§ 5 Real-time operating systems

5.1 Definition
5.2 Organization tasks of a real-time operating system
5.3 Development of a mini-real-time operating system
5.4 Software system design of the mini real-time operating system
5.5 Examples for real-time operating systems
Chapter 5: Real-Time Operating Systems

Chapter 5 - Learning targets

- to know what an operating system is
- to be able to explain what is meant by resources
- to know the functions of an operating system
- to know what interrupts are
- to be able to explain how the memory management is working
- to know the development process of a mini-real-time operating system
- to understand the composition of a mini-real-time operating systems
- to know how the mini operating system is working
- to understand the extensions of the mini operating system
- to understand how the mini operating systems is working
- to get an overview of real-time operating systems
§ 5 Real-time operating systems

5.1 Definition

5.2 Organization tasks of a real-time operating system

5.3 Development of a mini-real-time operating system

5.4 Software system design of the mini real-time operating system

5.5 Examples for real-time operating systems
What is an operating system?

**Definition DIN 44300:**
Operating systems are programs of a digital computer system that together with the characteristics of the computer hardware form the basis of the possible operating modes of the digital computer system and especially control and supervise the handling of programs.

**Operating system**

– Systematically built up collection of control programs and tools
– Allocation of the existing resources to the competing computation processes **Scheduling**
– Simplification of the operating and programming of the computer and its attached devices for the user **Driver**
Characteristic of operating systems

- Realization of the hardware-dependent tasks

- In many cases enclosing from the producer of the computer
  - efficient operating system requires exact knowledge on the hardware structure
  - often for entire computer lines
  - amortization of the high development costs of an operating system

- Size
  - several kilo-bytes in micro computer
  - several mega-bytes in mainframe computer

- Integration of classical operating systems modules in form of semiconductor chips
Resources

– Objects necessary for the execution of the computer process and for which allocation the computer process has to wait

– Device units
  • Processors
  • Memory
  • Peripheral devices like printers

– System programs
Categories of real-time operating systems (1)

- **Real-time - UNIX**
  - Compatible to UNIX-System V
  - Used in process control systems

- **Real-time - kernels**
  - UNIX-compatible micro-kernel with
    Memory management,
    Interrupt handling,
    Scheduler,
    Task management,
    Interfaces on basis of TCP/IP
  - Optimally adapted to requirements
  - Well optimized code for different platforms
Categories of real-time operating systems (2)

– Real-time operating system extensions
  • Extension of MS-DOS-systems
  • Library for the compliance with real time conditions

– Real-time operating systems
  • Very efficient
  • Flexibly configurable
  • Oriented on UNIX
### § 5 Real-time operating systems

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Definition</td>
</tr>
<tr>
<td>5.2</td>
<td><strong>Organization tasks of a real-time operating system</strong></td>
</tr>
<tr>
<td>5.3</td>
<td>Development of a mini-real-time operating system</td>
</tr>
<tr>
<td>5.4</td>
<td>Software system design of the mini real-time operating system</td>
</tr>
<tr>
<td>5.5</td>
<td>Examples for real-time operating systems</td>
</tr>
</tbody>
</table>
5.2 Organization tasks of a real-time operating system

**Tasks of a real-time operating system**

Management of computer processes and resources in compliance with the requirements for timeliness, concurrency and efficiency

**Functions of the operating system**

- Organization of the execution of the computation processes (Scheduling)
- Organization of the interrupt handling
- Organization of the memory management
- Organization of the input/output
- Organization of the process in case of irregular operating states and start-up/restart
Layer-architecture of an automation computer system

Automation Computer

- Computation processes and resources
- Real-time operating system
  - operating system kernel (micro kernel)
  - configurable extension
- Computer hardware

© 2004 IAS, Universität Stuttgart
### Automation Computer System

#### Automation programs

<table>
<thead>
<tr>
<th>computation process 1</th>
<th>computation process 2</th>
<th>\cdots</th>
<th>computation process n</th>
</tr>
</thead>
</table>

#### Resources

<table>
<thead>
<tr>
<th>resources 1</th>
<th>resources 2</th>
<th>\cdots</th>
<th>resources n</th>
</tr>
</thead>
</table>

#### Real-time operating system

Management of computer processes:
- register,
- instruct,
- planning,
- terminate,
- etc.

Management of resources:
- announce,
- create,
- request,
- occupy,
- unlock,
- etc.
5.2 Organization tasks of a real-time operating system

Computation process management

Different kinds of computation processes
- Application processes
- System processes

Tasks of the computation process management
- Coordination of the execution of application and system processes
- Parallel operation of as many resources as possible
- Work of queues for resources
- Synchronization of application system processes
- Avoidance, identification and elimination of deadlocks

© 2004 IAS, Universität Stuttgart
Interrupt handling

- Interruption of the planned program sequence
- Start of a service routine
5.2 Organization tasks of a real-time operating system

**Interrupt handling**

- Creation and processing of vectorized interrupts
- Start of an interrupt service routine while simultaneously interrupting the presently running computation process
- Prioritization of interrupts
- Hardware functions for the interrupt handling (within the range of micro-seconds)
Why memory management?

The cost of memory space is proportional to the access speed

→ optimal usage necessary

Memory hierarchy levels

- Cache memory (extremely fast semiconductor memory)
- Working memory
- Hard disc memory
- Floppy disc

Tasks of the memory management

- Optimal usage of the “fast” memories
- Coordination of the access on a shared memory area
- Protection of the memory area of different computer processes against false accesses
- Assignment of physical memory addresses for the logical names in application programs
## Input/output control

**Different kinds of input/output devices**

- Distinction in speed
- Distinction in data formats

## Realization of the input/output control

**Interface hardware-dependent/ hardware-independent**

- Hardware-independent level for the data management and the data transport
- Hardware-dependent level, that takes into account all device specific characteristics (driver programs)
Classification of errors (1)

– Faulty user inputs
  • Non-valid inputs have to be rejected with error message

– Faulty application programs
  • Guaranty, that a faulty application program does not affect other programs
Classification of errors (2)

– Hardware faults and hardware failures
  • Recognition of hardware faults and failures
  • Reconfiguration without the faulty parts
  • Shut-down sequences in case of power failures

– Deadlocks based on dynamic constellations
  • Reliable avoidance of deadlocks is not possible

Identification of deadlocks and elimination through withdrawal of operating resources
§ 5 Real-time operating systems

5.1 Definition
5.2 Organization tasks of a real-time operating system
5.3 Development of a mini-real-time operating system
5.4 Software system design of the mini real-time operating system
5.5 Examples for real-time operating systems
5.3 Development of a mini-real-time operating system

**Objective**

- Presentation of the structure and the mode of operation of a real-time operating system in strongly simplified form
- Gradual withdraw of the taken simplifications

**Development process**

- Clarification of the problem formulation, determination of the requirements
- Technical solution concept
- Software system design
- Implementation
Procedure of the development of the mini-real-time operating system

- **clarification and determination of requirements**
- **technical solution concept**
- **preliminary design of the program system**
- **final design of the program system**
- **implementation**
- **requirement specification**
- **functional specification**
- **block charts**
- **flow charts**
- **programs in a programming language**

Software System Design
Clarification and determination of the problem formulation and the requirements

– Management of a maximum of \( n \) computer processes
  
  \[ n = m + k \]

– One processor for the execution of operations
  
  • No optimization of the processes through simultaneous execution of computation operations and I/O operations

– Time signal for cyclic activities through internal clock generator
  
  • Clock impulses at fixed intervals \( T \) (i.e.: \( T = 20 \text{ms} \))
  • Different cycle times for the cyclic processes
### Simplifications that are withdrawn at a later point

| Å | The sum of all computing times of the computation processes is smaller that the interval T  
   | • Assurance that all computation processes are finished at the next clock pulse. |
| Ç | No tasks that are started by interrupt |
| ë | No resource management  
   | • Input/output times are negligibly short |
Design of a solution concept

Asynchronous programming method

– Asynchronous instruction of the individual computer processes
– No fixed sequence of tasks
– Conflict strategy according to priority numbers

Sub-solution

– Creation of cycle time
  • Derivation of the different cyclic times of the tasks from the clock impulse
– State management of the computer processes
  • Instruction of the tasks at the corresponding cyclic times and defined termination
– Start of the computation processes
  • Start of the task, which turn it is
Solution concept of the mini operating system

Sub-solution 1: cycle time formation

creation of the cycle times $T_1, T_2, ..., T_m$

Sub-solution 2: state management of the computation processes

instruction of the computer processes at the times $T_1', ..., T_m'$ and ending of the instruction

Sub-solution 3: start of the computation processes

start of the instructed computer processes according to priorities
Creation of cycle time

Task:

Formation of the cohesion between the cycle times $T_i$ ($i = 1, 2, ..., m$) and the interval $T$

Assumption: $T_i >> T$

$\Rightarrow T_i = a_i \cdot T$

$a_i$ Integral cycle time factors ($i=1, 2, ..., m$)

Interval variable $Z_i$ ($i=1, 2, ..., m$)

Arrival of the clock impulse reduces $Z_i$ by 1

$Z_i = 0$: Cyclic time $T_i$ is over

Reset of $Z_i$ on initial value $a_i$
### 5.3 Development of a mini-real-time operating system

#### Sub-solution 1: creation of cycle time

**Introduction of dimensionless, integral cycle time factors**

<table>
<thead>
<tr>
<th>$T_i = a_i T$</th>
<th>$a_i = \frac{T_m}{T_i}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1 = a_1 T$</td>
<td>$a_1 = \frac{T_m}{T_1}$</td>
</tr>
<tr>
<td>$T_2 = a_2 T$</td>
<td>$a_2 = \frac{T_m}{T_2}$</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$T_m = a_m T$</td>
<td>$a_m = \frac{T_m}{T_m}$</td>
</tr>
</tbody>
</table>

**Definition of (dimensionless) interval variables** $z_1, z_2, \ldots, z_m$ with the initial values

| $z_1 = a_1$ |
| $z_2 = a_2$ |
| $\ldots$ |
| $z_m = a_m$ |

At each arrival of clock impulse in interval $T$: decremention of the variable $z_i$ (decrement by 1), i.e. formation of:

| $z_1 := z_1 - 1$ |
| $z_2 := z_2 - 1$ |
| $\ldots$ |
| $z_m := z_m - 1$ |
as soon as a variable turns $z_i = 0$ the corresponding cycle time $T_i$ is reached. Therefore this result is transferred to the solution component 2, that is in charge of instructing the process $i$ (putting the state into "ready").

Reset of the corresponding interval variables to the initial value $z_i := a_i$. Continuation with solution component 2.
State management of the computation processes

Task:
– Management of the states of the computation processes
  • dormant
  • ready
  • blocked
  • running
– Bookkeeping on the respective states of each process
– Execution of state transitions
Sub-solution 2: State management of the computation processes

- In case of arrival of a message that the cycle time $T_i$ is over: Instruction of the computer processes in question (putting into the state "ready").
- Determine the computer process in state "ready" with the highest priority from the list.
- List of the current states of the computer processes.
- Message to solution component 3 (start of the concerning task).
- After the execution of a computer process: Write down the state "dormant" in the state list.
- Start message to subsolution 3.
- End message from subsolution 3.

From subsolution 1, if all interval variables are processed:
- Back to subsolution 1.

From subsolution 1:
- To subsolution 1.
Start of the computation processes

Task:

- Determination of the start address
- Starting of the computation processes
- Supervise the termination of the computer process
Sub-solution 3: Starting of the computation processes

- After the notification from sub-solution 2: Determine start address of the computation process in question
- Starting of the computation process in question (sub-program - jump to start address)
- Notification of sub-component 2 that the computer process is finished.

Start message from sub-solution 2

End message to sub-solution 2
§ 5 Real-time operating systems

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Definition</td>
</tr>
<tr>
<td>5.2</td>
<td>Organization tasks of a real-time operating system</td>
</tr>
<tr>
<td>5.3</td>
<td>Development of a mini-real-time operating system</td>
</tr>
<tr>
<td>5.4</td>
<td><strong>Software system design of the mini real-time operating system</strong></td>
</tr>
<tr>
<td>5.5</td>
<td>Examples for real-time operating systems</td>
</tr>
</tbody>
</table>
5.4 Software system design of the mini real-time operating system

Software system design based on strongly simplified formulation

Principle of the stepwise refinement

Dissection of the mini operating system in program routines, that are individually refined as well.
Dissection of the mini-real-time operating system program

- Sub-program TIME ADMINISTRATION
  - for the formation of the different cyclic times

- Sub-program TASK ADMINISTRATION
  - for the administration of the computer processes

- Sub-program PROCESSOR ADMINISTRATION
  - allocation of the resource “processor”
  - starting of the computer processes
Interaction of the sub-programs of the mini-real-time operating system

MINI-REAL-TIME OPERATING SYSTEM

Symbols

Kick-off (control flow)

Data transfer (data flow)

Program resp. program routine

List of data, e.g. single values (data)
Lists necessary for the TIME ADMINISTRATION (1)

- Time span variable $Z_i$ for cyclic time formation
  List TIME COUNTER

Filing of the time-span variables $z_i$

```
<table>
<thead>
<tr>
<th>$z_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_2$</td>
</tr>
<tr>
<td>$z_3$</td>
</tr>
<tr>
<td>$z_m$</td>
</tr>
</tbody>
</table>
```

List of variables
Lists necessary for the TIME ADMINISTRATION (2)

- Cyclic time $T_i$ for each computer process
  List CYCLE

Provision of the cyclic time factors $a_i$:

$$a_1 \quad a_2 \quad a_3 \quad \ldots \quad a_m$$

$m$ spaces

$= \frac{T_i}{T}$

List of constants
Block chart of the sub-program TIME ADMINISTRATION
List necessary for the TASK ADMINISTRATION and PROCESSOR ADMINISTRATION

- Status and base address of the computer process list

ADMINISTRATION BLOCK

Structure of the list for the administration of computer processes

ADMINISTRATION BLOCK

<table>
<thead>
<tr>
<th>B_1</th>
<th>Start address 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B_2</td>
<td>Start address 2</td>
</tr>
<tr>
<td>B_3</td>
<td>Start address 3</td>
</tr>
<tr>
<td>B_4</td>
<td>Start address 4</td>
</tr>
<tr>
<td>B_m</td>
<td>Start address m</td>
</tr>
</tbody>
</table>

- Status bits B_i
  - B_i = 0: ready
  - B_i = 1: dormant

- Base address of the code assigned to the computer process i (i = 1, 2, ..., m)

Two-dimensional field
Division of the sub-program TASK ADMINISTRATION

- ACTIVATION
- SEARCH
- DEACTIVATION
Block chart of the TASK ADMINISTRATION

- **Activation**
  - Selection of the start address by the PROCESSOR ADMINISTRATION
  - End of the computer process in question from/to TIME ADMINISTRATION

- **Search**
  - to the PROCESSOR ADMINISTRATION

- **Administration Block**

- **Deactivation**
  - from the PROCESSOR ADMINISTRATION
### Activation:
Modification of the status bits into “ready”

### Deactivation:
Modification of the status bits into “dormant”

### Search:
Check if there’s a task in state “ready”

Organizing the list ADMINISTRATION BLOCK allows a simple prioritization. Subdivision of PROCESSOR ADMINISTRATION is not necessary.
Overall block chart of the mini real time operating system
Fine design in form of a flow chart

Flow chart TIME ADMINISTRATION

CLOCK IMPULSE

TIME ADMINISTRATION

I = 1

TIME COUNTER (I) = TIME COUNTER (I) - 1

TIME COUNTER (I) = 0?

yes

TIME COUNTER (I) = CYCLE (I)

no

I = M?

yes

ACTIVATION

no

I = I + 1

search
Flow chart of the programs TASK ADMINISTRATION

ACTIVATION

- **ACTIVATION (I)**
- Address part from ADMINISTRATION BLOCK (I) = 0?
  - yes: Inquiry if computer process is available at all
  - no: Set STATUS BIT (I) IN ADMINISTRATION BLOCK (I) = 0
- ERROR MESSAGE
- Reentry

= ready
Search

- SEARCH
  - $I = 1$
    - STATUS BIT ($I$) from ADMINISTRATION BLOCK ($I$) = 0?
      - yes
        - $I = M$?
          - yes
            - Set STATUS BIT ($I$) in ADMINISTRATION BLOCK ($I$) = 1
          - no
            - $I = I + 1$
        - no
          - PROCESSOR ADMINISTRATION ($I$)
  - no

Deactivation

- DEACTIVATION ($I$)
  - Set STATUS BIT ($I$) in ADMINISTRATION BLOCK ($I$) = 1
  - SEARCH = dormant
Flow chart of the PROCESSOR ADMINISTRATION

- PROCESSOR ADMINISTRATION (I)
  - Start computer process over start address in ADMINISTRATION BLOCK (I), i.e. invoke as sub-program
  - DEACTIVATION (I)
### First extension of the system design

<table>
<thead>
<tr>
<th>Admission of longer computing times for the computer processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>In case of the arrival of a clock impulse it might be necessary to interrupt a (still) running computer process featuring a longer execution time and lower priority, in order to start a computer process with a higher priority.</td>
</tr>
<tr>
<td>Program for the interruption administration</td>
</tr>
</tbody>
</table>
Sub-program INTERRUPTION ADMINISTRATION (Administration program)

Task:
Rescue of the registers of the processor of a still running computer process.

- Program counter
- Accumulator
- Status register
- Working register
5.4 Software system design of the mini real-time operating system

Extended hierarchy chart after the admission of longer computing times in computer processes

- MINI-REAL-TIME OPERATING SYSTEM (Extension1)
- INTERRUPT ADMINISTRATION
- TIME ADMINISTRATION
- TASK ADMINISTRATION
- PROCESSOR ADMINISTRATION
5.4 Software system design of the mini real-time operating system

Extension of the list
ADMINISTRATION BLOCK

- Start address after an interruption
- Register memory location

<table>
<thead>
<tr>
<th>Task</th>
<th>Base address</th>
<th>Start address</th>
<th>Register contents 1</th>
<th>Register contents 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>B_1</td>
<td></td>
<td>(program counter contents)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B_2</td>
<td></td>
<td>(program counter contents)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B_m</td>
<td></td>
<td>(program counter contents)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2004 IAS, Universität Stuttgart
Extension of the sub-program PROCESSOR ADMINISTRATION

- Right before the start of a ready computer process the register with the contents of the list of the ADMINISTRATION BLOCK has to be loaded.

Extension of the sub-program DEACTIVATION

- After the computer process is finished its base address is loaded into the cell START ADDRESS and the register contents in the ADMINISTRATION BLOCK are to be initialized.
Overall chart of the mini real time operating system

First extension: Admission of longer computing times for computer processes
Second extension of the software system design

Having the possibility of alarm interrupts in mind

- Up to k computer processes, which activation is triggered by alarm interrupts that are not predictable from the point of view of time.

Extension of the INTERRUPTION ADMINISTRATION

- register rescue
- in case of clock impulse interrupts triggering of TIME ADMINISTRATION
- in case of alarm interrupts invoking of the ACTIVATION, in order to put the corresponding response program in the state “ready”
- kick-off SEARCH
Overall chart for the mini operating system

Second extension: Having the possibility of alarm interrupts in mind

- CLOCK IMPULSE INTERRUPT
- INTERRUPT1
- INTERRUPT2
- INTERRUPT K
- INTERRUPT ADMINISTRATION
- TIME ADMINISTRATION
- TASK ADMINISTRATION
- PROCESSOR ADMINISTRATION
- Computer process code
Third extension of the software system design

Operating resource administration for input/output devices
In/output operation are slower than
  – analog to digital converter ca. 20 ms

Introduction of a administration program I/O-ADMINISTRATION

Task:

Organization of slow input/output operations
  – Computer process is stopped.
  – Processor is able to work on other computer processes.
  – Finishing the input/output operations allows the continuation of accompanying computer processes.
Hierarchical chart of the mini operating system

MINI-REAL-TIME OPERATING SYSTEM (Extension 3)

- IN/OUTPUT ADMINISTRATION
- INTERRUPT ADMINISTRATION
- TIME ADMINISTRATION
- TASK ADMINISTRATION
- PROCESSOR ADMINISTRATION
Abolition of the simplifications

- Operating system programs themselves are not interruptible.
- The multiple instruction of a computer process, i.e. new instruction before the actual end of a computer process is impossible.
- A mutual instruction of computer processes is not possible.
- A synchronization of computer processes, i.e. through semaphore operations, is not possible.
- No data communication between the computer processes, i.e. no interchange of data, no common use of data.
- No dynamical modifications of the priorities of the computer processes during the program execution.
- Computer processes are located in the working memory, background memories are not available.
§ 5 Real-time operating systems

5.1 Definition
5.2 Organization tasks of a real-time operating system
5.3 Development of a mini-real-time operating system
5.4 Software system design of the mini real-time operating system
5.5 Examples for real-time operating systems
Market survey

Criteria of the selection of real time operating systems

– Development and target environment
– Modularity and kernel size
– Performance data
  • Amount of tasks
  • Priority levels
  • Task switch times
  • Interrupt latency time
– Adaptation to special target environments
– General characteristics
  • Scheduling method
  • Inter-task communication
  • Network communication
  • Design of user interface
### Selection of commercial real time operating systems

<table>
<thead>
<tr>
<th>Product</th>
<th>ERCOS</th>
<th>Lynx-OS</th>
<th>OS/9</th>
<th>OSE Delta</th>
<th>pSOS</th>
<th>PXROS</th>
<th>QNX</th>
<th>VRTX32</th>
<th>VxWorks</th>
<th>Windows CE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Embedded</td>
<td>Embedded</td>
<td>Embedded</td>
<td>Embedded</td>
<td>Embedded</td>
<td>Embedded</td>
<td>Embedded</td>
<td>Embedded</td>
<td>Embedded</td>
<td>Embedded</td>
</tr>
<tr>
<td><strong>Target architecture</strong></td>
<td>8016x, PowerPC</td>
<td>680x0, 80x86, PowerPC, PowerPC, 80000, i860, MIPS, SPARC, RS6000</td>
<td>680x0, PowerPC, PowerPC, PowerPC</td>
<td>680x0, 80x86, PowerPC, CPU32, AMD29k</td>
<td>680x86, 8016x, PowerPC</td>
<td>80x86, 8016x, Pentium</td>
<td>80286(16 bit)</td>
<td>680x0, 80x86, PowerPC, CPU32, i960, Hitachi SH, MIPS</td>
<td>Pentium 80x86, i486 PowerPC MIPS Hitachi S4, ARM</td>
<td></td>
</tr>
<tr>
<td><strong>Host-system</strong></td>
<td>UNIX, Win95, NT</td>
<td>UNIX, Windows</td>
<td>UNIX, Windows, NT</td>
<td>UNIX, SUN, Windows, NT, OS/2</td>
<td>UNIX, SUN, Windows, NT, OS/2</td>
<td>QNX</td>
<td>UNIX, SUN, Windows</td>
<td>UNIX, Win95, NT</td>
<td>Windows CE Win 95 NT</td>
<td></td>
</tr>
<tr>
<td><strong>Data system</strong></td>
<td>no</td>
<td>UNIX, FAT, NFS, Real-Time Filesystem</td>
<td>FAT</td>
<td>UNIX, FAT</td>
<td>UNIX, FAT, NFS, Real-Time Filesystem</td>
<td>UNIX, FAT</td>
<td>UNIX, FAT, ISO9660</td>
<td>UNIX, FAT</td>
<td>UNIX, FAT</td>
<td>FAT</td>
</tr>
</tbody>
</table>
### 5.5 Examples for real-time operating systems

<table>
<thead>
<tr>
<th>Product</th>
<th>ERCOS</th>
<th>Lynx-OS</th>
<th>OS/9</th>
<th>OSE Delta</th>
<th>pSOS</th>
<th>PXROS</th>
<th>QNX</th>
<th>VRTX32</th>
<th>VxWorks</th>
<th>Windows CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field bus</td>
<td>CAN</td>
<td>CAN, PROFI-BUS, Interbus-S</td>
<td>CAN</td>
<td>CAN, PROFI-BUS</td>
<td>CAN</td>
<td>CAN, PROFI-BUS, LON</td>
<td>CAN</td>
<td>CAN, PROFI-BUS, LON</td>
<td>CAN, PROFI-BUS, LON</td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>ROM-able</td>
<td>ROM-able, Multiprocessor, self-hosted</td>
<td>ROM-able, Multi-processor</td>
<td>ROM-able, Multi-processor, fehlertolerant</td>
<td>ROM-able, Multi-processor</td>
<td>ROM-able, Multi-processor, POSIX 1003 compliant</td>
<td>ROM-able</td>
<td>ROM-able, Multi-processor, POSIX 1003 compliant</td>
<td>ROM-able</td>
<td></td>
</tr>
<tr>
<td>Scheduling</td>
<td>preemptive, co-operative, priority controlled</td>
<td>preemptive, co-operative, priority controlled, Round-Robin</td>
<td>preemptive, co-operative, priority controlled, Round-Robin</td>
<td>preemptive, priority controlled, Round-Robin</td>
<td>preemptive, priority controlled, Round-Robin</td>
<td>preemptive, priority controlled, Round-Robin</td>
<td>preemptive, priority controlled, Round-Robin</td>
<td>preemptive, priority controlled, Round-Robin</td>
<td>preemptive, priority controlled</td>
<td></td>
</tr>
<tr>
<td>Task switch time</td>
<td>&lt; 54 µs 8016x (20 MHz)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.7 µs Pentium 166, 11.1 µs 486DX4 (100MHz), 74</td>
<td>17 µs</td>
<td></td>
<td>≥ 100 µs</td>
<td></td>
</tr>
</tbody>
</table>
Question referring to Chapter 5.2

Consider two different automation systems:

- an event-driven system (e.g. a control of a coffee machine)
- a time-driven system (e.g. a trajectory control of a robot)

For which type of system is the interrupt handling of an operating system more important?

Answer

In event-driven systems most of the processes are started by interrupt signals.

In time-driven systems no interrupts are caused. The events are handled during the next cycle.
Question referring to Chapter 5.4

Scheduling methods for the allocation of the processor are very important in real-time operating systems.

a) What is the purpose of those methods?

b) In which module of the mini operating system presented in the lecture a scheduling method is used? How is it called?

Answer

a) These methods are used to determine the execution sequence of the “runnable” tasks.

b) In the module SEARCH a scheduling method is used. It is the method of fixed priorities in which running tasks can be interrupted (preemptive scheduling).