Scheduling

- **Topics**
  - Simple process model
  - The cyclic executive approach
  - Process-based scheduling
  - Utilization-based schedulability tests
  - Response time analysis for FPS and EDF
  - Worst-case execution time
  - Sporadic and aperiodic processes

- **Goal**
  - To understand the role that scheduling and schedulability analysis plays in predicting that real-time applications meet their deadlines
  - Process systems with $D < T$
  - Process interactions, blocking and priority ceiling protocols
  - An extendible process model
  - Dynamic systems and on-line analysis
  - Programming priority-based systems
Scheduling

• In general, a scheduling scheme provides two features:
  – An algorithm for ordering the use of system resources (in particular the CPUs)
  – A means of predicting the worst-case behaviour of the system when the scheduling algorithm is applied

• The prediction can then be used to confirm the temporal requirements of the application
Simple Process Model

- The application is assumed to consist of a fixed set of processes
- All processes are periodic, with known periods
- The processes are completely independent of each other
- All system's overheads, context-switching times and so on are ignored (i.e., assumed to have zero cost)
- All processes have a deadline equal to their period (that is, each process must complete before it is next released)
- All processes have a fixed worst-case execution time
Standard Notation

B  Worst-case blocking time for the process (if applicable)
C  Worst-case computation time (WCET) of the process
D  Deadline of the process
I  The interference time of the process
J  Release jitter of the process
N  Number of processes in the system
P  Priority assigned to the process (if applicable)
R  Worst-case response time of the process
T  Minimum time between process releases (process period)
U  The utilization of each process (equal to C/T)
a–z The name of a process
Cyclic Executives

• One common way of implementing hard real-time systems is to use a **cyclic executive**

• Here the design is concurrent but the code is produced as a collection of procedures

• Procedures are mapped onto a set of **minor** cycles that constitute the complete schedule (or **major** cycle)

• Minor cycle dictates the minimum cycle time

• Major cycle dictates the maximum cycle time

Has the advantage of being fully deterministic
Consider a Process Set

<table>
<thead>
<tr>
<th>Process</th>
<th>Period, $T$</th>
<th>Computation Time, $C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>25</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>25</td>
<td>8</td>
</tr>
<tr>
<td>c</td>
<td>50</td>
<td>5</td>
</tr>
<tr>
<td>d</td>
<td>50</td>
<td>4</td>
</tr>
<tr>
<td>e</td>
<td>100</td>
<td>2</td>
</tr>
</tbody>
</table>
Cyclic Executive

loop
    wait_for_interrupt;
    procedure_for_a; procedure_for_b; procedure_for_c;
    wait_for_interrupt;
    procedure_for_a; procedure_for_b; procedure_for_d;
    procedure_for_e;
    wait_for_interrupt;
    procedure_for_a; procedure_for_b; procedure_for_c;
    wait_for_interrupt;
    procedure_for_a; procedure_for_b; procedure_for_d;
end loop;
Time-line for Process Set

![Diagram of process timeline with interrupts]

- Process Set 1: a, b, c
- Process Set 2: a, b, d, e
- Process Set 3: a, b, c

Interrupts occur at specific points in the timeline.
Properties

- No actual processes exist at run-time; each cycle is just a sequence of procedure calls.
- The procedures share a common address space and can thus pass data between themselves. This data does not need to be protected (via a semaphore, for example) because concurrent access is not possible.
- All “process” periods must be a multiple of the minor cycle time.
- You might have to break up processes to make them fit; this decomposition can be very sensitive to process details.
Issues with Cycle Executives

• The difficulty of incorporating processes with long periods; the major cycle time is the maximum period that can be accommodated without secondary schedules

• Sporadic activities are difficult (impossible!) to incorporate

• The cyclic executive is difficult to construct and difficult to maintain — it is a NP-hard problem

• Any “process” with a sizable computation time will need to be split into a fixed number of fixed sized procedures (this may cut across the structure of the code from a software engineering perspective, and hence may be error-prone)

• More flexible scheduling methods are difficult to support

• Determinism is not required, but predictability is
Process-Based Scheduling

• Two principal scheduling approaches
  – Fixed-Priority Scheduling (FPS)
  – Earliest Deadline First (EDF)
**Fixed-Priority Scheduling (FPS)**

- This is the most widely used approach and is the main focus of this course
- Each process has a fixed, *static*, priority which is computed pre-run-time
- The runnable processes are executed in the order determined by their priority
- In real-time systems, the “priority” of a process is derived from its temporal requirements, not its importance to the correct functioning of the system or its integrity
Earliest Deadline First (EDF) Scheduling

- The runnable processes are executed in the order determined by the absolute deadlines of the processes.

- The next process to run is the one with the shortest (nearest) deadline.

- Although it is usual to know the relative deadlines of each process (e.g. 25ms after release), the absolute deadlines are computed at run time and hence the scheme is described as dynamic.
Preemption and Non-preemption

- With priority-based scheduling, a high-priority process may be released during the execution of a lower priority one.
- In a preemptive scheme, there will be an immediate switch to the higher-priority process.
- With non-preemption, the lower-priority process will be allowed to complete before the other executes.
- Preemptive schemes enable higher-priority processes to be more reactive, and hence they are preferred.
- Alternative strategies allow a lower priority process to continue to execute for a bounded time.
- These schemes are known as deferred preemption or cooperative dispatching.
FPS and Rate Monotonic Priority Assignment

• Each process is assigned a (unique) priority based on its period; the shorter the period, the higher the priority

• i.e., for two processes $i$ and $j$,

\[ T_i < T_j \Rightarrow P_i > P_j \]

• This assignment is optimal in the sense that if any process set can be scheduled (using pre-emptive priority-based scheduling) with a fixed-priority assignment scheme, then the given process set can also be scheduled with a rate monotonic assignment scheme

• Note, priority 1 is the lowest (least) priority in this notation.
## Example Priority Assignment

<table>
<thead>
<tr>
<th>Process</th>
<th>Period, $T$</th>
<th>Priority, $P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>25</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>
Utilisation-Based Analysis

- For D=T task sets only
- A simple **sufficient but not necessary** schedulability test exists

\[
U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N (2^{1/N} - 1)
\]

\[
U \leq 0.69 \quad \text{as} \quad N \rightarrow \infty
\]

[Liu & Layland, 1973]
# Utilization Bounds

<table>
<thead>
<tr>
<th>N</th>
<th>Utilization bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100.0%</td>
</tr>
<tr>
<td>2</td>
<td>82.8%</td>
</tr>
<tr>
<td>3</td>
<td>78.0%</td>
</tr>
<tr>
<td>4</td>
<td>75.7%</td>
</tr>
<tr>
<td>5</td>
<td>74.3%</td>
</tr>
<tr>
<td>10</td>
<td>71.8%</td>
</tr>
</tbody>
</table>

Approaches 69.3% asymptotically
## Process Set A

<table>
<thead>
<tr>
<th>Process</th>
<th>Period (T)</th>
<th>Computation Time (C)</th>
<th>Priority (P)</th>
<th>Utilization (U)</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>0.33</td>
</tr>
</tbody>
</table>

- The combined utilization is 0.82 (or 82%)
- This is above the threshold for three processes (0.78) and, hence, this process set fails the utilization test
Time-line for Process Set A

Process

- a
- b
- c

Time

0 10 20 30 40 50 60

Process Release Time
- Process Completion Time
  - Deadline Met
  - Deadline Missed

Preempted
Executing
Gantt Chart for Process Set A

Time

0 10 20 30 40 50
**Process Set B**

<table>
<thead>
<tr>
<th>Process</th>
<th>Period T</th>
<th>Computation C</th>
<th>Priority P</th>
<th>Utilization U</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>80</td>
<td>32</td>
<td>1</td>
<td>0.400</td>
</tr>
<tr>
<td>b</td>
<td>40</td>
<td>5</td>
<td>2</td>
<td>0.125</td>
</tr>
<tr>
<td>c</td>
<td>16</td>
<td>4</td>
<td>3</td>
<td>0.250</td>
</tr>
</tbody>
</table>

- The combined utilization is 0.775
- This is below the threshold for three processes (0.78) and, hence, this process set will meet all its deadlines
## Process Set C

<table>
<thead>
<tr>
<th>Process</th>
<th>Period $T$</th>
<th>Computation Time $C$</th>
<th>Priority $P$</th>
<th>Utilization $U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>80</td>
<td>40</td>
<td>1</td>
<td>0.50</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>20</td>
<td>5</td>
<td>3</td>
<td>0.25</td>
</tr>
</tbody>
</table>

- The combined utilization is 1.0
- This is above the threshold for three processes (0.78) but the process set will meet all its deadlines
Task families

• There is a general rule here. We can improve the sensitivity of the Liu-Layland test by counting only task families, not individual tasks.

• A task family is a set of tasks whose periods are all integer multiples of the shortest period in the family.

• In process set C, there is only one task family, so the utilisation bound is 1.
Time-line for Process Set C

Process

- **a**:
  - Timeline: 0 to 80
  - Activity distribution:
    - 0-20: Light grey
    - 20-40: Olive green
    - 40-60: Light grey
    - 60-80: Olive green

- **b**:
  - Timeline: 0 to 50
  - Activity distribution:
    - 0-10: Light grey
    - 10-30: Olive green
    - 30-50: Light grey

- **c**:
  - Timeline: 0 to 30
  - Activity distribution:
    - 0-10: Olive green
    - 10-20: Light grey
    - 20-30: Olive green
A newer utilisation-based analysis for RMS

- A sufficient but not necessary schedulability test is

\[ \prod_{i=1}^{N} \left( \frac{C_i}{T_i} + 1 \right) \leq 2 \]  

[Bini et al., 2007]

- This is sometimes called the Hyperbolic Bound.
## Process Set D

<table>
<thead>
<tr>
<th>Process</th>
<th>Period</th>
<th>Computation Time</th>
<th>Priority</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>76</td>
<td>32</td>
<td>1</td>
<td>0.421</td>
</tr>
<tr>
<td>b</td>
<td>40</td>
<td>5</td>
<td>2</td>
<td>0.125</td>
</tr>
<tr>
<td>c</td>
<td>16</td>
<td>4</td>
<td>3</td>
<td>0.250</td>
</tr>
</tbody>
</table>

- The combined utilization is 0.796
- This is above the threshold for three processes (0.78)
- But, \(1.421 \times 1.125 \times 1.25 = 1.998 < 2\), so process set is schedulable by the Bini et al. test.
Criticism of Utilisation-based Tests

- Not exact
- Not general
- BUT it is $O(N)$

The test is **sufficient** but not **necessary**
Utilization-based Test for EDF

\[ \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1 \]

A much simpler test

- Superior to FPS as it can support high utilizations.
- FPS is easier to implement as priorities are static.
- EDF is dynamic and requires a more complex run-time system which will have higher overhead.
- It is easier to incorporate processes without deadlines into FPS; giving a process an arbitrary deadline is more artificial.
- It is easier to incorporate other factors into the notion of priority than it is into the notion of deadline.
- During overload
  - FPS is more predictable; Low priority process miss their deadlines first,
  - EDF is unpredictable; a domino effect can occur in which a large number of processes miss deadlines,
Response-Time Analysis for FPS

Here task $i$'s worst-case response time, $R$, is calculated first and then checked (trivially) with its deadline.

\[
R_i \leq D_i
\]

\[
R_i = C_i + I_i
\]

Where $I$ is the interference from higher priority tasks.
Calculating R

During $R$, each higher priority task $j$ will execute a number of times:

$$\text{Maximum Number of Releases} = \left\lfloor \frac{R_i}{T_j} \right\rfloor$$

The ceiling function $\left\lfloor \cdot \right\rfloor$ gives the smallest integer not less than the fractional number to which it is applied. So the ceiling of $1/3$ is 1, of $6/5$ is 2, and of $6/3$ is 2.

Total interference by task $j$ is given by:

$$\left\lfloor \frac{R_i}{T_j} \right\rfloor C_j$$
Response Time Equation

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

Where \( hp(i) \) is the set of tasks with priority higher than task \( i \).

Solve iteratively:

\[ w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left( \frac{w_i^n}{T_j} \right) C_j \]

This must converge or fail (>\( T_i \)) in a finite number of iterations.
Process Set D

<table>
<thead>
<tr>
<th>Process</th>
<th>Period</th>
<th>Computation Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>7</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>12</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>20</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ R_a = 3 \]

\[ w_b^0 = 3 \]

\[ w_b^1 = 3 + \left[ \frac{3}{7} \right] 3 = 6 \]

\[ w_b^2 = 3 + \left[ \frac{6}{7} \right] 3 = 6 \]

\[ R_b = 6 \]

\[ w_c^0 = 5 \]

\[ w_c^1 = 5 + \left[ \frac{5}{7} \right] 3 + \left[ \frac{5}{12} \right] 3 = 11 \]

\[ w_c^2 = 5 + \left[ \frac{11}{7} \right] 3 + \left[ \frac{11}{12} \right] 3 = 14 \]

\[ w_c^3 = 5 + \left[ \frac{14}{7} \right] 3 + \left[ \frac{14}{12} \right] 3 = 17 \]

\[ w_c^4 = 5 + \left[ \frac{17}{7} \right] 3 + \left[ \frac{17}{12} \right] 3 = 20 \]

\[ w_c^5 = 5 + \left[ \frac{20}{7} \right] 3 + \left[ \frac{20}{12} \right] 3 = 20 \]

\[ R_c = 20 \]
Revisit: Process Set C

<table>
<thead>
<tr>
<th>Process</th>
<th>Period T</th>
<th>Computation Time C</th>
<th>Priority P</th>
<th>Response time R</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>80</td>
<td>40</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td>b</td>
<td>40</td>
<td>10</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>c</td>
<td>20</td>
<td>5</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

- The combined utilization is 1.0
- This was above the utilization threshold for three processes (0.78), therefore it failed the test
- The response time analysis shows that the process set will meet all its deadlines
Response Time Analysis

- Is necessary and sufficient

- If the process set passes the test they will meet all their deadlines; if they fail the test then, at run-time, a process will miss its deadline (unless the computation time estimations themselves turn out to be pessimistic)
Response time and Utilisation

- Two processes:
  \[ R_2 = C_2 + \left\lfloor \frac{R_2}{T_1} \right\rfloor C_1 \]

- In the worst case,
  \[ T_1 = C_2 + C_1 \]

- Also, in the worst case,
  \[ T_2 = C_2 + 2C_1 \]
Worst case for two processes

For two processes
\[ C_1 + C_2 = T_1 \quad \text{and} \quad 2C_1 + C_2 = T_2 \]

Writing \( \lambda = \frac{T_2}{T_1} \)

We get
\[ U = \frac{C_1}{T_1} + \frac{C_2}{T_2} = \lambda + \frac{2}{\lambda} - 2 \]

Worst case is at \( \frac{dU}{d\lambda} = 0 \Rightarrow \lambda = 2^{1/2} \)

So utilisation worst-case bound is
\[ U \leq 2(2^{1/2} - 1) \]