§ 4 Real-time programming

4.1 Problem definition
4.2 Real-time programming methods
4.3 Computation Tasks
4.4 Synchronization of tasks
4.5 Communication between tasks
4.6 Scheduling methods
Chapter 4 - Learning targets

- to know what is meant by real-time programming
- to know the requirements of real-time programming
- to be able to differ between hard and soft real-time
- to understand what is meant by synchronous programming
- to understand the asynchronous programming
- to be able to explain what tasks are
- to know how time synchronization can be done
- to be able to use semaphores
- to know what scheduling methods are
- to be able to use the different scheduling methods
- to know what a schedulability test is
§ 4 Real-time programming

4.1 Problem definition
4.2 Real-time programming methods
4.3 Computation tasks
4.4 Synchronization of tasks
4.5 Communication between tasks
4.6 Scheduling methods
### What is real-time programming?

<table>
<thead>
<tr>
<th>Non-real-time computing</th>
<th>Real-time computing</th>
</tr>
</thead>
<tbody>
<tr>
<td>– Correctness of the result</td>
<td>– Correctness of the result</td>
</tr>
<tr>
<td></td>
<td>– Timeliness of the result</td>
</tr>
</tbody>
</table>

*Not too soon, not too late*
4.1 Problem definition

NON-REAL-TIME Computing:

Input data \rightarrow \text{Data Processing} \rightarrow \text{Output data}

REAL-TIME Computing:

\text{Time} \downarrow \text{Input data} \rightarrow \text{Data Processing} \rightarrow \text{Output data} \downarrow \text{Time}
Real-Time Programming

Creation of programs in such a way that the time requirements on the compilation of input data, on the processing and on the delivery of output data are fulfilled.

Time requirements are dependent on the processes in the technical system.

Real-time: according to the real-time sequences, no time expansion, no time compression

Different kinds of requirements on the time behavior of the data processing
  – Requirements for timeliness
  – Requirements for concurrency
4.1 Problem definition

Interaction technical process and automation system

Example:

malfunction due to too early/too late arriving sensor/actuator values
## Differences between information and real-time systems

<table>
<thead>
<tr>
<th>Information systems</th>
<th>Real-time systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>data-driven system</td>
<td>event/time driven system</td>
</tr>
<tr>
<td>complex data structure</td>
<td>simple data structure</td>
</tr>
<tr>
<td>great amount of input data</td>
<td>small amount of input data</td>
</tr>
<tr>
<td>I/O-intensive</td>
<td>computation-intensive</td>
</tr>
<tr>
<td>machine-independent</td>
<td>machine-dependent</td>
</tr>
</tbody>
</table>
4.1 Problem definition

**Important terms (1)**

**Reactive systems**

Real-time systems that react to the input signals of the technical process and deliver output signals to influence the technical process.

**Example: Automation systems**

**Embedded systems**

Integration of the automation system in the technical process physically and logically.

**Examples:** electric razor, mobile telephone, washing machine, power drill
Important terms (2)

**Hard real-time systems**

Strict deadlines must not be missed in any case

**Examples:** DDC-control in airplanes, motor control in automobiles

**Soft real-time system**

A violation of deadlines can be tolerated
4.1 Problem definition

Requirements for timeliness

- timely compilation of input data
- timely data processing
- timely delivery of output data

Time requirements regarding timeliness

- **Absolute time requirements**
  e.g.: 11:45 signal for departure

- **Relative time requirements**
  e.g.: Turn off signal 10 seconds after a measured value exceeds its threshold
4.1 Problem definition

Classification of time requirements (1)

Execution of a function at fixed times $t_1, t_2, t_3, t_4$

Execution of a function within a tolerance time range assigned to each time $t_n$
4.1 Problem definition

Classification of time requirements (2)

Execution of a function within an interval up to a latest time $t_1$

Execution of a function within an interval starting from an earliest time $t_1$
# Typical examples of applications with time requirements

<table>
<thead>
<tr>
<th>Absolute time requirements</th>
<th>Relative time requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution of a function at a fixed time</td>
<td>Recording of test values</td>
</tr>
<tr>
<td>Execution of a function within a tolerance time range</td>
<td>Recording of controlled variables</td>
</tr>
<tr>
<td>Execution of a function within an interval up to a latest time</td>
<td>Recording of data telegrams</td>
</tr>
<tr>
<td>Execution of a function within an interval starting from an earliest time</td>
<td>Sequence control in batch processes</td>
</tr>
</tbody>
</table>
4.1 Problem definition

Requirements for concurrency

Processes in the “real world” take place simultaneously

- Real-time systems have to react to that simultaneously
- Several computation tasks have to be executed simultaneously

Examples:
- Reaction to simultaneous trips of several trains
- Processing of several simultaneously occurring measurements in heating systems
- Control motor and ABS system simultaneously
4.1 Problem definition

Realization of concurrency

- Each computation task is processed on a separate computer
  real parallelism

- One computer for all data processing tasks
  quasi parallel

Prerequisites:

Processes in environment are slow in comparison to the computation of the programs on the computer
4.1 Problem definition

**Requirements for determinism**

Determination = *predictability* of the system behavior

Parallel and quasi-parallel sequences are generally not predictable; time shifts can lead to different execution sequences.

– the time behavior is not deterministic

– no guaranty of the safety of automation systems
4.1 Problem definition

**Deterministic real-time system**

A real-time system is called deterministic if for each possible state and for each set of input information there is a defined set of output information and a defined following state.

**Prerequisites:** a finite set of system states

**Timely deterministic system**

The response times for all output information are known.

In a deterministic system it is guaranteed that the system can react at every time. In a timely deterministic system it is additionally guaranteed at which time the reaction will have taken place.
4.1 Problem definition

Dialog systems

- Input data from input mediums
  - Keyboard
  - Light pen
  - Mouse
- Waiting for reply, i.e. output of results on an output medium
  - Screen
  - Printer

Examples of dialog systems
- Seat reservation systems of airline companies
- Account management in banks
- Storekeeping systems

Timeliness in dialog systems
Permitted response time within the range of seconds
4.1 Problem definition

**Automation systems**

Time reaction depends on the processes in the technical system

**Timeliness in automation systems**

Permitted response time within the range of milliseconds / microseconds

**Methods for real time programming are similar for**

– Automation systems
– Dialog systems
§ 4 Real-time programming

4.1 Problem definition

4.2 Real-time programming methods

4.3 Computation tasks

4.4 Synchronization of tasks

4.5 Communication between tasks

4.6 Scheduling methods
4.2 Real-time programming methods

Different methods

- **Synchronous programming:**
  Planning of the time sequence of the execution of programs
  
  *planned economy*

- **Asynchronous programming (parallel programming):**
  Organization of the time sequence during the execution of the programs.
  
  *market economy*
4.2 Real-time programming methods

e.g.: Dentist's office as real-time system (chalkboard writing)
4.2 Real-time programming methods

**Synchronous programming method**

**Synchronous programming:**
Planning of the time behavior of subprograms that have to be executed cyclically, before their actual execution

- Synchronization of the cyclical subprograms with a time grid
- Time grid generated by a real-time clock, interrupting signal for the call of subprograms
  *Time triggered*
- Strictly predefined sequence of the execution of subprograms
Example: heating system

- Controlled system 1: "heating circuit flat"
  - Input: $u_1(t)$
  - Output: $y_1(t)$
  - Room temperature flat

- Controlled system 2: "heating circuit office"
  - Input: $u_2(t)$
  - Output: $y_2(t)$
  - Room temperature office
  - Preheating temperature

- Controlled system 3: "boiler"
  - Input: $u_3(t)$
  - Output: $y_3(t)$

- Analog-output
- Analog-input
- Process-signal-in/output
- Real-time clock
- Automation computer
- Operating terminal

© 2004 IAS, Universität Stuttgart
4.2 Real-time programming methods

Technical (control engineering) conception

- Design and calculation of the control algorithms and control parameters
- Determination of the sampling time for the control circuits

\[
\begin{align*}
T & = \text{real-time clock - cycle time} \\
T_i & = \text{sampling time for control circuit } i \\
T_1 & = T \\
T_2 & = 2T \\
T_3 & = 5T
\end{align*}
\]
## 4.2 Real-time programming methods

### Assignment of the identifiers and sampling times to the subprograms (control programs)

A subprogram for each control circuit

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Identifier (name)</th>
<th>Sampling time (cycle time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperature control for subsystem “heating circuit: flat“</td>
<td>CONTROL 1</td>
<td>$T_1 = T$</td>
</tr>
<tr>
<td>Temperature control for subsystem “heating circuit: office“</td>
<td>CONTROL 2</td>
<td>$T_2 = 2T$</td>
</tr>
<tr>
<td>Temperature control for subsystem “heating circuit: boiler“</td>
<td>CONTROL 3</td>
<td>$T_3 = 5T$</td>
</tr>
</tbody>
</table>
Preliminary design with synchronous programming

Start

Interrupt signals from the real-time clock with the cycle time T

invoke all $T_1 = T$ CONTROL1

invoke all $T_2 = 2T$ CONTROL2

invoke all $T_3 = 5T$ CONTROL3

Wait loop
4.2 Real-time programming methods

Final design of the control program according to the synchronous programming method

Definition of the counter variables $Z_2$ and $Z_3$

In time intervals $T$ successively appearing interrupt signals cause start at this point

Initialize $Z_2$ and $Z_3$

CONTROL1

Z2 := 1
Z3 := 1

CONTROL2

Z2 := 1
Z2 := Z2 + 1

Z3 := Z3 + 1

CONTROL3

Z3 := 1
Z3 := Z3 + 1

Z3 := Z3 + 1

Z3 := Z3 + 1

Wait loop

© 2004 IAS, Universität Stuttgart
4.2 Real-time programming methods

**Time sequence of the synchronous programming method**

**Assumption:**
- Computation time for subprograms is identical
- Sum of computation times of the three subprograms is smaller than cycle time
Characteristics of the synchronous programming method (1)

- Requirement for timeliness is approximately fulfilled
  
- Requirements for concurrency is fulfilled, if cycle time $T$ is small compared to the time constants in the technical process

- Synchronous programming is good for real-time systems with cyclic program execution
  
- Synchronous programming is not suitable for the reaction on timely non-predictable (asynchronous) events
  
  - Increase of computation time through constant polling
  
  - Delay of the reaction
4.2 Real-time programming methods

Characteristics of the synchronous programming method (2)

- Normally deterministic behavior
- No complex organization program
- A bit more expensive in its planning (development)

Disadvantages of synchronous programming:

Modification of the requirements specification causes modification of the program structure

e.g.: PLC-programming
4.2 Real-time programming methods

**Asynchronous programming method (parallel programming)**

Organization program (real-time operating system) controls the timely execution of subprograms at run-time

- Execution of subprograms, when time requirements are fulfilled
- Simultaneous execution is sequentialized according to a certain strategy
  - Assignment of priority numbers
  - The priority is the higher, the lower the priority number is
4.2 Real-time programming methods

Example: heating system

synchronous (cyclic)

Controlled system 1

Controlled system 2

Controlled system 3

Analog-Output

Analog-Input

Digital Output

Digital Input

Process unit

Real-time clock

Automation computer system

Operating terminal

Printer

asynchronous
(stochastic)

Warning lamp

Burner failure signal

\[ u_1(t) \rightarrow \text{Controlled system 1} \rightarrow y_1(t) \]

\[ u_2(t) \rightarrow \text{Controlled system 2} \rightarrow y_2(t) \]

\[ u_3(t) \rightarrow \text{Controlled system 3} \rightarrow y_3(t) \]
### Assignment of indicators, sampling times and priority

<table>
<thead>
<tr>
<th>Subprogram</th>
<th>Name</th>
<th>Sampling time</th>
<th>Priority number</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reaction on burner failure with alarm message</td>
<td>ALARM</td>
<td>–</td>
<td>1</td>
<td>highest</td>
</tr>
<tr>
<td>Temperature control unit for heating circuit 1</td>
<td>CONTROL 1</td>
<td>$T_1 = T$</td>
<td>2</td>
<td>second highest</td>
</tr>
<tr>
<td>Temperature control unit for heating circuit 2</td>
<td>CONTROL 2</td>
<td>$T_2 = 2T$</td>
<td>3</td>
<td>third highest</td>
</tr>
<tr>
<td>Temperature control unit for heating circuit 1</td>
<td>CONTROL 3</td>
<td>$T_3 = 5T$</td>
<td>4</td>
<td>lowest</td>
</tr>
</tbody>
</table>
4.2 Real-time programming methods

Time sequence of the four subprograms

a) Desired course

CONTROL3 (Priority 4)
CONTROL2 (Priority 3)
CONTROL1 (Priority 2)
ALARM (Priority 1)

real-time clock

<table>
<thead>
<tr>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>T</td>
</tr>
</tbody>
</table>

b) Actual course

CONTROL3
CONTROL2
CONTROL1
ALARM Operating system

<table>
<thead>
<tr>
<th>0</th>
<th>T</th>
<th>2T</th>
<th>3T</th>
<th>4T</th>
<th>5T</th>
<th>6T</th>
<th>7T</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2(1)</td>
<td>T1(1)</td>
<td>T1(2)</td>
<td>T1(3)</td>
<td>T1(4)</td>
<td>T2(2)</td>
<td>T2(3)</td>
<td>T2(4)</td>
</tr>
</tbody>
</table>

Burner failure signals
Characteristics of the asynchronous programming method

- Requirements for timeliness only approximately fulfilled
  **Bad for low priority subprograms**

- Time requirements are fulfilled the better the higher the priority of the according subprogram

- Actual time sequence can shift away from the desired time sequence
  **Subprograms can pass each other mutually**

- The succession of the subprograms is not deterministic, but occurs dynamically

- At the program development it cannot be indicated in advance which of the subprograms will run at which point of time

  - **simple development**
  - **complexity in the administration program**
  - **program sequence is not transparent**
### Event-driven architectures

- all activities as sequences of events
  - activation of tasks
  - transmission of messages
- support through real-time operating systems
- non-deterministic behavior
- flexible regarding modifications

### Time-driven architectures

- periodical execution of all tasks and communication actions
- sampling of external variable at determined times
- low flexibility in case of modifications
- easy to analyze

**PLC-systems are time-driven real-time systems**
## § 4 Real-time programming

4.1 Problem definition
4.2 Real-time programming methods
4.3 **Computation tasks**
4.4 Synchronization of tasks
4.5 Communication between tasks
4.6 Scheduling methods
4.3 Computation tasks

Distinction

- Program (sequence of commands)
- Execution of the program (a single execution of command sequence)
- Task

Invocation of subprograms

- Execution of the invoking program is interrupted
- Execution of the subprogram
- Continuation of the invoking program

Invocation of a task

- Simultaneous execution of the invoking program and of the invoked task
4.3 Computation tasks

**Task**

a procedure of the execution of a sequential program controlled by a real time operating system

**Introduction of the term computation task**

- Task starts with entry in a list of the real-time operating system and ends with the deletion from that list
- Task does not exist only during the execution of the commands, but also during planned and forced waiting times
### Differences between task and thread

<table>
<thead>
<tr>
<th>Task</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner of resources</td>
<td>Cannot own resources besides the processor’s, accesses all resources of the task, to which it belongs</td>
</tr>
<tr>
<td>Own address space</td>
<td>Address space of the task, to which it belongs</td>
</tr>
<tr>
<td>Contains one or several threads</td>
<td>Element of a task</td>
</tr>
<tr>
<td>Communication beyond the task boundaries, preferred via messages</td>
<td>Communication between the threads, preferred via shared data</td>
</tr>
</tbody>
</table>
4 basic states

- State “running”
  - the subprogram is processed

- State “runnable” or “ready”
  - all time conditions for the process are fulfilled
  - what is missing is the start from the operating system
    i.e. the execution

- State “suspended”
  - task is waiting for the occurrence of an event
  - as soon as event occurs transition from state “suspended” to “ready”

- State “dormant”
  - task is not ready because time conditions or other conditions are not fulfilled
State diagram of a task

Planning of tasks
- Invocation of a task cyclically or at certain times
- “Planning” is the transition from the state “dormant” into the state “runnable”
Course of the task “Control 3” in the asynchronous programming method

The task Control 3 is:

- running (active)
- blocked
- runnable (ready)
- dormant
Assignment of priorities for tasks

- static priority assignment
- dynamic priority assignment (use of deadlines)

Time parameters of a task:

A: Arrival time
R: Request time
S: Start time
C: Completion time
D: Deadline
E: Execution time
P: Period time
L: Laxity
r: Remaining flow time
f: Flow time
4.3 Computation tasks

Appearance of the time parameters of a task

- **Execution time**: $E(t)$
- **Remaining flow time**: $r(t)$
- **Laxity**: $L$
- **Flow time**: $f(t)$
- **Period time**: $P$

**States of a task**:
- "running"
- "suspended"
- "ready"
- "dormant"

**Time of viewing**: $t$

Diagram showing the appearance of the time parameters of a task with various states and their corresponding periods.
Correlation between the time parameters of a task

\[ A \leq R \leq S \leq C \quad E \leq D \quad E \]

**Execution time (computing time)**

\[ E(t) = E_{\text{old}}(t) + E_{\text{new}}(t) \]

- \( E_{\text{old}}(t) \): present execution time at time of viewing
- \( E_{\text{new}}(t) \): remaining execution time

**Laxity**

for the execution of a task

\[ L = D - S - E \]
§ 4  Real-time programming

<table>
<thead>
<tr>
<th>4.1</th>
<th>Problem definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>Real-time programming methods</td>
</tr>
<tr>
<td>4.3</td>
<td>Computation tasks</td>
</tr>
<tr>
<td><strong>4.4</strong></td>
<td>Synchronization of tasks</td>
</tr>
<tr>
<td>4.5</td>
<td>Communication between tasks</td>
</tr>
<tr>
<td>4.6</td>
<td>Scheduling methods</td>
</tr>
</tbody>
</table>
4.4 Synchronization of tasks

Classification of the actions of a task

- Two actions within a task are called **parallel** if they can run simultaneously.
- Two actions are called **sequential** if they are arranged in a certain sequential order.
- Two actions of two different tasks are called **concurrent** if they can run simultaneously (outer parallelism).
- Two actions of a task are called **simultaneous**, if they can be run at the same time.

**Actions = Threads**
4.4 Synchronization of tasks

**Requirement on the execution of tasks**

Synchronicity between tasks and the technical process

**Dependencies between tasks**

- Logical dependency because of the technical process
  
  e.g.: The desired values for the control have to be defined at least once before they can be used

- Dependencies through the use of shared resources
4.4 Synchronization of tasks

Example

the dependencies of tasks due to shared resources

Shared resources:
– protocol printer
– analog-input

possibility of deadlock!
Problems of dependency

Deadlock:

Two or more tasks block themselves mutually

Permanent deadlock (livelock, starvation)

A conspiracy of tasks block a task

\[ \text{Time coordination of the tasks} = \text{synchronization of tasks} = \text{limitation of the free parallel execution} \]

The synchronization of tasks is equivalent to the synchronization of their actions.
Example for a deadlock

Road crossing without traffic lights. Traffic rule is “right before left“.

[Diagram of a road crossing with vehicles waiting at intersections]
Example for a livelock: “The Dining Philosophers”

- every philosopher needs 2 chopsticks to eat

- no chopstick can be reserved
Main forms of synchronization

Logical synchronization
(task oriented or process oriented synchronization)

- Adaptation of the sequence of actions of a task to the sequence of operations in the technical process.
- Synchronization means
  - Fulfillment of requirements regarding the sequence of actions
  - Consideration of given times resp. intervals
  - Reaction to the interrupt message from the technical process

Resource-oriented synchronization

- Fulfillment of requirements regarding the use of shared resources
Synchronization of tasks

Synchronization methods

– Semaphore
– Critical regions
– Rendezvous concept

Basic idea of all synchronization methods

– Task has to wait until a certain signal or event occurs
– Use of waiting conditions at critical places
**Semaphore concept**

Synchronization of tasks through signals (Dijkstra)

Semaphore variable: positive, integer value

Semaphore operations: V(S) und P(S)

\[ V(S_i) : \] Operation V (S_i) increases the value of the semaphore variable S_i by 1

\[ P(S_i) : \] Operation P(S_i) determines the value of S_i

- if value of S_i > 0
  - decrease S_i by 1

- if S_i = 0 it has to be waited, until S_i > 0

indivisible!
Example 1: Railroad traffic with single-track route

Direction of the trains

Operation

Variable

Synchronization

Position

Operation
Example 2: Railroad traffic with single-track route

Track magnet
Train A
Track magnet
Train B
Track magnet
Train A
Track magnet
Train B

P(S1)
Signal S1
V(S2)
V(S1)

P(S2)
Signal S2

Direction of the trains

with ordered sequence

© 2004 IAS, Universität Stuttgart
4.4 Synchronization of tasks

**Example:**

operations on two tasks, that always run alternately

logical synchronization
§ 4 Real-time programming

4.1 Problem definition
4.2 Real-time programming methods
4.3 Computation tasks
4.4 Synchronization of tasks
4.5 Communication between tasks
4.6 Scheduling methods
4.5 Communication between tasks

**Definition**

Synchronization =
Fulfillment of time related and logical conditions in the parallel run of tasks

Communication =
Exchange of data between parallel running tasks

**Interrelation: Synchronization - Communication**

Synchronization : Communication without information
Communication : Synchronization for information exchange
Possibilities of data communication

- Shared memory (commonly used memory)
  - common variable
  - common complex data structure

- Sending of messages
  - transmission of messages from a task to another (message passing)
  - especially in distributed systems
4.5 Communication between tasks

Different kinds of communication

- **Synchronous communication**
  - Sending and receiving processes communicate at a certain predefined position in the program flow

  waiting through blocking

- **Asynchronous communication**
  - Data are buffered

  no waiting times
§ 4 Real-time programming

4.1 Problem definition
4.2 Real-time programming methods
4.3 Computation tasks
4.4 Synchronization of tasks
4.5 Communication between tasks
4.6 Scheduling methods
Scheduling problem

- Tasks need resources
  (processor, in/output device etc.)
- Number of resources is limited
- Tasks compete for resources
- The allocation of resources has to be managed

Example: Railroad tracks in station area
4.6 Scheduling methods

**Scheduling:**

Allocation of the processor to runnable tasks according to a predefined algorithm (Scheduling-method)

**Problem:**

1. Is there an executable schedule for a set of tasks?
2. Is there an algorithm that can find an executable schedule?

**Example:** Lecture-room-time-allocation
4.6 Scheduling methods

Classification dependent on the time of the planning

- **Static scheduling**
  - Planning of the execution sequence of the tasks is done before the actual execution (dispatching table)
  - Consideration of information on taskset, deadlines, execution times, sequential relations, resources
  - Dispatcher carries out the allocation according to dispatching table
  - Runtime - overhead minimal
  - Deterministic behavior

- **Dynamic scheduling**
  - Organization of the execution sequence during the execution of the tasks
  - Considerable runtime - overhead
4.6 Scheduling methods

Classification according to the kind of execution

- **Preemptive scheduling**
  - running task can be interrupted
  - higher priority tasks replace lower priority tasks

- **Non-preemptive scheduling**
  - running task cannot be interrupted
  - processor-deallocation by the task itself
Scheduling methods

- FIFO scheduling (first-in-first-out)
- Scheduling with fixed (invariable) priorities
- Round-Robin-Method (Time slice method)
- Earliest-deadline-first method
- Rate monotonic scheduling
- Method of minimal laxity (least laxity)
**FIFO scheduling**

- Non-preemptive scheduling
- The processor is allocated to the task with the longest delayed planning
- Simple implementation

*unsuitable for hard real time systems*

The tasks are executed in the order in which they become runnable

![Diagram of tasks A, B, C, D over time t]
4.6 Scheduling methods

**Round-Robin-Method (time slice method)**

- Each task has a determined time slot in which the processor is allocated to it
- Sequence is determined statically
- Execution of a task “step by step”
- Used in dialog systems (multi-tasking systems)

unsuitable for hard real-time systems
4.6 Scheduling methods

**Example: Round-Robin method**

Each time slice has 10ms and the tasks were arranged in the following order: A-B-C-D.

Execution time of the tasks:

- Task A: 25ms
- Task B: 20ms
- Task C: 30ms
- Task D: 20ms
Scheduling with fixed priorities

- Priorities are assigned statically
- The processor is allocated to the task with the highest priority
- Preemptive and non-preemptive strategy is possible
- Simple implementation

for hard real-time systems only suitable under certain conditions
Example: fixed priorities

<table>
<thead>
<tr>
<th>Task</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
</tr>
</tbody>
</table>

Depending on the strategy one of the two tasks A or B is running as long as it has the highest priority.

* FIFO scheduling: A was the first runnable task
4.6 Scheduling methods

Rate monotonic scheduling

- Special case of scheduling with fixed priorities of cyclic tasks
- The shorter the period, the higher the priority
- Task with the shortest period has the higher priority
- Preemptive strategy

Method frequently used in real applications
Example: Rate-Monotonic scheduling

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution time</th>
<th>Period</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10 ms</td>
<td>40 ms</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>20 ms</td>
<td>50 ms</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>10 ms</td>
<td>80 ms</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>20 ms</td>
<td>100 ms</td>
<td>4</td>
</tr>
</tbody>
</table>

First scheduled call of all tasks at $t = 0$ ms. Afterwards the tasks have to be repeated cyclically.

Task D was interrupted.
4.6 Scheduling methods

**Earliest-Deadline-First-method**
*(Minimal-remaining-flow-time-method)*

- The processor is allocated to the task with the shortest remaining flow time
- Preemptive method
- High computation effort for the scheduling
- Compliance with requirements on time is specially supported
4.6 Scheduling methods

Example

<table>
<thead>
<tr>
<th>Task</th>
<th>Time</th>
<th>$T_{\text{min}}$</th>
<th>$T_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10 ms</td>
<td>0 ms</td>
<td>40 ms</td>
</tr>
<tr>
<td>B</td>
<td>10 ms</td>
<td>0 ms</td>
<td>30 ms</td>
</tr>
<tr>
<td>C</td>
<td>30 ms</td>
<td>30 ms</td>
<td>100 ms</td>
</tr>
<tr>
<td>D</td>
<td>40 ms</td>
<td>50 ms</td>
<td>200 ms</td>
</tr>
<tr>
<td>E</td>
<td>10 ms</td>
<td>70 ms</td>
<td>90 ms</td>
</tr>
</tbody>
</table>

$t_{\text{min}}$: earliest time  
$t_{\text{max}}$: latest time  
= deadline

Execution sequence:

Reason: Deadline of B is earlier as the one of A  
Deadline of E is earlier as the one of D  
Preemption
Least laxity method

- Processor is allocated to the task with the smallest laxity
- Consideration of deadlines and of the execution time
- Very expensive method

best suitable for hard real-time systems
Schedulability test

A schedulability test is a mechanism to prove whenever a set of tasks can be scheduled in such a manner that the deadlines are not missed.

Being a mathematical proof, it is important to differentiate between:

- necessary condition
- necessary and sufficient condition
Theorem of Liu and Layland (Part I)

Any given set of tasks (Deadline = Period) can be called by the preemptive Earliest-Deadline-First method when:

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

Where:
- \( C_i \): Execution time of task \( i \)
- \( T_i \): Period of task \( i \)
- \( n \): Number of tasks

Concerning the EDF method this test is necessary and sufficient.

For any scheduling method this is only a necessary condition.
4.6 Scheduling methods

Theorem of Liu und Layland (Part II)

Any given set of tasks (Deadline = Period) can be called by the Rate-Monotonic method when:

\[ \sum_{i=1}^{n} \frac{C_i}{T_i} \leq U_n^* = n \left( \frac{1}{2^n} - 1 \right), \quad n = 1, 2, \ldots \]

Where:
- \( C_i \): Execution time of task \( i \)
- \( T_i \): Period of task \( i \)
- \( U_n^* \): Utilisation of processor with \( n \) tasks
- \( n \): Number of tasks

and:

\[ U_i = \frac{C_i}{T_i} \]

Sufficient test
Question referring to Chapter 4.1

Which kind of real-time systems are the following ones?

Answer

<table>
<thead>
<tr>
<th></th>
<th>hard real-time system</th>
<th>soft real-time system</th>
</tr>
</thead>
<tbody>
<tr>
<td>paper printing device</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>car electrical window control</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>TV electron beam control</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>telephone switching control</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CNC milling head control</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>aircraft turbine engine control</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

If a deadline is missed in a **hard real-time system**, then this is equal to a failure of the automation system. Often damage to property or to persons can occur in such systems.
Question referring to Chapter 4.2

Which of the following statements regarding to real-time programming do you agree?

Answer

- In real-time programming only the timeline of an event is in center.
- Real-time means „as fast as possible“.
- In soft real-time systems timelines do not have to be fulfilled.
- The asynchronous programming method is more flexible regarding to outer events than a synchronous programming method.
- A synchronous programming requires cyclic programm flow.
- The synchronous programming method does not fulfill the requirement for concurrency.
Question referring to Chapter 4.4

a) Two tasks access the same temperature sensor in an automation system.

b) A control algorithm is divided into three tasks: „input of actual value“, „calculation of control variable“ and „output of control variable“. These tasks always have to be executed in this sequence. Which type of synchronization has to be used respectively? How many semaphore variables are necessary in each case?

Answer

a) This is a resource-oriented synchronization. One semaphore variable is required for each resource. (In this example: one)

b) The second example is a logical synchronization. One semaphore variable is required for each point of synchronization at a transition from one task to the next one. (Three in this example)
Question referring to Chapter 4.6

In the so called „shortest-job-first“-scheduling method, the task with the shortest execution time is selected for processing at run-time. However, running tasks cannot be interrupted by other tasks.

Which type of scheduling method is this?

Answer

The planning of the task sequence is done only at run-time of the system, during the program execution. Furthermore, the tasks cannot be interrupted.

Consequently, we can classify this method as a dynamic, non-preemptive scheduling method.

Remark:

The “shortest-job-first” method determines a schedule, that minimizes the average flow-time.
Crosswords to Chapter 4
Crosswords to Chapter 4

Across
1. Term for activities that are ordered in a specific order. (10)
3. Situation in which a task can not realize its duty, even if it continues to run. (8)
5. Procedure of the execution of a sequential program, controlled by a real time operating system. (4)
7. Transient from the state "dormant" into the state "Runnable" (8)
9. Synchronization construct (9)
11. A stalemate that occurs when two (or more) tasks are each waiting resources held by each other. (8)
13. Allocation of processor resources to tasks ready to execute. (10)

Down
2. Organization of timely execution of subprograms at runtime. (12)
4. Latest execution time point of a task. (8)
6. Transition from the state "dormant" into the state "runnable" (8)
8. Allocating processor resources to tasks ready to execute. (10)