monotonic (RM or DM)
  - Dynamic priority assignment:
    - Earliest Deadline First (EDF), Minimum Laxity First (MLF)

Four Paradigms of Scheduling Approaches:

- **Static table-driven scheduling:**
- **Static priority preemptive scheduling:**
- **Dynamic planning-based scheduling:**
- **Dynamic best effort scheduling:**

RTOS should have:

- CPU scheduling
- Resource allocation
- Predictability, requiring bounded OS primitives

RT Scheduling involves the allocation of resources and time to tasks
Analyzing Scheduling Algorithms:

- Performance metrics
- Scheduling paradigms
- Scheduling algorithms
- Other important scheduling issues

Performance metrics

- Static non-real-time systems
  - Minimize schedule length
- Dynamic non-real-time systems
  - Minimize response time
  - Increase throughput
- Both static & dynamic real-time systems
  - Achieve timeliness

Task characteristics:
- Computation times
- Resource requirements
- Importance levels (or priorities, criticalness)
- Precedence relationships
- Communication requirements
- Timing constraints

In static scheduling:
- Since schedule off-line
- So, meet all deadlines
- If exists,
  > Maximize average earliness
- If not,
  > Minimize average tardiness

In dynamic scheduling:
- Since information is not known a priori
- So, maximize number of arrivals meeting deadlines
Performance metrics

- Levels of predictability:
  - Using a particular approach how well can we predict that the tasks will meet their deadlines?

- Schedulability analysis or feasibility checking
  - Statically or dynamically

Four Paradigms of Scheduling Approaches:

- Static table-driven scheduling:
  - Static schedulability analysis
  - Resulting schedule (or table) used at run time

- Static priority-based preemptive scheduling:
  - Static schedulability analysis
  - No explicit schedule
  - Highest priority task first

- Dynamic planning-based scheduling:
  - Feasibility checked at run time
  - Dynamically accept arriving task if feasible schedule found

- Dynamic best effort scheduling
  - No feasibility check
  - Try its best to meet deadlines & may be aborted

Static table-driven scheduling:

- For periodic tasks
- Given task characteristics,
  - Table is constructed by using, e.g., search heuristics
  - With identifying start & completion times
  - Tasks dispatched according to table
- Highly predictable, but highly inflexible
Dynamic planning-based scheduling:
- With flexibility and predictability
- For new arrival,
  - Try to create a schedule containing previously guaranteed tasks as well as the new arrival
  - If fail, take other actions

Dynamic best effort scheduling
- A priority-driven preemptive approach
  > e.g., use deadlines as priorities & without any planning
  - Priority is computed based on task’s characteristics
  - Schedule based on priority
  - Confidence via extensive simulations

  - Lack of predictability and sub-optimality
  - Try its best to meet deadlines
  - But, do NOT know whether a timing constraint will be met

Uniprocessor Scheduling Algorithms

Notations:

\[ T_i = (I_i, P_i, e_i, d_i) \]

- \( n \): Number of tasks in the task set
- \( e_i \): Execution time of task \( T_i \)
- \( P_i \): Period of task \( T_i \), if it is periodic
- \( I_i \): \( k \)th period of (periodic) task \( T_i \) begins at time \( I_i + (k-1)P_i \), where \( I_i \) is call the phasing of task \( T_i \)
- \( d_i \): Relative deadline of task \( T_i \)
- \( D_i \): Absolute deadline of task \( T_i \)
- \( r_i \): Release time of task \( T_i \)
- \( h_T(t) \): Sum of the execution times of task iterations in task set \( T \) that have their absolute deadlines \( \leq t \)
Assumptions:
- A1: Fully preemptible with negligible costs
  - Can preempt any task at any time and resume it later without penalty
- A2: CPU is the only resource to deal with
  - i.e., don’t care with memory, I/O, etc.
- A3: Independent task
  - i.e., no precedence constraints between tasks
- A4: All periodic tasks
- A5: Relative deadline = period

Example: (a two-task system)

<table>
<thead>
<tr>
<th>Time</th>
<th>Task</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Example: (a 3-task system)

<table>
<thead>
<tr>
<th>Time</th>
<th>Task</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>I</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>P</td>
<td>2</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>e</td>
<td>0.5</td>
<td>2</td>
<td>1.75</td>
</tr>
</tbody>
</table>

Since \( P_1 < P_2 < P_3 \), priority: \( T_1 > T_2 > T_3 \)
Example: (a 3-task system)

- Check sufficient schedulability condition

\[ U = \frac{0.5}{2} + \frac{2}{6} + \frac{1.75}{10} = 0.7583 \]

\[ e_1 \leq P_1 \]

\[ t = \left\lceil \frac{t}{P_1} \right\rceil e_1 + e_2 \quad \& \quad t \in [0, P_2] \]

\[ t \geq \left\lceil \frac{t}{P_1} \right\rceil e_1 + e_2 \quad \& \quad t \leq P_2 \]

Check only \( t \) at multiples of \( P_1 \)

\[ t = \left\lceil \frac{t}{P_1} \right\rceil e_1 + \left\lceil \frac{t}{P_2} \right\rceil e_2 + e_3 \quad \& \quad t \in [0, P_3] \]

\[ t \geq \left\lceil \frac{t}{P_1} \right\rceil e_1 + \left\lceil \frac{t}{P_2} \right\rceil e_2 + e_3 \quad \& \quad t \leq P_3 \]

Check only \( t \) at multiples of \( P_1 \) & \( P_2 \)

Necessary (& Sufficient) Schedulability Conditions

- \( T_1 \): feasibly scheduled

- \( T_2 \): feasibly scheduled

- \( T_3 \): feasibly scheduled
### Necessary (& Sufficient) Schedulability Conditions

- **Task Set**: \( T = \{T_1, T_2, \ldots, T_n\} \) where \( T_i = (P_i, e_i) \)
- WLOG, assume \( P_1 \leq P_2 \leq \cdots \leq P_n \)
- \( T_i \) released at \( t = 0 \)
- \( T_i \)'s completion time: \( t_c \)
- Within time \( t_c \), \( T_i \) is preempted by each higher priority task \( T_j \)
  
  \[ t_c = \sum_{j=1}^{i-1} e_j \frac{t_c}{P_j} + e_i \]

### Theorem

**Given** \( n \) periodic tasks

- \( P_1 \leq P_2 \leq \cdots \leq P_n \), &
- \( W_i(t) = \sum_{j=1}^{i} e_j \left[ \frac{t}{P_j} \right] \)

**THEN**, task \( T_i \): feasibly scheduled using RM

**IFF** \( L_i = \min_{0 < t \leq P_i} \frac{W_i(t)}{t} \leq 1 \)

\[ \Rightarrow \text{In fact, only need to compute } W_i(t) \text{ at } \tau_i = \left\{ k \frac{P_j}{P_i} \mid j = 1, 2, \ldots, i; \ k = 1, \ldots, \left\lceil \frac{P_i}{P_j} \right\rceil \} \]

### Example: (a 4-task system)

<table>
<thead>
<tr>
<th>Time/Task</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>100</td>
<td>150</td>
<td>210</td>
<td>400</td>
</tr>
<tr>
<td>e</td>
<td>20</td>
<td>30</td>
<td>80</td>
<td>100</td>
</tr>
</tbody>
</table>

### Set of points of interest:

- \( \tau_1 = \{100\} \)
- \( \tau_2 = \{100, 150\} \)
- \( \tau_3 = \{100, 150, 200, 210\} \)
- \( \tau_4 = \{100, 150, 200, 210, 300, 400\} \)
Example: (a 4-task system)

Schedulability Conditions:

- \( T_1 \) is RM-Schedulable iff \( e_1 < 100 \)
- \( T_2 \) is RM-Schedulable iff \( e_1 + e_2 \leq 100 \) OR \( 2e_1 + e_2 < 150 \)
- \( T_3 \) is RM-Schedulable iff \( e_1 + e_2 + e_3 \leq 100 \) OR \( 2e_1 + e_2 + e_3 \leq 150 \) OR \( 2e_1 + 2e_2 + e_3 \leq 200 \) OR \( 3e_1 + 2e_2 + e_3 < 210 \)
- \( T_4 \) is RM-Schedulable iff ...

With Sporadic Tasks:

- Define fictitious highest-periodic task & execution time

- RM with Deferred Server (DS):

Earliest Deadline First (EDF) Algorithm:

- Assign higher priorities to tasks whose absolute deadline is the earliest
- Optimal dynamic-priority scheduling algorithm
- Tasks: periodic or aperiodic
- Schedulable on a uniprocessor by the EDF iff:

\[
U = \sum_{i=1}^{n} \frac{e_i}{P_i} \leq 1
\]
### Uniprocessor Scheduling Algorithms

**Example: (a 3-task system)**

<table>
<thead>
<tr>
<th>Time</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival Time</td>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Execution Time</td>
<td>10</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Absolute Deadline</td>
<td>30</td>
<td>10</td>
<td>25</td>
</tr>
</tbody>
</table>

at $t = 4$, $D_2 < D_1$, $T_2$ preempts $T_1$

at $t = 5$, $D_2 < D_3$, $T_2$ continues, $T_3$ waits

at $t = 7$, $D_3 < D_1$, $T_3$'s term

at $t = 17$, $T_1$ resumes

---

**Allowing for Precedence & Exclusion Conditions:****

<table>
<thead>
<tr>
<th>Task $T_i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_i$</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>$D_i$</td>
<td>6</td>
<td>7</td>
<td>20</td>
<td>21</td>
<td>27</td>
<td>28</td>
</tr>
</tbody>
</table>

**Multiple Task Versions: Primary & Alternative**

- Better-quality service v.s. just-acceptable service

<table>
<thead>
<tr>
<th>Time</th>
<th>Task</th>
<th>Primary</th>
<th>Alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst-case run time</td>
<td>20</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Average run time</td>
<td>7</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Period</td>
<td>15</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

**Multiple Task Versions: 5-task system**

<table>
<thead>
<tr>
<th>Task $T_i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l(i)$</td>
<td>10</td>
<td>10</td>
<td>15</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>$\alpha(i)$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>$P(i)$</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>40</td>
<td>40</td>
</tr>
</tbody>
</table>

Run-time limit of primary version

Worst-case run-time of alternative version

---

**Run-time limit**

---

**Period**

---

Krishna & Shin 97
### Increased Reward with Increased Service (IRIS):

<table>
<thead>
<tr>
<th>Task</th>
<th>( m_i )</th>
<th>( o_i )</th>
<th>( r_i )</th>
<th>( D_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>19</td>
</tr>
</tbody>
</table>

Identical Linear Reward Function

\[
R_i(x) = \begin{cases} 
0 & \text{if } x < m_i \\
 x - m_i & \text{if } m_i \leq x \leq o_i + m_i \\
o_i & \text{if } o_i + m_i < x 
\end{cases}
\]

### Rate Monotonic (RM, static priority):
- Task set: periodic, preemptible, deadline = period
- Statically assign higher priorities to tasks with lower periods
- It is schedulable under RM if its total processor utilization \( \leq n(2^{\frac{1}{n}} - 1) \)
- RM is an optimal static-priority uniprocessor scheduling algorithm

### Rate Monotonic Deferred Server (DS):
- Similar to RM
- Handle both periodic and aperiodic tasks
- Allot some time slots for aperiodic tasks

### Earliest Deadline First (EDF, dynamic priority):
- Tasks: preemptible
- The earliest the deadline, the higher the priority
- Optimal if preemption is allowed and jobs do not contend for resources
- If a task set is not schedulable on a single processor by EDF, no other processor can successfully schedule that task set

### Precedence and Exclusion Conditions:
- Take precedence conditions into account
- Algorithm might be with exclusion conditions such as some tasks are not allowed to interrupt some, irrespective of priority

### Multiple Task Versions:
- In some cases, the system has primary and alternative versions of some tasks
- Varying in execution time or quality of output they provide
- Primary version for top-quality output, alternative for lower-quality

### Increased Reward with Increased Service (IRIS):
- Algorithm can be stopped early and output still useful
- Quality of output: a monotonically nondecreasing function of the execution time

---

**Krishna & Shin 97**

**04/08/03**
Uniprocessor Scheduling Algorithms

- Rate Monotonic (RM, static priority):
- Earliest Deadline First (EDF, dynamic priority):
- Rate Monotonic Deferred Server (DS):
- Multiple Task Versions:
- Precedence and Exclusion Conditions:
- Increased Reward with Increased Service (IRIS):

Multiprocessor Scheduling Algorithms

- Task Assignment:
  - The optimal assignment of tasks to processors is, in almost all practical cases, an NP-complete problem
  - Do with heuristic procedures:
    - Allocate the tasks
    - Check their feasibility
    - If not feasible, modify the allocation
  - CANNOT guarantee that a feasibly scheduled allocation can be found
- Need to account for communication costs

Utilization Balancing Algorithm:
- Next-Fit Algorithm for RM Scheduling:
- Bin-Packing Algorithm for EDF:
- Myopic Offline Scheduling (MOS) Algorithm:
- Focused Addressing & Bidding (FAB) Algorithm:
- Buddy Strategy:
- Assignment with Precedence Constraints:
Next-Fit Algorithm for RM Scheduling:
- Tasks: preemptible
- With RM uniprocessor scheduling algorithm
- Set of tasks \(\rightarrow\) Various classes
- Set of processors \(\rightarrow\) Each task class

Bin-Packing Algorithm for EDF:
- Tasks: preemptible
- Total utilizations \(\leq\) a given threshold
- Threshold: the uniprocessor scheduling algorithm is able to schedule the tasks assigned to each processor
- Minimize the number of processors needed
  > Many algorithms exist for solving it
  > The First Fit Decreasing (FFD) algorithm

Example: 4-Class & 11-Task

<table>
<thead>
<tr>
<th>Class</th>
<th>Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>(0.41, 1.00)</td>
</tr>
<tr>
<td>C2</td>
<td>(0.26, 0.41)</td>
</tr>
<tr>
<td>C3</td>
<td>(0.19, 0.26)</td>
</tr>
<tr>
<td>C4</td>
<td>(0.00, 0.19)</td>
</tr>
</tbody>
</table>

By RM on each processor

<table>
<thead>
<tr>
<th>Processor</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>T1</td>
</tr>
<tr>
<td>p2</td>
<td>T2, T5, T6</td>
</tr>
<tr>
<td>p3</td>
<td>T11</td>
</tr>
<tr>
<td>p4</td>
<td>T3, T4, T7, T8, T9, T10</td>
</tr>
<tr>
<td>p5</td>
<td>T6</td>
</tr>
</tbody>
</table>

Example: 4-Class & 11-Task

\[ L = ( T1, T6, T2, T5, T11, T10, T3, T9, T8, T4, T7 ) \]

\[ U = ( U1, U2, U3, U4, \ldots) \]

containing the total utilizations of processor \(p_i\) in \(U_i\)

Number of processors used by the FFD algorithm \(\frac{11}{9} = 1.22\)

Number of processors used by optimal algorithm \(\frac{9}{9} = 1\)
### Myopic Offline Scheduling (MOS) Algorithm:
- Can deal with nonpreemptible tasks
- Build up a schedule tree and based on a search process to find feasible schedule minimizing a heuristic function $H$ such as execution time, deadline, start time, laxity, etc.
- For $n$ tasks, the schedule tree has $n+1$ levels (including the root)

### Example: 5-(nonpreemptive)-Task & 2-Processor

<table>
<thead>
<tr>
<th>Task</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_i$</td>
<td>10</td>
<td>15</td>
<td>9</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>$D_i$</td>
<td>15</td>
<td>20</td>
<td>18</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>

$H(i) = r_i$
### Focused Addressing & Bidding Algorithm:
- Tasks arrive at the **individual** processors
- If one processor finds itself **unable** to meet the deadline or other **constraints**, the task must be offloaded.
- Then it tries to **offload** some of its workload onto other processors
- **By announcing** which task(s) it would like to offload and waiting for other processors to offer to take them up

### Buddy Strategy:
- Roughly the same as the **focused addressing** algorithm
- **Processor load:** under-loaded, fully loaded, overloaded
- Overloaded processors ask under-loaded to take over some

### Assignment with Precedence Constraints:
- Take precedence into account
- Use a **trial-and-error** process to assign tasks that **communicate heavily** with one another
- So that communication **costs** are **minimized**

### Example: 1-Task & 8-Subtask, 2-Processor

<table>
<thead>
<tr>
<th>Subtask</th>
<th>( e_i )</th>
<th>( D_i )</th>
<th>LFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>4</td>
<td>-</td>
<td>7</td>
</tr>
<tr>
<td>s1</td>
<td>10</td>
<td>-</td>
<td>24</td>
</tr>
<tr>
<td>s2</td>
<td>15</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>s3</td>
<td>4</td>
<td>-</td>
<td>26</td>
</tr>
<tr>
<td>s4</td>
<td>18</td>
<td>-</td>
<td>42</td>
</tr>
<tr>
<td>s5</td>
<td>3</td>
<td>-</td>
<td>42</td>
</tr>
<tr>
<td>s6</td>
<td>6</td>
<td>-</td>
<td>32</td>
</tr>
<tr>
<td>s7</td>
<td>3</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>s8</td>
<td>8</td>
<td>40</td>
<td>40</td>
</tr>
</tbody>
</table>

**LFT:** Latest finishing time

\[
\frac{e_i + e_j}{c_{ij}} < k_i
\]
### Static algorithms:
- Periodic tasks with *hard* deadlines
- Not applicable to *aperiodic* tasks b/c timing info unknown

### Dynamic algorithms:
- Centralized
  - All tasks distributed by *one central processor* into others
  - So, processors' load is known and deadlines are guaranteed
- Distributed
  - Tasks arrive *independently* at each processor
  - *Transfer policy*: guarantee constraints of incoming tasks
  - *Location policy*: find other processors if not schedulable
  - *Information policy*: collect & maintain state info of others

#### Utilization Balancing Algorithm:
- Tasks: preemptible
- Assign tasks to processors *one by one* such that at the end of each step
  utilizations of various processors nearly balanced

#### Next-Fit Algorithm for RM Scheduling:
- Tasks: preemptible
- With RM uniprocessor scheduling algorithm
- Set of tasks $\rightarrow$ Various classes
- Set of processors $\rightarrow$ Each task class

#### Bin-Packing Algorithm for EDF:
- Tasks: preemptible
- Total utilizations $\leq$ a given threshold
- *Threshold*: the uniprocessor scheduling algorithm
  is able to schedule the tasks assigned to each processor

#### Myopic Offline Scheduling (MOS) Algorithm:
- Can deal with *nonpreemptible* tasks
- Build up a schedule tree and based on a search process
  to find feasible schedule *minimizing* a heuristic function
  such as execution time, deadline, start time, laxity, etc.
Focused Addressing & Bidding (FAB) Algorithm:
- Tasks arrive at the individual processors
- If one processor finds itself unable to meet the deadline or other constraints,
  - Then it tries to offload some of its workload onto other processors
- By announcing which task(s) it would like to offload and waiting for other processors to offer to take them up

Buddy Strategy:
- Roughly the same as the focused addressing algorithm
- Processor load: under-loaded, fully loaded, overloaded
- Overloaded ask under-loaded to take over some

Assignment with Precedence Constraints:
- Take precedence into account
- Use a trial-and-error process to assign tasks that communicate heavily with one another
- So that communication costs are minimized

Utilization Balancing Algorithm:

Next-Fit Algorithm for RM Scheduling:

Bin-Packing Algorithm for EDF:

Myopic Offline Scheduling (MOS) Algorithm:

Focused Addressing & Bidding (FAB) Algorithm:

Buddy Strategy: