Concurrent Process Model

Implementations:

- General purpose processor
- Custom processor

Single Processor
- Share processor time and GPU
- Achieve necessary execution

Multiple Processors

Single Processor Multiple Processes
- Convert to sequential program by embedding scheduling in code
- Use a multitasking operating system
  - RTOS: schedules processes, allocates storage, interfaces to peripherals; etc.
  - Guarantee execution rates are met

Terminology

Process: Heavyweight Process
  - Own address space
  - System resources

Thread: Lightweight Process
  - Subprocess within process
  - Program counter, stack, registers
  - Shared address space
  - Small compared to heavyweight process
Real-time Scheduling

**Deadlines**
- **Hard Deadline:** A time constraint is called hard if not meeting that constraint could result in catastrophe
- **Soft Deadline:** All other deadlines

**Periodic Task:** A task that must executed at a defined period (every $p$ units of time)

**Sporadic Task:** Task that must execute at unpredictable times

**Pre-emption**

**Preemptive Scheduler:** Task execution can be interrupted before the task has been completed
- use for systems that have long running tasks
- need to respond quickly to external events

**Non-preemptive Scheduler:** Tasks are executed until they are done
- may lead to delayed responses to external events

**Schedulability:** A set of tasks is schedulable if a schedule exists such that the constraints are met
Earliest Deadline First (EDF)

- Scheduling approach that will execute the task with the earliest deadline at any given point
- preemptive scheduling

Example:

<table>
<thead>
<tr>
<th>Arrival</th>
<th>Duration</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>T2</td>
<td>4</td>
<td>23</td>
</tr>
<tr>
<td>T3</td>
<td>5</td>
<td>29</td>
</tr>
</tbody>
</table>

T2 has earlier deadline than T1
T3 has earlier deadline than T1
T2 continues to executed, earlier deadline than T1 or T3

Dynamic Scheduling:

- when a task arrives:
  1. Insert into scheduling queue of runnable (ready) task sorted by absolute deadlines
  2. If task is inserted into head of queue, current task is preempted
Periodic Scheduling:

Definitions:

\[ p_i = \text{period of task } T_i \]
\[ c_i = \text{execution time of } T_i \]
\[ d_i = \text{deadline of task } T_i \]
\[ t_i = \text{time between arrival of } T_i \text{ and when } T_i \text{ must finish executing} \]

Rate Monotonic Priority Assignment

- Each process is assigned a unique priority based on its period.
- Tasks with shorter periods have higher priority.
- \( P_i \rightarrow P_j \Rightarrow \) Priority \( P_i < P_j \)
- Typically used for systems where deadline = period.
- Optimal in that any task set can be scheduled with a fixed priority scheme, it can be scheduled with a rate monotonic priority assignment.

Example Priority

<table>
<thead>
<tr>
<th>Process/Task</th>
<th>Period (P)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>29</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>60</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>42</td>
<td>4</td>
</tr>
<tr>
<td>d</td>
<td>105</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>75</td>
<td>2</td>
</tr>
</tbody>
</table>

Rate Monotonic Scheduling

- All tasks that have hard deadlines are periodic.
- Tasks are independent.
- \( c_i \geq 0 \) for all tasks.
- \( c_i \) is known for all tasks and is constant.

Policy/Algorithm:

- Priority of a task is a monotonically decreasing function of the period runnable/ready.
- The highest priority task will execute at any given time.
Utilization Test:

- Utilization is a measure of the amount of time required to execute tasks.

- Simple Test:
  \[ U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1 \]
  
  \( n \) is the number of tasks

  - High utilizations are harder to implement

- Better measure:
  \[ U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n \left( \frac{2^n - 1}{n} \right) \]
  \[ \leq 0.69 \text{ as } n \to \infty \]

Example:

<table>
<thead>
<tr>
<th>( P_i )</th>
<th>( D_i )</th>
<th>( C_i )</th>
<th>( P_r )</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>50</td>
<td>12</td>
<td>1</td>
<td>0.24</td>
</tr>
<tr>
<td>b</td>
<td>40</td>
<td>10</td>
<td>2</td>
<td>0.25</td>
</tr>
<tr>
<td>c</td>
<td>30</td>
<td>10</td>
<td>3</td>
<td>0.35</td>
</tr>
</tbody>
</table>

\( U = 82\% \) (above threshold for 3 tasks at 0.78)

- Process set fails utilization test
Another Example

\[\begin{align*}
T_i & | P_i | C_i | P_{ri} | U \\
A & 30 & 32 & 1 & .4 \\
B & 40 & 5 & 2 & .125 \\
c & 16 & 4 & 3 & .25
\end{align*}\]

\[U = .775 < .78\text{ threshold for 3 tasks}\]
\[L\text{ will meet all deadlines}\]

Yet Another Example

\[\begin{align*}
T_i & | P_i | C_i | P_{ri} | U \\
A & 30 & 40 & 1 & .5 \\
B & 40 & 10 & 2 & .25 \\
c & 20 & 5 & 3 & .25
\end{align*}\]

\[U = 1 \Rightarrow \text{above threshold for 3 tasks}\]
\[L\text{ but we can still meet all deadlines}\]

Utilization Test

-2A task set passes utilization test, it \textit{will} meet deadlines
-2B task set fails utilization test, it \textit{MAY} miss deadlines
Deadline Nonpreemptive Scheduling

Optimal for task sets where \( d_i \leq p_i \) for all tasks

Priority: \( \text{Priority}_i \Rightarrow d_i \leq d_j \)

Response Time Analysis

- Can schedule a process set if
  \( R_i \leq D_i \) for all tasks

Response Time

\[ R_i = C_i + I_i \]

\( I \Rightarrow \) Interference from higher priority tasks

- During \( R_i \), each higher priority task \( j \) will execute a number of times

\[ \text{Releases} = \left\lceil \frac{R_i}{P_j} \right\rceil \]

- Total interference of task \( T_i \) = \( \left\lceil \frac{R_i}{P_j} \right\rceil C_j \)

\[ R_i = C_i + \sum_{j \in \text{hp}(i)} \left\lceil \frac{R_i}{P_j} \right\rceil C_j \]

where \( \text{hp}(i) \) is the set of higher priority tasks than \( T_i \)

WCET - worst case execution time

- Obtained by measurement, but it is hard to find the worst case

- Obtained by analysis

Needs detailed model of processor, cache, pipelines, memory, etc.
a:
\[ R_a = 3 \]

b:
\[ \omega_b^0 = 3 \]
\[ \omega_b^1 = 3 + \frac{5}{7} \cdot 3 = 6 \]
\[ \omega_b^2 = 3 + \frac{6}{7} \cdot 3 = 6 \]
\[ R_b = 6 \]

c:
\[ \omega_c^0 = 5 \]
\[ \omega_c^1 = 5 + \frac{5}{7} \cdot 3 + \frac{8}{10} \cdot 3 = 11 \]
\[ \omega_c^2 = 5 + \frac{11}{7} \cdot 3 + \frac{11}{10} \cdot 3 = 14 \]
\[ \omega_c^3 = 5 + \frac{14}{7} \cdot 3 + \frac{14}{10} \cdot 3 = 17 \]
\[ \omega_c^4 = 5 + \frac{17}{7} \cdot 3 + \frac{17}{10} \cdot 3 = 20 \]
\[ \omega_c^5 = 5 + \frac{20}{7} \cdot 3 + \frac{20}{10} \cdot 3 = 20 \]
\[ R_c = 20 \]
\( R_e = 3 \leq 9 \quad \checkmark \)
\( R_0 = 12 \leq 12 \quad \checkmark \)

\( w_0^0 = 9 \)
\( w_0^0 = 9 + \left[ \frac{9}{30} \right] 3 = 12 \)
\( w_0^1 = 9 + \left[ \frac{18}{30} \right] 3 = 12 \)

\( R_A = 16 \quad 16 \leq 20 \quad \checkmark \)

\( w_A^0 = 4 \)
\( w_1 = 4 + \left[ \frac{4}{30} \right] 3 - \left[ \frac{4}{18} \right] 9 = 16 \)
\( w_A^2 = 4 + \left[ \frac{16}{30} \right] 2 + \left[ \frac{16}{18} \right] 9 = 16 \)

\( R_B = 50 \quad \neq 35 \quad (\text{fail}) \)

\( w_B^0 = 5 \)
\( w_B^1 = 5 + \left[ \frac{5}{30} \right] 3 + \left[ \frac{5}{18} \right] 9 + \left[ \frac{5}{20} \right] 4 = 21 \)
\( w_B^2 = 5 + \left[ \frac{25}{30} \right] 3 + \left[ \frac{25}{18} \right] 9 + \left[ \frac{25}{20} \right] 4 = 34 \)
\( w_B^3 = 5 + \left[ \frac{35}{30} \right] 3 + \left[ \frac{35}{18} \right] 9 + \left[ \frac{35}{20} \right] 4 = 37 \)
\( w_B^4 = 5 + \left[ \frac{35}{30} \right] 3 + \left[ \frac{35}{18} \right] 9 + \left[ \frac{35}{20} \right] 4 = 40 \)
\( w_B^5 = 5 + \left[ \frac{45}{30} \right] 3 + \left[ \frac{45}{18} \right] 9 + \left[ \frac{45}{20} \right] 4 = 50 \)
\( w_B^6 = 5 + \left[ \frac{55}{30} \right] 3 + \left[ \frac{55}{18} \right] 9 + \left[ \frac{55}{20} \right] 4 = 50 \)

\( u = 9 \)
\( u_B = 4 \left( 2^{1/4} - 1 \right) \approx 0.76 \)
\( 0.9776 \)
\( F_{u, 1/4} \)
but so was less than or equal to point so...

what if \( p_i = P_i \)?

\[
R_0 = 9 \quad \text{< 18} \checkmark
\]

\[
R_4 = 13 \quad \text{< 20} \checkmark
\]

\[
w^0 = 4
\]

\[
w^1 = 4 + \left[ \frac{4}{18} \right] 9 = 13
\]

\[
w^2 = 4 + \left[ \frac{13}{18} \right] 9 = 13
\]

\[
R_6 = 16 \quad \text{< 30} \checkmark
\]

\[
w^3 = 3
\]

\[
w^4 = 3 + \left[ \frac{3}{18} \right] 9 + \left[ \frac{3}{20} \right] 9 = 16
\]

\[
w^5 = 3 + \left[ \frac{16}{18} \right] 9 + \left[ \frac{16}{20} \right] 9 = 16
\]

\[
R_8 = 50 \quad 50 = 50 \checkmark
\]

\[
w^6 = 5 + \left[ \frac{5}{18} \right] 9 + \left[ \frac{5}{20} \right] 9 + \left[ \frac{5}{30} \right] 9 = 21
\]

\[
w^7 = 5 + \left[ \frac{21}{18} \right] 9 + \left[ \frac{21}{20} \right] 9 + \left[ \frac{21}{30} \right] 9 = 34
\]

\[
w^8 = 5 + \left[ \frac{35}{18} \right] 9 + \left[ \frac{35}{20} \right] 9 + \left[ \frac{35}{30} \right] 9 = 37
\]

\[
w^9 = 5 + \left[ \frac{37}{18} \right] 9 + \left[ \frac{37}{20} \right] 9 + \left[ \frac{37}{30} \right] 9 = 48
\]

\[
w^{10} = 5 + \left[ \frac{46}{18} \right] 9 + \left[ \frac{46}{20} \right] 9 + \left[ \frac{46}{30} \right] 9 = 50
\]

\[
w^{11} = 5 + \left[ \frac{50}{18} \right] 9 + \left[ \frac{50}{20} \right] 9 + \left[ \frac{50}{30} \right] 9 = 50
\]
How to measure RTOS performance

Context Switch: Time to save current task/threads context, find highest priority thread, restore context

Interrupt Latency Range: The interrupts are disabled

RTOS System Services:

Immediate Response: Time to process request immediately

InTask Suspend: Time to process request when thread is suspended

InThread Resumed: Time to process request when suspended thread is resumed as a result

InThread resumed and context switch: suspended higher-priority thread is resumed resulting in a context switch
Back to the Basics: How to measure real-time performance.

#1 factor considered by developers when selecting an RTOS

Real-time Performance: speed with which an RTOS can complete its functions in response to an event.

Events are typically grouped as interrupts.

Let's on an event, we must respond to the event.

**Interrupt Processing**

- suspend current task
- save task related data
- transfer control to ISR
- perform to execute ISR
- determine next task to execute (may not be suspended task)
- clear interrupt flag
- transfer control to selected task

System Services: scheduling, message passing, etc.