RTOS-UH

-

Introduction and Overview

Andreas Hadler

1 Contents

2 Realtime-Systems

RTOS-UH is a realtime multitasking operating system and differs in its very conception from most other common operating systems. The operating system is an attempt to satisfy the special requirements for measuring, control and regulation technology with two special strategies. As the name realtime multitasking operating system itself implies: the operating system is realtime-capable and can work with several tasks.

As a looser definition, you could say:

A system is called realtime-capable when it can react with sufficient speed to external events at any time.

Although this definition spoils every purist´s fun in technical discussion, it is pragmatically accurate. It normally makes no difference whether the system complies with the regulations of hard realtime computing or is a real realtime system. Usually it are the pragmatic questions "Is the system fast enough for my requirements? Can I control my machine/plant etc. with the system? Can I react quickly enough to limit switches, inputs, etc.?" which are important. In the early days of using computers in measuring, control and regulation technology, the much clearer term keep-step data processing was used. Here it is clear what is meant: the computer can keep in step with its environment, i.e. react so quickly to external events that it does not miss anything.

2.1 Solutions

To fulfill these requirements of speed of reaction and computing speed, two different routes can be followed essentially:

- "brute force"

  By using pure computing force you can induce a behaviour which satisfies all realtime requirements. If a computer is not fast enough to solve a problem, the next fastest version is used. If this is still not enough, 2, 4 or in gigantic solutions up to 65536 computers are used and the problem is redefined until it can be solved in independent parts.

  By mere use of material, every problem can be solved which is solvable in principle unless the communication and synchronization of these sometimes countless computers require overproportional computing capacity and the point is reached at which an increase in the number of computers may even reduce the reaction capability.

- Algorithmic intelligence

  The intellectual and if you like, aesthetically more satisfying solution is in the development of suitable and efficient algorithms. As an example let us take the multiplication of two numbers: the multiplication 7*3 can be broken down to the addition 3+3+3+3+3+3+3 (of course a little thought will tell you that 7+7+7 is the better solution). However it does not become clear what advantage the logarithming with subsequent addition offers until 17839*15757: instead of carrying out 15757 additions, the problem can be solved by forming two logarithms and adding them.

Both methods of solution are successful, one with a greater material and the other with a greater thinking effort.

2.2 Responsiveness and Interrupts

Realtime systems try to take some of the thinking workload off the user. The main problem in the realization of controllers and regulators is guaranteeing sufficient speed of reaction to reliably control the system. In addition to calculating frequently time-consuming control algorithms, limit switches have to be monitored, limit values for pressure, temperature, etc. checked and reacted upon in the prescribed time. The necessary reaction times are often very short in comparison with the computing requirements of the control so that a great deal of thought needs to go into the maintaining of the reaction time during processing of complicated control processes when programming the controller. Realtime systems offer the programmer mechanisms for simple realization of high speeds of reaction.

This high reaction speed is usually guaranteed by a sophisticated and efficient handling of program interrupt signals, the interrupts. Interrupts are signals which are fed externally into the CPU of the computer. When one of these signals becomes active, the central unit interrupts your current work and branches to a special program section, the interrupt handler. Realtime systems are distinguished by the fact that this interrupt handling is particularly efficient and takes place according to a standard procedure. Unlike common systems, a programmer of a realtime system can be certain that the program interrupt is processed virtually delay-free and there is no need for him to program interrupt mechanisms himself and continue running programs. The programmer can program the two problems, the control algorithm and the reaction to limit switches almost independently and be sure that the realtime system takes care of a suitable and fast switching between program sections. Of course the programmer must inform the system which program section is assigned to which interrupt; however, this is far less trouble than programming the interrupt handler himself.

2.3 Multitasking

The second element for simplifying the programming of problems in measuring, control and regulation technology is the so-called multitasking. A task is a program which can run independently. In a computer capable of multitasking, several of these tasks may exist simultaneously and the computer´s operating system takes care of assigning the computing capacity to the individual tasks. The simplest assignment strategy is known under the term time-sharing: each task is assigned computing capacity for a fixed time, a so-called time slot, usually between approx. 20 milliseconds and 2 seconds and can then operate. At the end of a time slot the operating system carries out a task change and executes another task.

The original concept which led to the development of multitasking was the use of a computer by several users who can all work on one computer at the same time and run their programs quasi-simultaneously. These so-called multi-user-systems enable a computer to be used simultaneously by several users and save the investment in individual workstation computers. Of course there is the added advantage that a single user has much more computing capacity at his disposal at times when only a few users are working on the central computer than if he had only one workstation computer.

The multitasking also offers advantages for programming system controls: the division of a control task into complex algorithms and reaction-critical limit value checks described above can be done efficiently and elegantly by using multitasking. Both problem solutions are formulated in separate tasks and programmed independently of each other. The operating system takes care of switching between the individual tasks.

To ensure a sufficiently fast reaction of the program package consisting of several tasks to external events, the task switching mechanism of the operating system must be designed according to the special requirements of the realtime programming. Interrupts (externally applied interrupt signals)

can be used for this. This has two advantages over the pure time slot control: on the one hand task changes only take place when they are really necessary and on the other hand, tasks which process complex algorithms and do not wish react to interrupts run undisturbed.

2.4 The RTOS-UH Offer

RTOS-UH is an attempt, with as fast as possible and simple reaction to interrupts and a clear multitasking concept, to provide the tools to simplify the solving of special problems in measuring, control and regulation technology by the programmer. The multitasking which is only available on system level under other systems is made directly accessible to every programmer under RTOS-UH and can be used very clearly for solving smaller problems. RTOS-UH supports, especially in connection with the programming language PEARL (Process and Experiment Automation Realtime Language), the division of a problem solution into more manageable individual programs (tasks) which can be programmed, tested and executed independently of each other. With the possibility of assigning priorities to individual tasks a problem-oriented process of the entire program system is ensured. The operating system assigns the individual tasks computing time in the order of their priority. Together with the very fast reaction to interrupts and the possibility of planning and executing task changes as a reaction to interrupts an efficient tool is provided.

Since using this system requires the programmer to get used to an unfamiliar working method - the division of a problem into smaller independent parts, the definition of the individual tasks with their priorities, the recognition of necessary synchronization of the individual tasks, the assignment of individual program sections to individual interrupts, etc. - it is advisable to study the fundamental functional principles of the RTOS-UH to ensure successful and efficient use of the system.

Starting with a description of the different task properties and statuses, the significance of the tasks which already exist permanently in the system and their interaction with interrupt routines is explained below. When you are familiar with the principle function of the RTOS-UH, the I/O system and its function which differs very greatly from common operating systems, is described. Based on this, the operation of the RTOS-UH-own editor is then explained to give a few examples of what has been explained up till now. A short description of the programming language PEARL ends this overview of the RTOS-UH.

The type of description will vary greatly: cursory overviews and detailed explanations will be mixed up with each other. It is necessary to study at least the first few sentences of a chapter to get a rough idea of the operating system and its special features. Detailed explanations can be ignored initially but are essential for effective work with the operating system.

This text is not meant to replace study of the system manual and therefore does not give instructions on actions in the correct detail in places, it does however help to understand the often rather abstract explanations of the manual.

In this overview, theory and practice are dealt with separately because the author was of the opinion that actual working on the computer requires at least some knowledge of the functioning of the operating system. This is also meant to avoid that working on the computer shows up too many phenomena with which you are not familiar in theory which you would initially have to avoid elegantly.

3 RTOS-UH - The Operating System

RTOS-UH as an operating system should manage the resources of a computer (memory, processor, I/O interfaces) and provide user programs with service for simple use of these resources. To fulfill this task efficiently, RTOS-UH makes use of three tools: these are

- an operating system nucleus which controls the manipulation of tasks by system calls, traps and manages the system memory,
- interrupt routines which handle fundamental I/O operations and
- system tasks which take care of more complex management tasks.

Since the term task is fundamental to understandling RTOS-UH, it should be explained in more detail. Then it will be explained why interrupt routines are part of the operating system and how these cooperate with the system tasks.

3.1 Tasks and Tasking

The term task is fundamental to the RTOS-UH and identifies an independently runnable program. Under RTOS-UH these tasks may have different properties and be in different statuses. The operating system divides the available computing capacity between the individual tasks (multitasking) such that the change from one task to another is possible at any time (realtime capability). In multiprocessor computers, several tasks may be active simultaneously, in a single-processor system of course only one task can be executed at once. In the text which follows, a single-processor system is always assumed, the analogous transposition to a multi-processor system is relatively easy.

3.1.1 Task-Properties

RTOS-UH understands a task as an independently operating program. In order to be recognized as a task under RTOS-UH, this program must have its own properties. These properties must have been noted by the respective task in a management block, the task header, according to the rules of RTOS-UH.

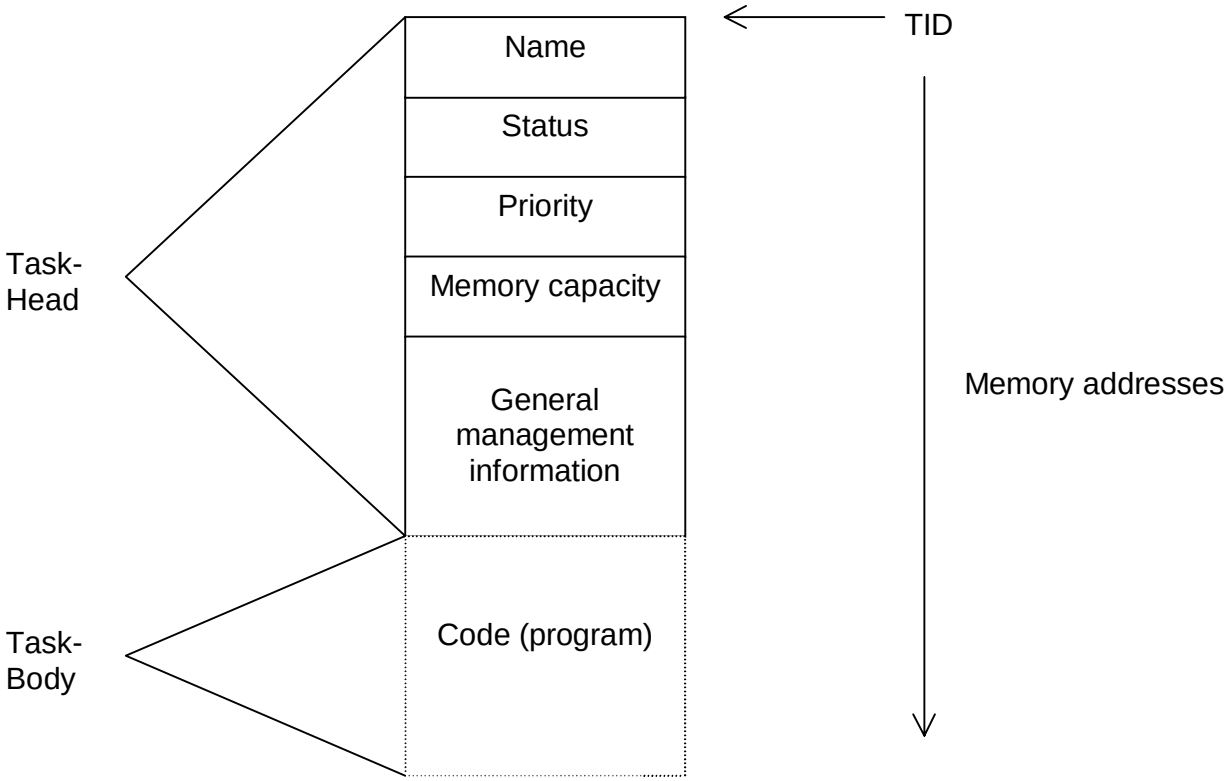

Abbildung 3-1: Structure of a task

High-level-language programmers normally do not need to give any thought to the generation of these task headers, this is taken over by the compiler, assembler programmers, however, must generate this headers themselves. After loading a program, this task header must be available in the memory; if under RTOS-UH a task address is referred to, the address of the header is meant, the so-called TID (task identification). In addition to the information explained explicitly here, the task header contains a lot more data which is partly only operating system-internal.

3.1.1.1 Task Name

The task name is, next to the TID, another important part of the task. All tasks in a computer should not only be distinguished by their TIDs but also by their names. Problems with identical names are posed at the latest when a task is to be started by its name.

A task name may consist of a maximum 24 characters at present.

3.1.1.2 Status

The status of a task indicates whether and if necessary how a task is involved in any activities. The status can be controlled by the operator with commands and by tasks and with the operating system by means of tasking operations.

3.1.1.3 Priority

The most important property of a task for RTOS-UH is its urgency, the priority. RTOS-UH divides the processor capacity between the runnable tasks according to their individual priority. Priorities are specified as integers in the range -32768...32767 in descending order of priority. A task with priority 1 is therefore more urgent than a task with priority 5. Unlike numeric values, the task with

priority 1 is said to have a higher priority than the task with priority 5. Negative values are reserved for the operating system itself, at least intentionally.

Every time the RTOS-UH has reason to assign the processor to a task, i.e. to execute a task, the process switch of the RTOS-UH, the so-called Dispatcher, inspects a list of available tasks and assigns the processor to the task with the highest priority which is ready to run. In case no task is ready to run, RTOS-UH contains an empty task, task named IDLE, which has the lowest priority and which is always ready to run. This task has no activities but is very important to the function of the operating system.

3.1.1.4 Memory Requirement

In order to be able to keep as many tasks as possible in the computer, RTOS-UH only assigns the tasks memory space for their local variables when starting. The size of the necessary task working memory, the so-called task workspace (TWSP), must therefore be entered in the task header. When starting a task, under RTOS-UH, activation of a task is referred to, RTOS-UH assigns this task memory space of sufficient size. If a task ends its work, under RTOS-UH this is referred to as terminating a task, this TWSP is free again and can be assigned by the operating system.

3.1.1.5 Resident Tasks

If a task declares itself resident, its TWSP is only assigned by the operating system the first time it is activated. This method has the advantage that new memory space does not need to be assigned (time-saving) every time for frequently activated tasks and on the other hand enables a task to keep static variables, i.e. variables the values of which are retained up to the next activation when terminated.

3.1.1.6 Autostart Capability

Especially important for stand-alone systems: if a task declares itself as an autostart task, it will be activated by the operating system as soon as this has started. All tasks which do not have this property are not activated automatically. They can only be started by other tasks or by operating commands.

Under RTOS-UH there is no other way for tasks to become runnable of their own accord.

3.2 Multi-Tasking

The concept of RTOS-UH is based on the idea of multitasking, the ability of the operating system to handle several autonomous (and independently runnable) programs (quasi) simultaneously. These tasks may be available simultaneously in the computer; the operating system assigns the available processor capacity to the tasks which are ready to run.

3.2.1 Task Statuses

The phrase ready to run already indicates that RTOS-UH can identify different operating statuses for a task. RTOS-UH manages the tasks in their different operating statuses and provides mechanisms for initiating and checking status changes. Figure 2.2 shows the most important operating statuses and the designation of the possible changes. The following text explains this.

sind alle Betriebszustände sowie die möglichen Übergänge nebst den diese veranlassenden Bedienbefehlen exemplarisch aufgeführt. Der folgende Text gibt hierzu Erläuterungen.

In addition to the basic task statuses DORM (dormant), RUN (ready to run), SUSP (suspended), SCHD (scheduled) and SEMA (waiting for synchronization) which can be achieved directly by task or operator activities, the more system-internal statuses I/O? (waiting for termination of an I/O operation), PWS? (waiting for workspace), ???? (multiple blocking) and CWS? (waiting for I/O space) are also described here.

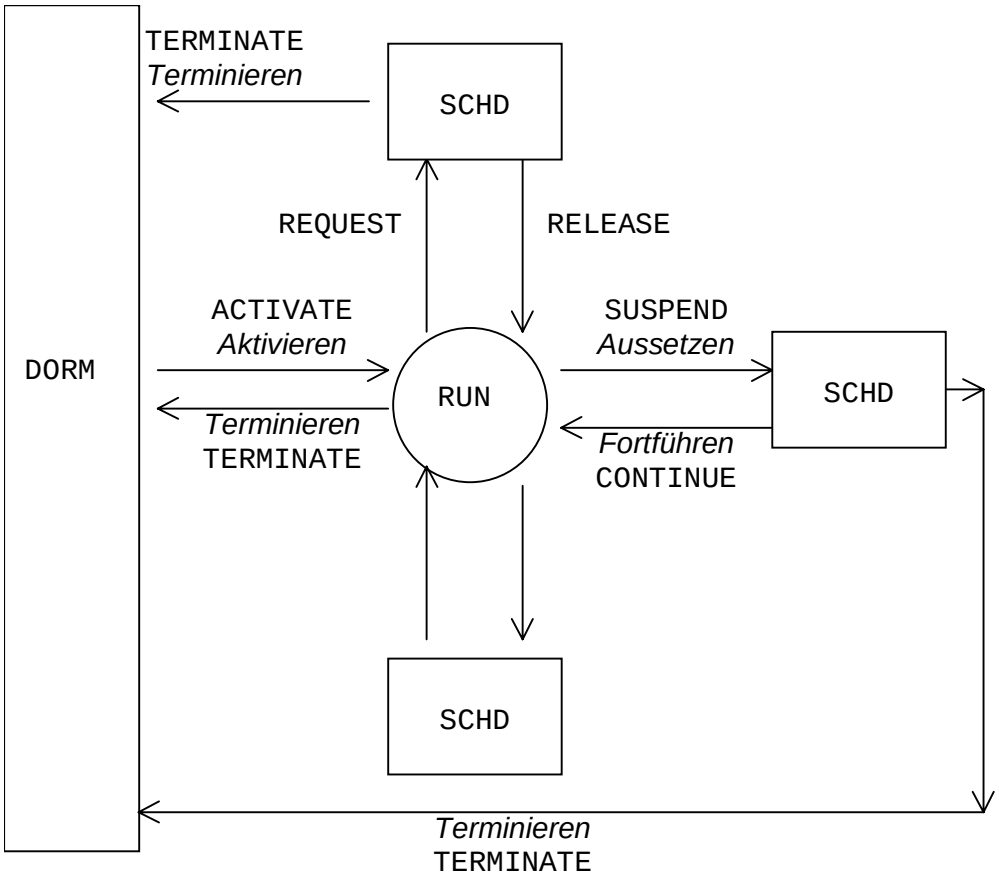

Abbildung 3-2: Status changes

Italics: Name of the status change

`Normal`: Command for the status change

3.2.1.1 DORM

The simplest operating status is called DORM (dormant). A task in the DORM status is loaded in the computer but is not ready to run. It will never be assigned processor space unless its operating status is changed by external influences. A dormant task only takes up space from the operating system for its code and if nec. (resident tasks!) for static variables. In all other operating statuses a task certainly has memory space for its TWSP (only exception see change Activate).

3.2.1.2 RUN

A task is ready to run in the RUN status. As soon as no task with a higher priority is ready to run, this task is assigned the processor and can execute its job. The change from the DORM status to the RUN status is known as Activation of this task. As soon as the task is assigned the processor it starts processing at the start of the program. It can only be activated externally, i.e. by another task or by the operator with the ACTIVATE taskname command. It is obvious that a task cannot change itself from the DORM status to the RUN status because it has no processor space in the dormant status and cannot possibly initiate its own activation. However, a task which has started running can activate itself. In this case this activation is buffered, i.e. saved for later. If the task wants to or has to return to the DORM status later, RTOS-UH determines this buffered activation and resets the task to the RUN status. The task is then executed again from the start. RTOS-UH can buffer up to 3 activations for every task.

Allerdings kann eine einmal laufende Task sich selbst aktivieren. In diesem Fall wird diese Aktivierung gepuffert, d. h. für später aufgehoben. Will oder soll diese Task später wieder in den Zustand **DORM** übergehen, so stellt RTOS-UH diese gepufferte Aktivierung fest und versetzt die Task wieder in den Zustand **RUN**. Die Task wird dann erneut von Beginn an ausgeführt. RTOS-UH kann max. 3 Aktivierungen für jede Task puffern.

3.2.1.3 SUSP

A task in the SUSP status, suspended, is ready to run in principle and has been assigned processor space once or several times but does not want any processor space at present. It has suspended the processing of its job voluntarily or controlled externally for an indefinite period. RTOS-UH bypasses a suspended task when assigning the processor capacity.

The SUSP status can be set by the operator with the SUSPEND command taskname. The task concerned stops processing where it is and can be continued later with the CONTINUE taskname command.

3.2.1.4 SCHD

A task is scheduled. The status is similar to SUSP in that this task is principally ready to run but does not want any processor space at present. However, a condition is already defined for this task under which it returns to the RUN state. This condition may be a scheduled planning (e.g. by the operator AT 17:00:00 ACTIVATE taskname) or a planning in to an interrupt (e.g. by the operator WHEN EV 00000001 ACTIVATE taskname); see page 15 for more details on planning in.

The change to the SCHD status is possible from all statuses except SEMA. In particular the status SCHD alone does not reveal whether this task has already been assigned processor space or is only activated by the planning in.

If the given planning in condition is satisfied, RTOS-UH changes the appropriate task automatically to the RUN status and assigns processor space if no task of higher priority is ready to run.

3.2.1.5 I/O?

A task is waiting for termination of an input or output. Under RTOS-UH the actual input and output operations are not carried out directly by the initiating task (see page 41) but by so-called support tasks. A task in the I/O? status has triggered an input or output operation and is waiting for its termination. For this period it is not assigned processor space. As soon as the I/O operation is terminated, RTOS-UH returns this task to the RUN status.

3.2.1.6 PWS?

A task is waiting for assignment of memory space, procedure workspace, PWSP. It has asked the operating system for workspace of a size which is presently unavailable. Since RTOS-UH manges the memory dynamically, sufficient space may become available at any time. RTOS-UH then satisfies the request for workspace (in the order of priority if several tasks are waiting) and switches the task immediately to the RUN status.

3.2.1.7 CWS?

A task is waiting for workspace to be assigned for I/O operations. This is a special form of the PWS ? status which is not due to there not being enough free space in the system but to exceeding the space contingent for I/O operations. RTOS-UH limits the workspace provided for every single task at every single activation for I/O operations to 2 KB to prevent individual tasks reducing the system workspace excessively by numerous outputs.

3.2.1.8 ????

A task is multiply blocked, e.g. if it was suspended by the operator when still in the PWS? status. The output of ???? as a task status is to be considered as an act of desperation on the part of the operating system which is no longer able to report the task status in one word. Internally the combination of different statuses is handled correctly.

However, the output of ???? as a task status may also point to an act of self defence by the operating system: if a task has executed an illegal action which the operating system can no longer prevent but only observe (e.g. memory access error for priveleged memory accesses) the task is put in the ???? status to reliably prevent the task from continuing.

3.2.2 Task Status Changes

Having explained the individual statuses which the tasks may have, the possibilities of initiating and influencing status changes are described here. This only concerns the basic task statuses DORM, RUN, SUSP, SCHD and SEMA. The operations which can initiate status changes are Activate, Terminate, Suspend, Continue, Plan in, Plan out and synchronization operations as well as the entry of times and interrupts.

The effect of the operation and the operating commands to be given (and the analogous PEARL instructions) are described roughly, you will find detailed descriptions in the manual, chapter Operating Commands, and for the assembler programmer in the manual, chapter Traps.

3.2.2.1 Activate

Activate only has a direct effect on tasks in the DORM status: they are switched to the RUN status and program execution begins with the assignment of the processor at the start of the program. Activation of a task in another status is possible and causes a so-called buffered activation. RTOS-UH remembers (in the task header, where else?) that this task has been activated and starts it again from the start of the program as soon as it wants to go back into the DORM status.

In the Activation RTOS-UH first assigns the task space for its local variables (TWSP), then the task is started from the beginning. If there is not enough space in the system, the task is still switched to the RUN status. RTOS-UH then checks for every change in the space assignment whether the request can be granted and assigns the processor only if the task has sufficient TWSP. The activation can be initiated by the user or any other task. Tasks can activate themselves and other tasks.

Activation is initiated by the operating command ACTIVATE taskname, also in the abbreviated forms A taskname and ´Taskname or simply by specifying the taskname. In PEARL, the ACTIVATE instruction is available, assembler programmers can use the traps ACT and ACTQ.

ACTIVATE ***Taskname***, auch in den Kurzformen **A** ***Taskname*** und **'** ***Taskname***

oder allein durch die Angabe des Tasknamens veranlasst. In PEARL steht hierzu die **ACTIVATE**-Anweisung zur Verfügung, Assembler-Programmierer können die Traps **ACT** und **ACTQ** benutzen.

3.2.2.2 Terminate

Termination is the opposite action to activation. The execution of a task is aborted at the point which the task has reached. The task is immediately switched to the DORM status unless there is a buffered activation.

When terminating, all assigned space is withdrawn from the task (exception: TWSP of resident tasks and I/O memory sections) and declared as free space. If a task has already triggered I/O operations which are not yet ended, the following procedure runs: all outputs are executed, all inputs except those which may be currently running are rejected, i.e. if a task has triggered inputs, only those inputs which are currently being processed by an I/O support task are finished, all other inputs are rejected.

- alle Ausgaben werden durchgeführt
- alle Eingaben, bis auf evtl. gerade in Bearbeitung befindliche, werden verworfen, d.h. hat eine Task Eingaben veranlasst, so werden nur diejenigen Eingaben, die gerade von einer I/O-Betreuungstask bearbeitet werden, zu Ende bearbeitet, alle anderen Eingaben werden verworfen.

Termination is possible in every task status. Even a task which is already DORM can be terminated; the operating system does not execute any actions in this case, however. The termination can be initiated by the operator and by another task; for tasks both self and external termination is permitted.

Termination is initiated by the operating command TERMINATE taskname, also in the abbreviated form T taskname, by the PEARL instruction TERMINATE and the traps TERMI, TERME, TERMEQ and TERV.

TERMINATE ***Taskname,*** Kurzform **T** ***Taskname***,

durch die PEARL-Anweisung **TERMINATE** und die Traps **TERMI**, **TERME**, **TERMEQ** und **TERV** veranlasst.

3.2.2.3 Suspend

Suspend switches a task from the RUN to the SUSP status. The execution of the task is suspended for an indefinite period, the task is frozen so to speak in its momentary status. A suspended task is bypassed in the assignment of processor workspace. All local variables are retained as well as every assigned memory. I/O operations triggered by the task are executed.

Suspend can be initiated both by the operator and by tasks; tasks can suspend execution of themselves or other tasks. For a single task it is possible to suspend with a continue condition, see Planning in with suspend. Suspend is initiated by the operating command SUSPEND taskname, abbreviated form SU taskname, the PEARL instruction SUSPEND and the trap SUSP.

SUSPEND ***Taskname***, Kurzform **SU** ***Taskname***,

die PEARL-Anweisung **SUSPEND** sowie den Trap **SUSP**.

3.2.2.4 Continue

Continue is the opposite operation to suspend. A task is switched from the SUSP status to the RUN status, i.e. it is immediately reconsidered for the assignment of processor space. The task resumes its activity at exactly the same place as it was suspended. For the task itself no difference is noticeable from uninterrupted execution except for a change in the time.

Continue can be initiated by the operator and by another task. It should be obvious that a suspended task has no way of recontinuing itself, unless it starts the suspension with a planned in continue (see planning in).

CONTINUE *Taskname*, kurz **CONT *Taskname***,

the PEARL instruction CONTINUE and the traps CON and CONQ.

3.2.2.5 Planning In

Planning ins are the nucleus of RTOS-UH. The ability to make activation and continuance of tasks dependent on the occurence of different events is an essential feature of the operating system and enables program systems to be created which can react very quickly and very flexibly to external situations.

Planning ins can be distinguished on the one hand by the type of event, time or interrupt and on the other hand by the type of planned operation, activation or continue.

All planning ins have in common the status change from DORM, RUN or SUSP to the SCHD status. The planning ins can be made by the operator or a task, independent planning in of a task is also possible (of course not if this task is in the DORM status).

- chronologically planned activation

  A chronologically planned activation leads to starting a task at a specified time or after a specified time. It can be determined that this activation be repeated at certain intervals und this cyclic planning can be restricted to a certain period of time. RTOS-UH manages time and duration with the resolution of 1 ms; the longest processable duration is approx. 24 days.

  Planning in to a time is executed by the operating command

  AT *Uhrzeit* ACTIVATE *Taskname*,

  planning in to a duration with the command

  AFTER *Zeitdauer* ACTIVATE *Taskname*.

  The PEARL instructions correspond to the operating commands.

- cyclic planning in

  A cyclic planning in is executed with the operating command

  ALL *Zeitdauer* ACTIVATE *Taskname*

  and can be limited to a certain period of time in the form

  ALL *Zeitdauer* DURING *Zeitdauer* ACTIVATE *Taskname*

  Instead of limiting to a specific period of time, a time for the last activation can be given in the form

  ALL *Zeitdauer* UNTIL *Uhrzeit* ACTIVATE *Taskname*

gegeben werden. The PEARL instructions correspond to the operating commands. Cyclic planning ins in the simple form result in immediate activation, i.e. the first of the cyclic activations takes place immediately. If this is not desired, the cyclic planning in can be combined with planning in to a time or a duration, e.g..

AFTER ***Zeitdauer*** **ALL** ***Zeitdauer*** **DURING** ***Zeitdauer*** **ACTIVATE** ***Taskname***

or

AT ***Uhrzeit*** **ALL** ***Zeitdauer*** **UNTIL** ***Uhrzeit*** **ACTIVATE** ***Taskname***

Here too the PEARL instructions are identical to the operating commands. Assembler programmers have the traps TIAC and TIACQ at their disposal which can be used for all chronologically planned activations.

- activation planned in to interrupt

  Very important for the realtime operating system is the ability to react quickly to external events. External events are events which occur asynchronously to the program run and which are signalled to the processor by the periphery, so-called interrupts (in the regular program run).. RTOS-UH can make the activation of tasks dependent on the appearance of these interrupts, i.e. if an interrupt occurs, the task planned in for it is activated. If this task has a high priority the processor is assigned immediately and tasks of lower priority cannot obstruct handling of this interrupt situation.

  Planning in to an interrupt is executed with the command

  WHEN ***Interruptkennzeichnung*** **ACTIVATE** ***Taskname***,

  the same PEARL instruction or the traps ACTEV or EVACTQ..

- chronologically planned in continuation

  Chronologically planned in continuances correspond to the chronologically planned in activations, whereby neither a cyclic planning in nor limiting to a certain period of time or up to a certain time are possible. A chronologically planned in continuance is defined with the operating commands

  AT ***Uhrzeit*** **CONTINUE** ***Taskname***

  or

  AFTER ***Zeitdauer*** **CONTINUE** ***Taskname***,

  the analogous PEARL instructions or the traps TICON or TICONQ.

  If a task wishes to suspend its execution for a certain time, it can do so with the PEARL instructions

  AT ***Uhrzeit*** **RESUME**

  and

  AFTER ***Zeitdauer*** **RESUME**

  or the trap TIRE. The task puts itself in the SCHD status and rejects assignment of processor capacity until the desired time is reached or the desired period of time has expired.

- planned in continuance to interrupt

  The continuance planned into an interrupt can be understood as analogous to activation planned into an interrupt. For a task, it is defined that it is to be continued when an interrupt signal arrives. On condition, however, that this task is suspended until the interrupt arrives, otherwise

the error message taskname NOT SUSPENDED appears when the interrupt arrives because it is impossible to continue a task which has not been suspended. Planning in is defined by the operating command

WHEN ***Interruptkennzeichnung*** **CONTINUE** ***Taskname***,

the analogous PEARL instructions or the traps CONEV or EVCONQ.

If a task wishes to suspend itself at the same time and plan in continue at an interrupt, it can use the PEARL instruction

WHEN ***Interruptkennzeichnung*** **RESUME**,

or the combination of the traps CONEV or EVCONQ with the trap SUSP.

3.2.2.6 Plan Out

Planning out is the opposite operation to planning in. Planning outs can only be done generally for a task, i.e. differentiation according to the type of planning in is not possible. All the planning ins peformed for a task are deleted by a planning out. A task which is in the process of running will not be terminated by this, the current run status of the task concerned is not changed.

The most common changes of status are SCHD - DORM, if a task was planned in for activation and SCHD - SUSP, if a task was planned in for continue. A task can also delete its own planning ins.

The planning out is achieved by the operating command

PREVENT ***Taskname***,

the PEARL instructions PREVENT, if nec. with taskname, and the traps PREV and PREVQ.

3.2.2.7 Synchronization Operations

In a multitasking operating system it is often the case that several tasks need to access the same data base. If a task changes this data base, it must be ensured that the other tasks which only read out the data base always obtain consistent data. This requirement cannot be satisfied by the priority selection alone:

- If the changing task has the highest priority, it may make all changes unhindered but could interrupt a reading out task in the middle of the read process which leads to inconsistent data in the reading out task.
- If the changing task has the lowest priority, it can be interrupted in the middle of a data change, which means a reading out task gets inconsistent data.

To solve this problem, RTOS-UH provides synchronization variables, so-called semaphores, italian for traffic lights, and bolts.

3.2.2.8 Semaphores

The function of semaphores can be explained best by their similiarity with traffic lights. In the example, there is a bottleneck in the road which can be blocked off for entry of vehicles by the traffic lights on both sides. Induction coils are installed at both entry points in the direction of travel which reports request to drive in and detects leaving the bottleneck.

The simplest case consists of two vehicles approaching the bottleneck from opposite directions. The vehicle which reaches the drive in induction coil first gets green to proceed and automatically sets red for the opposite side so that the opposite vehicle´s request to drive in is rejected. The ve-

hicle clears the bottleneck when it drives over the exit induction coil and sets green for the waiting vehicle.

Transposed to semaphores under RTOS-UH, only one new terminology is required: the requesting entry permission by driving over the entry induction coil is called a REQUEST operation, exiting the bottleneck is called a RELEASE operation. The individual tasks can be seen as vehicles: task A places a request to the semaphore bottleneck and occupies the bottleneck. This operation has no effect on the status of task A. If task B then also executes a REQUEST operation, e.g. because it has become ready to run by an interrupt signal or an operator intervention and has been assigned the processor because of its higher priority, it is suspended, i.e. its execution is suspended despite the higher priority and its status is set to SEMA, i.e. waiting for the release of a semaphore. Reassignment of the processor is necessary and task A has a good chance of being the highest priority, runnable task and being assigned the processor.

When task A leaves the bottleneck it carries out a RELEASE operation to the semaphore bottleneck. This releases the bottleneck again. Transposed onto the computer, this means a new processor assignment is necessary and task B is assigned the processor on account of its higher priority.

Transposed from road bottlenecks to data sections, the integrity of the data base is guaranteed by the synchronization operations REQUEST and RELEASE.

From the operator level, semaphores can only be accessed through their address with the instructions

REQUEST ***Semaphoradresse***

and

RELEASE ***Semaphoradresse***

ansprechbar to make it simpler, an instruction

RELEASE ***Taskname***

From the operator level, semaphores can only be accessed through their address with the instructions REQUEST semaphore address and RELEASE semaphore address; to make it simpler, an instruction RELEASE taskname is possible which releases the semaphore for which a task is waiting in the SEMA status. Note that the latter form of instruction on the one hand does not address a certain semaphore specifically because the address of the semaphore variables concerned is not specified, but on the other hand always concerns one and only one semaphore because a task in the SEMA status can only wait for one semaphore.

The PEARL instructions REQUEST semaphore variable and RELEASE semaphore variable are available, the corresponding traps are REQU and RELEA.

Supplementary to what has already been described, semaphores are multipied implemented under RTOS-UH, i.e. depending on the pre-assignment of the semaphores, several REQUEST operations can be carried out without blocking. Then the synchronization of two tasks active as generator and consumer is possible: if the consumer carries out a REQUEST operation before using the generated data (which may be stored for example in a cyclic buffer) and the generator carries out a RELEASE operation after provision of a data set, synchronization of both is guaranteed.Der Verbraucher stets erst dann lauffähig, wenn zu bearbeitende Daten erzeugt wurden.

3.2.2.9 Bolts

Bolts operate quite differently to semaphores. They can distinguish between read only tasks and tasks which also modify data. Reading processes can signal the start of their access to the critical data section with the operation

ENTER ***Boltvariable***

and the end with the operation

LEAVE ***Boltvariable***

das Ende ihres Zugriffs auf den kritischen Datenbereich signalisieren. Writing processes, i.e. tasks which modify data, use the operation

RESERVE ***Boltvariable***

for entering and

FREE ***Boltvariable***

for leaving the critical path.

The aim of these differnt operations is not to obstruct reading processes if no writer wants to use the data. As long as no writer has carried out a RESERVE operation, the ENTER and LEAVE operations only serve to mark use of the data. The synchronization mechanism linked with the bolts does not come into action until a RESERVE operation is processed. If a reading process is in the critical path at this time, execution of the RESERVE-ing writer is suspended until the reader or readers have left the critical path. At the same time, entry into the critical path is blocked for readers and other writers. If there are no more readers in the critical path, i.e. all readers have executed their LEAVE operations, the writer is put back in the ready to run status and is assigned the processor. After processing the FREE operation allocated to the RESERVE, the critical path is released.

3.2.3 Event Entry

Changing from the SCHD status into the RUN status is only possible by entry of the event determined in the planning in, either by reaching a time, expiry of a period of time or arrival of an interrupt.

The operator only has limited influence on this. The time can be influenced with the CLOCKSET command (no PEARL equivalent, no trap); an interrupt can be simulated with the command

TRIGGER ***Interruptkennzeichnung***

The PEARL instruction for this corresponds to the operating command, the corresponding trap is TRIGEV.

3.2.4 Überblick über Taskzustandsübergänge

Hier die möglichen Zustandsübergänge mit veranlassenden Operationen im Überblick:

DORM	→	**RUN**	:	activate
DORM	→	**SCHD**	:	plan in
RUN	→	**DORM**	:	terminate
RUN	→	**SUSP**	:	suspend
RUN	→	**SCHD**	:	plan in with suspend

RUN	→	**SEMA**	:	synchronization operation REQUEST
SUSP	→	**DORM**	:	terminate
SUSP	→	**RUN**	:	continue
SUSP	→	**SCHD**	:	plan in
SCHD	→	**DORM**	:	plan out
SCHD	→	**RUN**	:	event entry
SCHD	→	**SUSP**	:	plan out
SEMA	→	**DORM**	:	terminate
SEMA	→	**RUN**	:	synchronization operation RELEASE
SEMA	→	**SCHD**	:	impossible
SCHD	→	**SEMA**	:	impossible
SUSP	→	**SEMA**	:	impossible
DORM	→	**SEMA**	:	impossible
DORM	->	**SUSP**	:	impossible

Tabelle 3-1: Task- Status changes

3.3 Interrupt Routines

The second most important element for the efficiency of RTOS-UH after concept of the tasks is the planned use of interrupt routines.

Interrupts are events triggered under certain circumstances by the periphery or the driver components for peripheral equipment. They interrupt the regular process of a program and are processed by special program segments, the interrupt routines. The figure below explains the usual program run and processing of an interrupt:

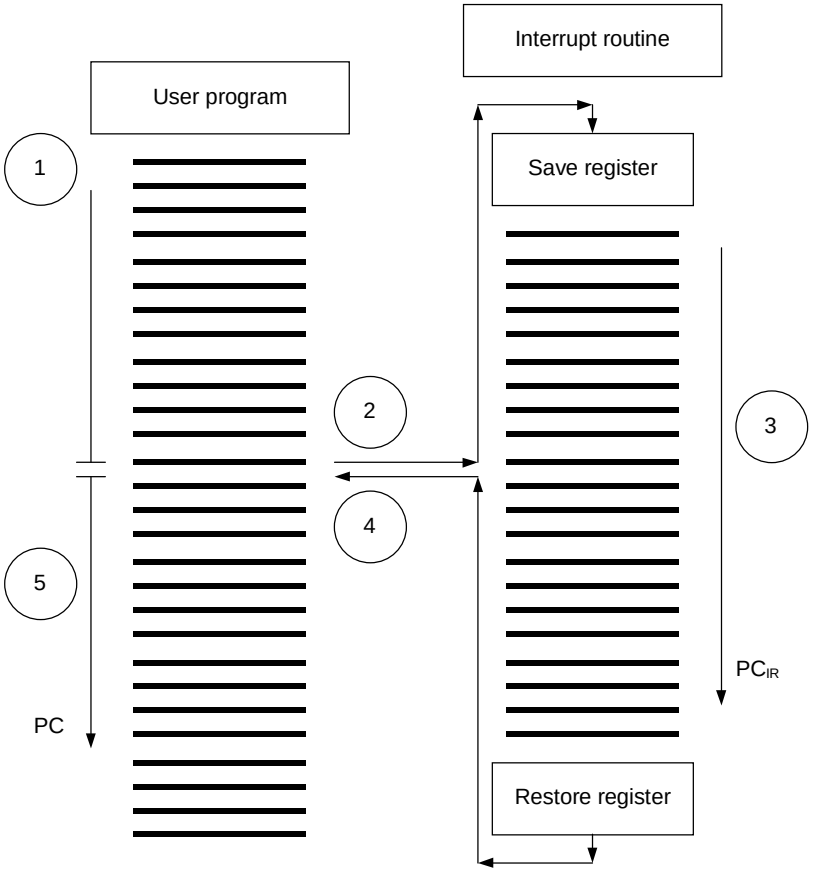

Abbildung 3-3: Program interrupt by interrupts

1. User program runs
2. Interrupt - save the processor state
3. Process the interrupt routine
4. Restore the processor state
5. User program runs

The reason for using interrupt routines is the slow functioning of peripheral devices in comparison with the processor. With a serial interface, the transfer of one character at 9600 baud takes about 1 ms, the processor, however, could send and receive the data about 1000 times faster. Since the transfer of a character is processed independently by the hardware, it is advisable to have the processor perform other tasks during this time. The processor is not required again until the character transfer has ended, either to provide a new character or to detect the end of the transfer. RTOS-UH uses these pauses by assigning the processor capacity to runnable tasks. Only when the processor is required to look after an I/O component, will it be used for this. The I/O components trigger an interrupt in this case which interrupts the regularly operating programs. RTOS-UH keeps special interrupt routines for looking after the components for such cases. This avoids processor capacity being wasted anywhere in the operating system by repeated polling of peripheral components.

Interrupt routines are outside the task concept of RTOS-UH because the interrupt reply is already available in the processor as a special case. The task change mechanism of RTOS-UH is paralysed for the duration of interrupt routine processing. To keep these paralysis times as low as possible, all interrupt routines of RTOS-UH operate together with system utilities which are usually created as tasks. Here too, the RTOS-UH-own task status change mechanisms are exploited: the system tasks suspend their execution or plan in to be continued again by the interrupt routines. Only the system programmer has controlled access to these interrupt routines: user programs always use the system tasks for this.

3.3.1 Timer-Interrupt

The only exception here is the system clock: a periodic interrupt which is triggered every millisecond in most RTOS-UH systems, manages the system time independently. The appropriate interrupt routine checks for every clock interrupt, also known as clock tick, whether a time is available for which a planning in exists. If so, the necessary measures are taken to change the task status and the task switch started to assign the processor to the highest priority task.

3.3.2 Interface Interrupt

As already mentioned above, output of characters through most serial and parallel interfaces is slow in comparison with the processor speed. Therefore there is an interrupt routine in the RTOS-UH for every interface which carries out input and output of characters independently of every task. These interrupt routines only work together with their system tasks for the start and end of the input and output process. For the duration of the actual input or output, these system tasks suspend their processing (are in the SUSP status) and are continued by the interrupt routines at the end of input/output.

3.3.3 Floppy-Interrupt

Longer waits also occur in the operation of mass memories. Therefore interrupt routines usually exist which enable the processor to be assigned to runnable tasks during wait periods. The corresponding system task also suspends its processing for this time and is continued by the interrupt routine if necessary.

3.4 System Tasks

Now that it has been explained what tasks are and what can be done with them, the function of the system tasks permanently related to RTOS-UH will be explained. The practical benefits of the task concept become clear first; many features of the operating system will become comprehensible and recognized as self-conclusive.

The function of the RTOS-UH is based on a number of system tasks which offer a number of key services in addition to the services in the nucleus of the operating system. All respectively available system tasks are necessary for the function of the operating system and fulfill their task without operator intervention. Therefore they are protected against illegal interventions in their very names: the character # with which the name begins cannot be entered by the operator as part of a valid taskname. For the sake of simplicity, the tasks are named below without this leading #. Not all conceivable or already realized system task types are listed below. For the support tasks in particular only the typical representatives are presented. Other system tasks with similar jobs behave similarly and are also realized similarly.

3.4.1 Support Tasks and Data Stations

A support task is responsible for the management of all requests made by the system or by user programs of a peripheral device. RTOS-UH contains a support task for every physically independent device. These devices are referred to as data stations under RTOS-UH because they are the destination or source for data transfers. Data stations have a name which is specified in the form /data station according to the RTOS-UH nomenclature. The exact meaning of the support tasks is explained in the chapter I/O system.

3.4.1.1 IDLE

A task with this name exists in every RTOS-UH system. It is the so-called idle task of the system which is always ready to run. This task has the lowest possible priority and is always assigned the processor when no other task is ready to run. The IDLE task serves no other purpose than its constant availability. The code processed by IDLE normally consists of an endless loop. However, systems do exist in which the IDLE stops the processor until an interrupt arrives. This saves power in battery-operated computers.

3.4.1.2 USER

The task with the name USER is the most important one for the user. This task forms the operating interpreter of an RTOS-UH system and is responsible for the input, evaluation and execution of operating commands. Once activated, it exhibits its readiness to receive commands by outputting the character * as a prompt. It then waits for entry of a command. On termination of the input it analyses the entered text and executes the received commands. The name USER comes from the fact that this task is the system-internal representative of the user in an RTOS-UH system.

The task USER operates with a very high priority in order to be able to receive and process operator interventions for high priority continuous tasks. This task also has all the necessary memory space even in the DORM status. Therefore it can still be started even in a computer suffering from an acute lack of memory space, e.g. to remedy the cause of the lack of memory space (termination of memory "muncher" etc.).

For more extensive commands, for example, compiling of a program, it generates independent tasks, so-called sub-tasks. These tasks are generated dynamically and process complex operating commands with low priority independently. This makes the operating interpreter ready to receive new commands earlier.

A good example of this is the COPY command. After giving a COPY command, a subtask is generated immediately which actually executes the copying process. Since the USER task has completed its job with the generation of the subtask, a new COPY command can be given immediately for example. Both copying procedures then run quasi-parallel processed by two separate subtasks. In wait periods of one COPY subtask, the other COPY subtask can operate and vice versa. In addition to the psychological effect that waiting for the computer is avoided, shorter copying time than in the chronological processing of copying tasks can be achieved. But be careful: if you are copying from or to the mass memory, the total copying time required may be greater than the sum of the times for single COPY commands because a lot more positioning processes are required on the mass memory.

Subtasks derive their name from the operating command to which the system appends a consecutive number (COPY/xx, the number xx is separated by an oblique stroke). At the end of the job they automatically disappear from the system again.

If an RTOS-UH has several tasks which begin with the name USER and are distinguished by a following digit, several jumps to the operating interpreter are available. This is normally the case in systems with several serial interfaces. Every single USERx task is responsible for command entries from one interface. Since a USER exists for every interface, operating commands can be entered independently of each other (multi-user facility). This means that several users can work on one computer.

However, RTOS-UH is not a real multi-user operating system: the individual users are not protected from each other in any way, files have no user assignment, tasks can be mutually manipulated, etc. The task USER1 usually has the highest priority of all USER tasks.

3.4.1.3 XCMMD

A task with the name XCCMD is also an operating interpreter in an RTOS-UH system. It gets its commands, however, not from operator inputs but from outputs from other programs. Every task can transmit operating commands by outputs to the data station /XC supported by the XCMMD. In this way programs can perform operations which do not belong to the scope of the respective programming language.

3.4.1.4 ACIA, SCC, RS232

A task with the name ACIA or SCC is responsible for looking after a serial interface. It receives input and output jobs from other system tasks or from user tasks, manages the operating status of the serial interface, fulfills the input/output jobs in cooperation with the appropriate interrupt routine and returns the completed jobs to the contractor. An ACIA cannot process several input or output jobs simultaneously.

The name ACIA or SCC comes from the name of typical components for operating serial interfaces.

This application range is typical for a so-called support task. In the course of an input or output job an ACIA runs through different task statuses. Normally DORM, it is activated by the system when a job arrives. In the RUN status it performs its management tasks and passes on the job to the appropriate interrupt routine. The ACIA then goes into the SUSP status. At the end of the input or output, it is continued by the interrupt routine. Back in the RUN status it performs further management tasks and returns the job to the contractor. Then it terminates itself.

If several serial interfaces are available in a system, several tasks ACIAx also exist, whereby x stands for the appropriate interface. The USER task assigned to a serial interface has the same identification x as the corresponding ACIA. The name of the assigned data station is /Ax. The interface supported by the ACIA1 is referred to as the system interface because it has access to the highest priority USER.

3.4.1.5 SOUT

As a rule a SOUTx exists for every ACIAx. A task SOUT is required for the output of data through a serial interface in full duplex mode, i.e. if transmission and reception are to take place simultaneously through one serial interface, the output is made via the SOUT and the input via the ACIA. A SOUT cannot process input requests, its name comes from the characterization serial out.

If outputs are demanded from ACIA and SOUT at the same time, both tasks control their access to the interface with synchronization variables, semaphores. Therefore a SOUT may also be in the SEMA status.

The name of the supported data station is /Dx, whereby x is selected as described in the ACIA.

3.4.1.6 EDFM

An RTOS-UH system which has an EDFM task contains a RAM disk. EDFM is the support task for this file-oriented data management in the system memory. The name EDFM comes from EDit File Manager which implies that there is an editor in RTOS-UH which can work directly on the RAM disk. The supported data station, the RAM disk is called /ED/.

EDFM has the same tasks as an ACIA but does not work together with an interrupt routine because there are no wait phases for the readiness of a peripheral device in the memory. As soon as EDFM has received a job, it carries it out without going into any other status than RUN.

3.4.1.7 ERROR

The ERROR is the highest priority task in an RTOS-UH system. It is responsible for processing error messages. By operating system-internal special handling of the ERROR, it can never be blocked such that it is no longer able to run.

The ERROR has a cyclic buffer in which error messages are stored with an error number and clear text if necessary. Directly after entering an error message, the ERROR is activated. It provides an error message for output through the ACIA and if necessary measures for error-tolerant continuation of the operating system.

In addition, the ERROR is responsible for activating the USERs: if the interrupt routine of an ACIA receives a special character, the ^A, ^B, ^C or a BREAK signal and activation of the USER is allowed, it transmits a special message to the ERROR. The ERROR then activates the USER. This at first glance somewhat complicated path was chosen because ERROR is always ready to run. Therefore in case a USER has been suspended by force by the system because of illegal operations or for whatever reason has gone into an endless loop, it can get it out of trouble and maintain at least emergency operation by restarting the USER.

3.4.1.8 UHFM

UHFM is the support task for mass memory, i.e. an RTOS-UH-system which has a UHFM task, has a driver for mass memory (floppy, harddisk). The name comes from University Hanover File Manager. UHFM is responsible for inputs and outputs through the mass memory and management of the file structure on the mass memory. Since floppies and harddisks also have wait phases (positioning of the write-read heads, motor start-up phases), the UHFM also operates with an interrupt routine. The UHFM can therefore go into the statuses SUSP or SCHD.

Since RTOS-UH enables data management on the mass memory in different ways (at present DOS-compatible or RTOS-UH-own management) the UHFM uses different management routines depending on the required type of management. These management routines can be connected via external jump terminals, see manual. The supported data station is called /FO, /MO, ...

3.4.1.9 VDATN

VDATN is the support task for a memory-internal FIFO-similar organized input/output unit. The data stations are called /VO as destination for outputs and /VI as a source of inputs.

VDATN is used especially for the unsynchronized communication between tasks: a task outputs a message at any time to the VDATN, another task reads this message, again at any time. If the writing task has not made any output, the reading task is suspended by the system (status I/O?) until the desired input can be satisfied. The writing task can continue unhindered in any case. Several outputs of the writing task are passed on to the reading task in the order of their arrival in

VDATN (FIFO structure). VDATN can manage as many of these I/O channels distinctively with filenames as you wish.

If, however, several tasks with different priority are output to one I/O channel, a priority-wise order is superposed on the FIFO principle to such an extent that the outputs of the higher priority task push it at the start of the FIFO in the order of priority. Input requests of higher priority tasks also push out requests of lower priority tasks.

The name VDATN comes from Virtual DATa station.

3.4.1.10 NIL

NIL is the support task for the data station /NIL which represents the ideal data source and sink. Outputs to the station /NIL are always performed successfully; when reading from the station /NIL, a CR is always returned.

3.4.1.11 PPROT

PPORT is the support task for a Centronics interface. Only outputs are permitted via PPORT otherwise the ACIA executions can be taken over entirely. The name is an abbreviation of Printer PORT, the data station name is /PP.

4 First Steps

After all this theory it is time for a little practice. With the knowledge already gained, it is possible to observe the behaviour of the RTOS-UH, but before giving first examples to demonstrate what we have already learned, the fundamentals of system operation and some elementary operating commands need to be explained. However, the computer should be used here, the sooner it is used, the more errors can be made without risk before things get serious.

4.1 System Start

The first hurdle when starting an RTOS-UH system has been taken when the system message appears on the screen. For computers which operate with a connected terminal this may be more difficult, computers with a built-in terminal are usually easier to handle.

4.1.1 Computer Configuration with External Terminal

The terminal must be connected to the main interface of the computer, usually identified by A or 1. For 25-pole Sub-D connections, the pins 2, 3 and 7 (RX, TX and GND) must be connected. Pins 4 and 5 (RTS and CTS) should be bridged at both cable ends at first for the sake of simplicity.

The terminal should be TELEVIDEO-compatible, VT52 is OK as a last resort. The necessary baud rate is specified in the manual, chapter Implementation features. If it is unknown, you can start with 9600 baud. The other transmission parameters are: 8 data bits, 1 stop bit, no parity. However 7 data bits and 2 stop bits are also possible because RTOS-UH only uses 7 data bits. XON/XOFF should be chosen as a protocol (the hardware handshake RTS/CTS is bridged). The terminal should not do any character conversion (CR=CR), on reception of an LF scroll automatically to the bottom line on the screen and when the line length is exceeded, position automatically at the beginning of the next line. The terminal should not carry out a local echo.

When these basic settings have been made, the computer can be started with inserted RTOS-UH EPROMs or bootdisks. The system message should appear about 10 s after starting the computer at the latest or at the end of the booting process.

If nothing happens on the screen, the connections to pin 2 and 3 at one end of the terminal connecting cable should be switched and the process repeated. It does not hurt to press the No scroll or Hold screen button on the terminal twice. An XON (press Control and Q-keys simultaneously ^Q) has never bothered a computer either.

If only strange characters appear on the screen (providing a RTOS-UH system message does not display a collection of strange characters) the baud rate is probably wrong - try 19200 baud.

If none of this helps, this brief introduction is no use. It is then best to leave the computer alone just before finishing time or on Friday afternoon - perhaps it will have another think about things. Otherwise consult a colleague and/or friend with experience in commissioning serial interfaces. The excuse of defective EPROMs or bootdisk is practical but usually not applicable because every RTOS-UH system delivered has already been run.

If the system message appears, the data output works at least. If after simultaneously pressing the keys Control and A a * appears on the screen, the opposite direction also works, if the asterisk does not appear, the cable is probably at fault. Either connections or jumpers are missing or the line is faulty.

4.1.2 Computer Configuration with Integrated Terminal

Computers with an integrated terminal usually only need to be started with inserted EPROMs or bootdisks. To draw level with their colleagues with external terminals the Control and A keys should be pressed simultaneously here too.

4.2 Making Contact

By pressing the keys Control and A at the same time the first contact is made with the computer. The computer has acknowledged the keypress by outputting a * as a prompt and is now waiting for a command to be entered.

Before direct work with the computer can begin, a few conventions: simultaneous pressing of the Control key together with another key, e.g. A, is specified below as ^A. Further, sending a command means entering 'A, the command text and then a CR, either with the RETURN or ENTER key or as ^M.

The access to the operating interpreter in the RTOS-UH is basically different to other operating systems: before entering a command, ^A must be entered.

The reason for this is the blocking of the terminal output for appropriate input. If the operating interpreter is always waiting for inputs, outputs of tasks could not appear on the screen, unless they were output through the duplex channel (system task SOUT). This on the other hand would mean that the input text would be illegible due to the strewn in outputs. To avoid this effect, inputs and outputs are normally only made via the system task ACIA so that no outputs appear on the screen during input processing.

4.2.1 The System Message

The system start message is still visible on the screen. This gives you important information on the current configuration of the operating system. A typical system message looks like this:

```
============================
 > > >  R T O S  -  U H  < <
============================

      Version 2.2  Aug./1989  - Lizenznummer: IEP---TEST---EPAC 68000

Nuc=6.5-E       Imp=2.N         Pbus=1.1        Error=0.6       EdFm=0.5
Vi/Vo=0.5       sh/ext=1.0      Sh/sr=3.2-H     Shell=3.2-G     Hyp=12.3-D
Dev = 3.2       Math=1.B        assign=0.7      r/w=1.2         Loader=4.5-F
Help=1.6c       Prom=2.8        RTC/XPAC=3.1    PWfail=1.2      ADDint=1.0
P=Mini-12.3-E

RESET.
```

The individual system parts are specified here with their own version numbers in addition to header, version and licence number. The system shown here, a version for a 68000 single-board computer, which is used below as a sample computer, contains the following parts:

Nuc	:	operating system nucleus, in version 6.5E
Imp	:	implementation disk with ACIA and clock interrupt in version 2.N
Pbus	:	driver routines for the bus system of the computer, in version 1.1
Error	:	system task ERROR, in version 0.6
EdFm	:	system task EDFM, in version 0.5
Vi/Vo	:	system task VDATN, in version 0.5
sh/ext	:	operation interpreter extensions, in version 1.0
Sh/sr	:	subroutine package for the operation interpreter, in version 3.2-H
Shell	:	operation interpreter with USER, in version 3.2-G
Hyp	:	runtime package for high level language programs, in version 12.3-D
Dev	:	various help routines, in version 3.2
Math	:	floating point arithmetic, in version 1.B
assign	:	output deflection, in version 0.7
r/w	:	binary input/output, in version 1.2
Loader	:	linker and loader, in version 4.5-F
Help	:	operating command HELP, in version 1.6c
Prom	:	utility for creating ROM-resident programs, in version 2.8
RTC/XPAC, Pwfail, ADDint	:	computer-specific
P	:	PEARL-Compiler, in version mini-12.3-E

Tabelle 4-1: Parts of the operating system

Then the RESET message follows, which indicates that RTOS-UH has just performed a cold start. At this point the ABORT message may also appear. In this case RTOS-UH has performed a warm start, i.e. all task activities have been ended properly, a test for correct structure of the memory management carried out and all system components reset. An ABORT is a milder form of system initialization in which loaded tasks and files in the RAM disk are not lost. The triggering of an ABORT may, however lead to a RESET, complete system initialization with loss of all data in the RAM on recognizing errors in the memory manager.

In most computers, RESET and ABORT can be triggered by the hardware with special keys.

4.2.2 The Memory Structure

RTOS-UH is designed as an operating system in which the user can get as much information as possible on the current system status. The memory assignment of the sample computer after being switched on should be considered as the first example: with the command S (^A S CR) the following memory assignment list is obtained:

```
*S
00002086->00002090  MARK
00002090->000020F2  ATSK  Resident  #IDLE
000020F2->00002154  TASK  Resident  #ACIA1
00002154->000021B6  TASK  Resident  #ACIA2
000021B6->00002218  TASK  Resident  #SOUT1
00002218->0000227A  TASK  Resident  #SOUT2
0000227A->000022DC  TASK  Resident  #PPORT
000022DC->0000233E  ATSK  Resident  #ERROR
0000233E->000023A0  TASK  Resident  #EDFMN
000023A0->00002402  TASK  Resident  #VDATN
00002402->00002464  TASK  Resident  #XCMMD
00002464->000024C6  TASK  Resident  #USER1
000024C6->00002528  TASK  Resident  #USER2
00002528->0000258A  TASK  Resident  #NIL
0000258A->0001FFF4  FREE
0001FFF4->00000000  MARK
```

Die Angaben in den einzelnen Spalten haben folgende Bedeutung:

Start					address of the memory segment
	End				of the memory segment
		Type			of the memory segment
			add. info		(optional)
				Name of the owner	

Tabelle 4-2: Memory assignment

In table 3.3 all possible types of memory segments are listed.

Mnemo		Bedeutung
MARK	:	specially marked segment, start or end of the system memory
TASK	:	task identification; the task code can but need not be in this segment. Owner of the segment is the task itself.
ATSK	:	is not a typing error but the task identification of an autostart-capable task..
TWSP	:	task workspace. The memory range for local variable of a task. The owner is this task.
PWSP	:	procedure workspace. Extra memory space required by the task for procedure-local variable. The owner here is also the task.

CWSP : communication workspace. Memory space requested by a task to perform inputs or outputs. The owner is the task. If the task is to be terminated, the memory section is under the ownership of RTOS-UH (name specification (RTOS)) until the end of the I/O operation.

MDLE : module. Data section for permanent data, e.g. for common use by several tasks. A module has its own name and is itself owner of the memory segment.

A special module with the name #NORAM signals a gap in the memory expansion of the computer. The area concerned cannot be accessed.

EDTF : Edit file. This is a segment of the RAM disk. The filename is specified as owner.

FREE : This is free memory space.

PMDL : Prom module. A memory segment formed in the creation of PROM-resident programs.

SMDL : Shell module. The module contains an operating command coded in high level language. Generation by PEARL is possible.

???? : A memory segment is no longer identifiable. The type identification noted in the memory segment was destroyed by illegal operations, program errors or similar.

Tabelle 4-3: Type specifications for memory segments

In the example only memory start, system tasks (partly autostart-capable but all with the Resident property), free memory space and end of memory are available. During further processing other segments will appear. The section before the start of the memory is occupied by RTOS-UH internal management informations and the TWSP of the resident system tasks and is no longer available.

The command S allows a limiting of the output to certain types of segments, see the manual.

4.2.3 Die Taskzustände

An overview of the statuses of all tasks in the system can be obtained by the command L (List Tasks). This looks like this in the sample computer:

```
*L
00002090  +FFF/1  RUN   TWS=00001092 PC=000C14D4  #IDLE
000020F2  -005/1  RUN   TWS=000010EE PC=000C2342  #ACIA1
00002154  -005/1  DORM  TWS=000011D6 PC=00000000  #ACIA2
000021B6  -005/1  DORM  TWS=000012BE PC=00000000  #SOUT1
00002218  -005/1  DORM  TWS=000013A6 PC=00000000  #SOUT2
0000227A  -001/1  DORM  TWS=0000148E PC=00000000  #PPORT
000022DC  -00A/1  SCHD  TWS=0000162A PC=000C2CCC  #ERROR
0000233E  -001/1  DORM  TWS=0000169C PC=00000000  #EDFMN
000023A0  -002/1  DORM  TWS=0000171E PC=00000000  #VDATN
00002402  -002/1  DORM  TWS=000017A0 PC=00000000  #XCMMD
00002464  -007/1  RUN   TWS=00001962 PC=000C447E  #USER1
000024C6  -006/2  DORM  TWS=00001C3E PC=00000000  #USER2
```

```
00002528  -002/1  DORM  TWS=00001F1A PC=00000000  #NIL
```

Die Angaben in den einzelnen Spalten bedeuten:

TID							addr. of the task header
	Prio						Priorität (hex)
		User					USER which performed the activation
			Zustand				Status
				TWSP-Adr.			address of the TWSP
					PC		addr. of the command just executed
						Taskname	

Tabelle 4-4: Output of the task list

The priority is output hexadecimally with restricted numeric range.

The specification of the addresses of the task workspace and the program counter of the tasks enable insiders to make further investigations of the current activity of a task. In particular, it can be observed under some circumstances that a task is ready to run but has not yet been assigned a TWSP and a PC. Such a task is still waiting for assignment of TWSP.

The list obtained here is explained by the interaction of the system task as follows: IDLE is ready to run (as it should be), ACIA1 also because it is just dealing with the task list output. USER1 is ready to run because it must create the task list for output by the ACIA1. All other system tasks have no jobs at present and are therefore DORM. Only the ERROR is planned in for treating error messages.

The order of priority of the system task is also easy to follow: the ERROR has the highest priority (-10), followed by USER1, which is marked as USER at the system interface and has a higher priority than USER2. The ACIAs and SOUTs are in the next priority stage because they are more important for input and output of the USERs in the system than the support tasks for the other data stations.

The task statuses can be observed with the following commands:

L	-	output list of all tasks
LU	-	output user tasks only (list user tasks)
SHOW	-	for specific observation of a single task (SHOW taskname)

Tabelle 4-5: Commands for output of task statuses

The L command can can only be given by options for output according to certain criteria of selected tasks.

4.3 Some Examples

Some examples for the behaviour of tasks under RTOS-UH are given here. Since the creation of programs, especially operation of the editor and treatment of files, is not yet possible with the previous explanations, this chapter is restricted to interventions by operating commands. The aim is to explain the system operation including some basic commands as well as the practical observation of the theoretical knowledge of the task behaviour.

4.3.1 Input of Commands

RTOS-UH requires the operating interpreter to be started with ^A before entering a command. The activated operating interpreter logs in with a * as a prompt and accepts the input of commands.

During command input, the following editing possibilities exist:

- Cursor left, backspace (^H) or DEL (Delete) delete one character to the left
- Cursor right (^L) restores the character at the current position from the previous input
- CR, RETURN or ENTER (^M) and LF (^J) end the input

It is possible to enter commands in small or capital letters and mixed, no differentiation is made between small and capital. Task, module and filenames must be written with the correct small and capital letters, however.

Parameters are separated from the commands by blanks or commas.

The command line may be a maximum 128 characters long.

A command ends with a semicolon ; or a double hyphen --; this end identification can be omitted if only one command is given in a line. Commands separated by semi-colons are processed by the system at the same time if possible (e.g. two files can be copied simultaneously in this way), commands separated by -- are processed chronologically (command chaining). The following commands will not be processed if the respective preceding command was erroneous. In the command P--LOAD (translate PEARL with susequent loading of the program) the load command is only executed if the translation has taken place without any errors.

The following characters are available for checking the output:

- XOFF (^S) stops the output, i.e. the ouput of the prompt is also suppressed after pressing ^A.
- XON (^Q) continues it.

Since XOFF can easily be pressed accidentally instead of ^A, an XON is always useful in unclear situations. Access to the operating interpreter may then be free again.

XOFF and XON are never taken over into the input string.

4.3.2 Generation of Tasks

Tasks are normally generated by high level language or assembler programming or generated automatically by the interpreter to process complex tasks. With the DEFINE command, the operating intepreter can also be forced to direct task generation, however. The

```
DEFINE.X -- SHOW X
```

command has the following effect:

The operating interpreter generates a task with the name X, whose job is to execute the subsequent (--, command chaining) SHOW command. This construction enables the definition of fixed, own operating command sequences.

Internally, this task is generated in the same way as, for example, the subtask generation in the COPY command. The generated subtask is given a prescribed subtask name by the suffix . Name after the operating command. Blanks may appear before the point. If no name is prescribed, the system generates the name from the operating command, followed by a / and a two-digit, consecutive assigned number.

After entering the command DEFINE.X -- SHOW X the system replies with the output belonging to the SHOW command. With L and LU it can be determined that the task X is now available as a user task in the system.

4.3.3 Chronological Planning Ins

With this task X, some examples of activations and planning ins can then be performed. With the command

```
X
```

(short for ACTIVATE X), X is activated and the output appears again. With the command

```
ALL 5 SEC X
```

(short for ALL 5 SEC ACTIVATE X), X is planned in cyclically. X is activated immediately and then in a 5 second rhythm, recognizable from the output which appears. Between the outputs, it can be determined with the LU command that X is planned in in the SCHD status.

Nevertheless X can also be started asynchronously by hand: The input of X as a command only leads to an additional activation and output, but does not change anything in the rhythm of the regular outputs. This poltergeist can only be kept away by

```
PREVENT X,
```

a planning out of task X.

A difficult planning in can be observed with the command

```
AFTER 10 SEC ALL 2 SEC during 10 sec X
```

if nothing happens for 10 seconds, 6 output lines appear and then nothing happens again. With LU, X is found in the DORM status, it has fulfilled its task, has been planned out automatically and is now sleeping peacefully.

If you try to wake up X with the x command, you will soon see the reaction to input errors: the system does not find any task with the name x and complains with

```
>> #USER1:x WRONG COMMAND.
```

Error messages start with >>, followed by the task name of the causer and specify the cause of the error, here, the input of x as an unknown command.

If X is planned in faster (ALL 1.0 SEC X) and the output with XOFF is stopped, the ability of RTOS-UH to buffer task activations can be observed. If XON is given within 5 seconds after XOFF, output lines appear immediately. Their number corresponds to the number of seconds for which the output was blocked. Since X could not end its output, the corresponding number of activations has taken place, but could not be executed because the first activation was not ended (no change from RUN to SCHD). These activations were buffered and are repreated on release of the output. This is easily recognizable particularly in a task which outputs the current time:

DEFINE.Y–CLOCK,

because the time of the actual task run is output here.

If the output is blocked for longer, a line with the output of the first activation appears on releasing the output, followed by an error message (or several)

>> X:INTRPT OVERFLOW (ACT).

and then the outputs of the other, buffered activations. The error message indicates that the activation buffer of task X has overflowed, caused by an interrupt routine (here the clock tick).

4.3.4 Interrupt Planning Ins

RTOS-UH knows 32 different interrupt sources which are named by the designation EV xxxxxxxx (EV = event). xxxxxxxx is a hexadecimally noted bit mask in which the interrupt concerned is noted with a 1, all others with a 0. Interrupt EV 00080000 serves as an example here.

The task can be changed from the DORM status to the SCHD status with

WHEN EV 00080000 activate X

With the simulated triggering of the interrupt by

TRIGGER EV 00080000

nothing at all happens. RTOS-UH blocks the appearance of an interrupt at initialisation. X cannot be started with the TRIGGER EV 00080000 command until after the interrupt has been released with

ENABLE EV 00080000

X is reactivated every time the interrupt appears. The planning in can be deleted again by PREVENT X. However, the activation is already prevented by the interrupt disable

DISABLE EV 00080000

4.3.5 Suspend and Continue

These operations can be observed well in an endless loop of the type

DEFINE.endlos–endlos

with the commands SUSPEND (SU for short) and CONTINUE. By the way the sense of the high priorities for ACIA and USER also becomes clear: although the task endless runs at priority -1 in an endless loop, operating commands can still be given. The computer behaves for the higher priority tasks as if the lower priority stages do not exist.

After a

T endlos

(terminate the endless task), this can be shown better with the aid of the task Low from

Define.Niedrig PRIO 10 -- Show Niedrig; All 2 sec Niedrig

(the task Low is assigned priority 10) and the task endless. If endless is activated whilst Low is running, the outputs of Low disappear first because it no longer has any processor capacity. Low, however, is still activated regularly and therefore error messages appear after approx. 10 sec. which point to overflow of the activation buffer of Low. The effect here is identical with stopping the output in the chapter Chronological planning ins.

5 The I/O System

The I/O system of RTOS-UH differs considerably from that of other operating systems in concept. As an explanation, the basic thought behind the design of the I/O system is demonstrated here. This is followed by a detailed explanation of the functional principle. Although this goes into great detail in part, it is very important to study this description to understand the possibilities which RTOS-UH offers. A brief description of the most important devices and data stations which exist in an RTOS-UH systems concludes this chapter.

5.1 Queues

RTOS-UH decouples the execution of I/O operations from the initiating task. A task which wants to perform an input or output operation first requests space for I/O operations from the operating system, the so-called Communication Workspace (CWSP). Such a memory segment is also known as Communication Element, CE. In this CE, the task enters on which device or which file which I/O operation is to be performed and passes on the filled in CE to the operating system.

The operating system hands this CE into the queue of the device concerned. Ordering takes place in the order of priorities of the initiating tasks, in the event of equal priority in the order of arrival. At the same time it activates the support task belonging to this device if this is not yet active.

The support task then takes a CE from the queue and processes it. After processing, it is returned to the operating system, otherwise the used memory section is automatically declared as free memory space. The support task then terminates if no other CE exists in its wait queue.

If a task which has been initiated by incomplete inputs or outputs is terminated, its input CEs which are still in queues are taken immediately from the queue and declared as free memory space. CEs already being processed by a support task and output CEs are left in the system and completed.

Execution of the I/O operation is thus decoupled from the processing of the initiating task. A task can initiate an output by providing the output data in a CE and passing this CE on to the operating system. If the task has no further interest in the result of the output (errors could occur for example), it can continue immediately after the transfer. The times in which a support task suspends its execution can thus be used by the initiating task.

Similiar behaviour can be observed for inputs: a task can generate an input request, then continue processing and only later show an interest in the read-in data. If the data are not available at this time, however, the task is put in the I/O status and its execution suspended until the end of the I/O operation.

The COPY command uses this facility by peforming ping-pong operation with two CEs: One CE requests input from the copying source, another CE is used for output to the copying destination. Both the input and the output requests can then be processed simultaneously by the system. The generated subtask COPY/xx merely switches the data direction of the CEs after an input and output so that the input received is passed on as an output and the old output CE is then used for input.

5.2 LDNs, Drives and Device Names

RTOS-UH manages its own queue for every I/O device. Operating system internally, every queue is identified by a number, the LDN (Logical Device Number). If a device has several subdivision possibilities (e.g. the device mass memory can manage different drives) another drive number can

or must be added as an exact specification. RTOS-UH manages the individual drives but not in their own queues. Division according to drives must take place in the support task.

To simplify system operation, the operating system provides symbolic device names for many LDN drive combinations. In the input of operating commands, a certain LDN drive combination can be identified simply by specifying the device name. The LDN drive combinations available in an RTOS-UH system can be displayed with the HELP-D command. The following output then appears for the sample computer:

```
*HELP -D
RTOS Devices  (LDN/Drive)
A1:......00/00  A2:......02/00  UL:......02/03  B1:......00/02  B2:......02/02
C1:......00/06  C2:......02/06  D1:......0B/00  D2:......0C/00  PP:......0A/00
ED:......01/00  EDB:.....01/01  VO:......07/00  VI:......08/00  NO:......7F/00
TY:......7E/00  TYB:.....7E/02  TYC:.....7E/06  XC:......09/00  NIL:.....0F/00
```

The system contains:

LDN	Drive	Gerätename	Funktion	Betreuungstask
0	0	**/A1/**	System interface	**#ACIA1**
0	2	**/B1/**	dito, buffered	"
0	6	**/C1/**	dito, buffered and polled	"
1	0	**/ED/**	Ramdisk, ASCII data	**#EDFM**
1	1	**/EDB/**	Ramdisk, binary data	"
2	0	**/A2/**	2nd serial interface	**#ACIA2**
2	2	**/B2/**	dito, buffered	"
2	3	**/UL/**	dito, buffered without LF	"
2	6	**/C2/**	dito, buffered and polled	"
7	0	**/VO/**	virt. data station, output	**#VDATN**
8	0	**/VI/**	virt. data station, input	"
9	0	**/XC/**	operating interpreter	**#XCMMD**
10	0	**/PP/**	Centronics interface	**#PPROT**
11	0	**/D1/**	system interface, duplex	**#SOUT1**
12	0	**/D2/**	2nd serial interface, duplex	**#SOUT2**
15	0	**/NIL/**	ideal data sink/source	**#NIL**

Tabelle 5-1: Assignment LDN/Drive and mnemos of the data stations

The other devices specified /TY, /TYB, /TYC and /NO have special functions. It is not possible to use them directly, their meaning will be explained later.

In the input of operating commands, devices can be specified both by the mnemonic (e.g. /A1) or specification of LDN and Drive (e.g. /LD/0.0 for /A1 or /LD/0.2 for /B1).

5.3 Structure and Use of CEs

In this chapter, the direct handling of CEs is shown to explain the possibilities and capabilities of this I/O concept. Although the examples here are of more interest to assembler programmers (high level language programmers are relieved of this work by a runtime system adapted to the high level language), but should still be read because the behaviour of the runtime support for high level language programs only then becomes comprehensible.

5.3.1 Request a CE

A CE is requested by the operating system with the trap FETCE. The size of the desired data buffer must be entered in the processor register D1 for this. The trap fetches a memory segment of the CWSP type in the required size from the operating system and parametrizes the CE. After returning from the trap, the processor register A1 indicates this memory segment.

The memory segment has the following structure (offset in byte relative to the processor register A1, name according to the RTOS-UH conventions):

Offset	Name	Funktion
0 (0x00)	FORL	Forward pointer of the memory manager, points to the next memory segment
4 (0x04)	BACKL	Backwards pointer of the memory manager, points to the previous memory segment
8 (0x08)	OWNER	number of the initiating USER
9 (0x09)	TYPE	type identification of the memory segment
10 (0x0A)	FORT	forward pointer for chaining of the task affiliation
14 (0x0E)	BACKT	backward pointer for chaining the task affiliation
18 (0x12)	TIDO	TID of the initiating task
22 (0x16)	FORS	forward pointer for queue chaining. Is at 1 if the CE is being processed by a support task.
26 (0x1A)	BACKS	backwards pointer for queue chaining
30 (0x1E)	PRIO	priority of the CEs (for queue)
32 (0x20)	BUADR	pointer ot the data section of the CE
36 (0x24)	RECLEN	for output: number of bytes to be output
		at input: length of the available data buffer
38 (0x26)	STATIO	status information through the CE
39 (0x27)	LDN	number of the wait queue for which the CE is determined
40 (0x28)	MODE	coding of the type of I/O job
42 (0x2A)	DRV	duration of a timeout for processing this CE (only makes sense for suitable support task)

43 (0x2B)	**DRIVE**	number of the drive for which the CE is determined
44 (0x2C)	**FNAME**	filename for which the CE is determined (is not evaluated by all support tasks)
	IOBUF	data buffer. The number of available bytes is determined by calling FETCE. BUADR points to the data section.

Tabelle 5-2: Meaning of the individual fields of a CE

The user can influence the elements after PRIO. The elements up to and including BACKS serve for system-internal management and may not be used.

A CE procured with FETCE was parametrized by the system as follows:

- **PRIO** was occupied with the priority of the requesting task
- **BUADR** points to **IOBUF**
- **STATIO** contains 0
- **FNAME** contains 8 blanks

5.3.2 Filling in a CE

Before a CE can be used, all the data important for input and output must be entered. The meaning of the individual elements and the possible entries are explained below.

Name	Bedeutung des Feldinhalts
PRIO	CEs are entered in a queue in the order of PRIO of the CEs. The priority of the initiating task is used here as a standard (default). This specification can be converted however.
BUADR	BUADR points to the data section of the CE, after request by FETCE, this is to IOBUF, the section after FNAME. Here as many bytes as requested are available. However, BUADR can also be converted. If output data are already available for example, BUADR can be set so that it points to these data. In this case the request of a CE with data field length 0 may be useful.
RECLEN	Indicates the number of data bytes. For outputs, RECLEN must be set to the maximum number of bytes to be output, for inputs to the maximum number of bytes to be read in. The actual number of read or written bytes is additionally influenced by MODE.
STATIO	In the status byte of the CE only the bits (mask $02) and 3 (mask $08) are available. Bit 1 is the so-called scrapping bit (STABRE), if this bit is set, the CE is immediately declared as free memory space at the end of the input or output and not returned to the initiating task (may be useful for output). Bit 3 has no definite function and can be used freely.
LDN	The queue number of the destination device must be entered here.
MODE	Gibt spezifische Anweisungen für die Ausführung des I/O-Auftrags an. Die wesentlichen Punkte werden in der folgenden Tabelle erläutert.
DRV	DRV contains the duration of the timeout for the input or output operation in multiples of a support task-specific basic time. At the end of the timeout, the I/O process is aborted with an error message. This specification is only evaluated by some support tasks. For support tasks which do not know any timeout, DRV should be set to 0.
DRIVE	Must contain the possible drive division of a device.

FNAME In devices which operate file-oriented, the path of the file must be specified. The path specification must end with $FF. Under FNAME there is ample space for every path specification permissible under RTOS-UH (at present a max. 24 characters). The device name may not be specified because it is already coded in LDN and DRIVE. Support tasks which do not manage any files ignore any file specification.

Tabelle 5-3: Konfiguration eines CE's

Die genaue Art des I/O-Auftrags sowie gezielte Anweisung zru Durchführung des Auftrags sind im **MODE**-Feld kodiert. MODE gliedert sich in 2 wesentliche Gruppten:

- Die höchstwertigen 11 Bit sind als Einzelbits kodiert und steuern Details der Auftragsbearbeitung.
- The lowest 5 bits of MODE specify the type of I/O job in the form of a job number..

The most frequently required jobs are:

	Kürzel	Auftrag
0	**READ/WRITE OLD**	read or write an already existing file If an attempt is made to write in an already existing file, this is rejected as faulty. If the file is not yet open, opening takes place with positioning to the file start. There is no further positioning. If the file end is overrun when reading, it is closed automatically depending on the bit MODMSC of MODE (see below). If writing continues at the file end, the file is extended. If the device concerned does not know any file manager, the file specification is ignored.
6	**CLOSE**	Close a file. Any specified data under BUADR are ignored. The file is closed. Should only be performed as an input CE.
7	**READ/WRITE ANY**	Read or write a file If the file concerned does not yet exist, it is re-created and positioned to the start of the file. The other remarks from job no. 0 READ/WRITE OLD apply here accordingly.
8	**REWIND OLD**	Position an already existing file to the file start. If the file does not yet exist, an error message is generated. Should only be used as an input CE.
9	**APPEND**	Outputs are appended to an existing file It is positioned at the file start before this write job. The file is extended by this write job. Only possible as a write job.

21	**REWIND ANY**	Positioning to the file start If the file does not yet exist, it is created first. Should only be performed as a read job.
22	**REWIND NEW**	Generate a new file If the file already exists, an error message is output. Should only be performed as a read job..

Tabelle 5-4: The most important jobs in the CE

The other job numbers are explained in the corresponding support tasks. If a support task receives job numbers which it does not know, it generates an error message.

The other bits in MODE have an individual meaning (starting with the highest significant bit, name according to RTOS-UH convention):Die Einzelbit-Parameter sind in MODE wie folgt abgelegt (Bit-nummer in Motorola-Notation):

15	14	13	12	11	10	9	8	7	6	5
MODMWA	MODMOU	MODMCR	MODMLF	MODMEO	MODMSC	MODMNE	MODBIN	IOCEF	NERR	EXCLU

Tabelle 5-5: Bitzuordnung der MODE-Bits

Sie haben die Bedeutung:

Bez.	Name	Function
MODMWA	Wait-Bit	If this bit is set, the initiating task is set to the I/O? status by the end of processing of the I/O job and not taken into account in the processor assignment. Not useful with set bit 1 of STATIO.
MODMOU	Output-Bit	If this bit is set, the CE is considered as an output CE, if the bit is not set, an input is made.
MODMCR	End on **CR**	If this bit is set, an output or input is ended before reaching RECLEN, if a CR (^M) has been output or read in. The CR is also transferred. If the bit is not set there is no special handling for CR.
MODMLF	End on **LF**	If this bit is set, an output or input is ended before reaching RECLEN if an LF (^J) has been output or read in. The LF is transferred. If the bit is not set there is no special handling for LF.
MODMEO	End on **EOT**	If this bit is set an output or input is ended before reaching RECLEN if an EOT (^D) has been output or read in. The EOT is also transferred. If the bit is not set there is no special handling for EOT. EOT serves as an end of file identification under RTOS-UH.
MODMSC	Suppress-command	If this bit is set, the file managers of RTOS-UH do not automatically close a file when reading past the end of the file and the serial interfaces allow no access to the operating interface through ^A, ^B or ^C.
MODMNE	No echo	Only for serial interfaces: for set bit no automatic echo of the

		input characters is generated.
MODBIN	Binary -Bit	Only for serial interfaces: for set bit binary data transfer is performed, i.e. the top data bit is not obligatorily set to 0 and the XON/XOFF protocol is switched off (only hardware handshake possible via RTS/CTS).
IOCEF		Not for CE parameterization but for restoration of the CE after an input. The end of the file has been reached in a read process.
NERR	No-error-messages	For set bit support tasks generate no error messages on the screen but enter an error number in RECLEN.
EXCLU	Exclusive-Bit	If this bit is set the write or read access should be exclusive, i.e. excluding other tasks. As soon as a task has exclusively used a file, read or write accesses of other tasks are no longer permitted until the task has closed the file.

Tabelle 5-6: The MODE bits of the CE

5.3.3 Execution of the Input or Output

After parameterization, the CE can be given to the operating system with the aid of the trap XIO. The address of the CE must be entered before the trap call in the processor register A1 (output register of FETCE). RTOS-UH then takes all further necessary steps. Modifications on the CE may no longer be made! Depending on the wait bit (MODE, mask $800) the initiating task is then suspended in its execution (status I/O? for set wait bit) or can run on unhindered (wait bit 0).

If the STABRE bit (bit 1 of STATIO) is set, no further operations are permissible with the CE from now on. The CE is automatically destroyed by the operating system after input or output, i.e. the CWSP is declared to free space.

5.3.4 Evaluation and Enable of a CE

If the wait bit of a CE is not set, the initiating task can wait for the end of the processing of a CE with the trap IOWA. The address of the CE must be entered in the processor register A1 before the trap call (output register of FETCE). The initiating task is put in the I/O? status if the processing of the CE is not yet ended.. After processing a CE by a support task, the following information can be derived from the CE:

- Input CE

  BUADR points to the read-in characters.

  RECLEN characters have been read in.

  If the end of the file was reached when reading in (can be reached exactly with End-on EOT) IOCEF is set in MODE..

- Input and output CE

  If RECLEN is 0 or negative, the I/O operation has not been performed successfully. In the event of an error, RECLEN contains an error number with set NERR in MODE (see the manual page C-IV-6 ff) with set highest significant bit, with NERR not set, an error message has already been output by the support task and RECLEN contains 0.

The CE can now be re-used or returned to the system with the trap RELCE. After transmitting the trap RELCE (A1 must point to the CE!), the memory section of the CE is declared as free space again and may not be used further.

5.4 Example

The parameterization and behaviour of the CEs can be observed easily on the sample computer.. With the command

```
*CP /A2/>/ED/MIST
```

a copy subtask (CP: short form of COPY) is generated which is copied from the second serial interface into the file MIST of the Ramdisk. Since no data source is connected to the interface, there is no data transfer and the generated CEs can be saved in the memory with the command

```
*S-C
0001FCF4->0001FDBA CWSP CP/00 /ED/MIST
0001FDBA->0001FE9E CWSP CP/00 /A2/-
```

im Speicher dingfest machen. S-C is the command S with the option -C which allows only CWSP to be displayed.

The two CEs can be observed more closely with the DM command (Dump Memory). The first CE is given at

```
*DM 1FCF4 1FDBA
0001FCF4: 0001 FDBA 0000 358A 0004 0001 FDBA 0001 ......5.........
0001FD04: FE9E 0001 FF44 0000 0000 0000 0C20 0014 .....D.......  ..
0001FD14: 0001 FD38 0002 0001 8015 0000 4D49 5354 ...8........MIST
0001FD24: FF0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D ................
0001FD34: 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D ................
0001FD44: 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D ................
0001FD54: 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D ................
0001FD64: 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D ................
0001FD74: 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D ................
0001FD84: 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D ................
0001FD94: 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D ................
0001FDA4: 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D 0D0D ................
0001FDB4: 0D0D 0D0D 0001 0001 FE9E 0001 FCF4 0004 ................
```

The output of the DM command consists in one line of the start address of the memory extract and the contents of the memory section of the start address up to and including start address+15 once in hexadecimal and once in ASCII notation.

The second CE gives

```
*DM 1FDBA 1FE9E
0001FDBA: 0001 FE9E 0001 FCF4 0004 0001 FE9E 0001 ................
0001FDCA: FCF4 0001 FF44 0000 0001 0000 0C2A 0014 .....D.......*..
```

```
0001FDDA: 0001 FDFE 0002 0002 C007 0000 2DFF FF49 ............-..I
0001FDEA: FF00 0000 0000 0C16 000F 000D 3BCE 0016 ............;...
0001FDFA: 8200 0A3D 0A3D 494C 4520 5741 5320 494E ...=.=ILE WAS IN
0001FE0A: 5354 414C 4C45 4420 4259 2045 442E 0D0A STALLED BY ED...
0001FE1A: 0D0A 1B44 1B44 1B44 1B44 1B44 1B44 1B44 ...D.D.D.D.D.D.D
0001FE2A: 1B44 1B44 1B44 1B44 1B44 1B44 1B44 1B44 .D.D.D.D.D.D.D.D
0001FE3A: 1B44 1B44 1B44 1B44 1B44 0A0A 0A0A 0A0A .D.D.D.D.D......
0001FE4A: 0A0A 0A0A 0A0D 0000 FF00 0000 FF00 0000 ................
0001FE5A: FF00 0000 FB00 0000 FF00 0000 FF00 0000 ................
0001FE6A: FF00 0000 AE00 0000 FF00 0000 FF00 0000 ................
0001FE7A: FF00 0000 DF00 00FF FFFF 00FF FFFF 00FF ................
0001FE8A: FFFF 00FF A0FF 00FF FFFF 00FF FFFF 00FF ................
0001FE9A: FFFF 0001 0001 FF44 0001 FDBA 0002 0001 .......D........
```

A transfer of the values to the structure of a CE gives the following contents:

CE-Element	CE1	Bedeutung	CE2	Bedeutung
FORL	1FDBA	The memory chain can	1FE9E	
BACKL	358A	be checked with S	1FCF4	
OWNER	00	is always 1 less than the specification x in #USERx	00	
TYPE	04	corresponds to CWSP	04	
FORT	1FDBA	used RTOS-UH-internally	1FE9E	
BACKT	1FE9E	"	1FCF4	
TIDO	1FF44	recognizable as TID with LU	1FF44	
FORS	0	not being processed	1	being processed by support task
BACKS	C20	system-internally	C2A	
PRIO	14	corresponds to task priority	14	
BUADR	1FD38	points to respective IOBUF	1FDFE	
RECLEN	2	insignificant at present because CE not being used at the moment	2	input not
STATIO	0		0	
LDN	1	**/ED/**	2	/A2/
MODE	8015	REWIND ANY, input with WAIT (from previous use when creating the MIST file)	C007	READ/WRITE ANY, input with WAIT and End on CR

DRV	0	no timeout	0	
DRIVE	0	Drive 0	0	
FNAME	MIST	specified as for command	-	standard if not spec.
IOBUF	- empty -	((not yet used)		LF u and. =

Tabelle 5-7 : Contents of the example CE

The unused parts of the CE, especially the rest of FNAME and the complete IOBUF, may exhibit any values from the previous memory use.

With the now known contents of a CE, an own study of the status of I/O operations can be made in unclear situations.

5.5 Devices and Data Stations

Only short notes on the most important devices in an RTOS-UH system are given here. Detailed descriptions follow in a later chapter, only the serial interfaces are described in detail here.

5.5.1 Device Parameters

Every device available in an RTOS-UH system provides a short parameter block for describing its capabilities. This can be observed with the command

DD device names

DD means Dump Device.

The command DD outputs the address of the parameter block as well as 16 bits which characterize the individual device functions. The individual bits have the following meaning:

Mask	Meaning (for set bit)
0x8000	device allows positioning to file start
0x4000	a file must be opened (under RTOS-UH only possible as REWIND) before using, and closed after using.
0x2000	If CR ends a line, it should be replaced by CR+LF (only for outputs to this devi ce).
0x1000	The device is capable of dialog (serial interface, terminal)
0x0800	For serial interfaces: no echo should be generated.
0x0400	deletion of files is possible (commands RM and ERASE)
0x0200	data output is possible through the device.
0x0100	data input is possible through the device.
0x0080	The device can supply a list of available files (the command DIR is permitted)
0x0040	The device can be formatted (the command FORM is accepted).
0x0020	The device accepts the command CF..
0x0010	The device knows subdirectories (the commands MKDIR and RMDIR are permit ted).
0x0008	Free positioning within files on the device.

0x0004 Reserved.

0x0002 For serial interfaces: The terminal does not jump automatically to the next line when it exceeds the last column of a line (no auto wrap)

0x0001 for serial interfaces: cursor control should take place for VT52 via ESC sequences (standard is TELEVIDEO compatibility)

Tabelle 5-8: Device parameters - the individual bits

After the system start these parameter blocks are supplied with standard values. These values can be changed for setting other operating modes within the scope of the possibilities of the individual devices with the command SD

SD ***/Gerätename/ Parameter***

These parameter bits do not control the behaviour of the individual devices (support tasks) but only serve as information for other programs on the way the device wants to be treated. The individual support tasks base their behaviour on the control information received with the CE alone.

Operating system programs poll these parameters and base their behaviour on this, a COPY, for example, makes a REWIND on a Ramdisk file before use and a CLOSE after use, not for a serial interface however, according to the parameter set for the respective device.

5.5.2 Serial Interfaces /Ax, /Bx, /Cx, /Dx

The serial interfaces of an RTOS-UH computer are called /Ax, /Bx, /Cx and /Dx. Support tasks for an interface are #ACIAx (#SCCx, #RS232) and #SOUTx, the identification x is counted hexadecimally for the number of available interfaces. The letter A, B, C or D identifies different operating modes of the same interface.

A serial interface /Ax has for example the device parameters 3300, i.e. the device is a dialog-capable data terminal ($1000) and allows output ($0200) and input ($0100). It is also desired that system programs extend ending CRs with an LF for outputs through this device ($2000). The echo of input characters should take place. The general function of a serial interface under RTOS-UH is as follows:

5.5.2.1 Output

The data contained in the CE are output. If the serial interface receives an XOFF during the output or an XOFF was received before starting the output, the output is stopped until receiving an XON. Blocking by XOFF can be cancelled by the operating command

SB ***Gerätename Baudrate***

(Set Baudrate). If the RTOS-UH implementation supports the RTS/CTS hardware protocol, the CTS status Low, i.e. line level <2.2 V, is handled identically to a received XOFF.

Outputs via /Dx can take place while /Ax, /Bx or /Cx inputs are active, i.e. input CEs are being processed.

5.5.2.2 Input

Inputs can only be made via the interfaces with the identifications A, B and C. The responsible support task is #ACIAx. The behaviour of the interface in the input depends on its identification (from the DRIVE of the CE) and the MODE of the CE. The identification enables the following differentiations:

- Identification A - unbuffered operation

  The CE is filled up with the characters which arrive through the support task when processing of the CE begins. Processing of the CE is ended after arrival of RECLEN characters or the reception of one of the characters CR, LF or EOT (depending on MODMCR, MODMLF and MODMEO). No XOFF is sent. The reception buffer for the buffered operation is deleted.

- Identification B - buffered operation

  The CE is filled first from a reception buffer (length implementation-dependent, usually 32 characters) which is also used by the interrupt routine in the absence of a CE. If the reception buffer is empty an XON is sent to start the transmitter. Other characters are taken directly from the input data flow. Processing of the CE is ended after entering RECLEN characters or receiving one of the characters CR, LF or EOT (depending on MODMCR, MODMLF and MODMEO)..

- Identification C - buffered operation, only empty buffer

  The behaviour is the same as for identification B except that characters from the input data flow are not waited for. The CE is first filled from the reception buffer. If RECLEN or an abort condition is satisfied, the CE is returned, otherwise the CE is filled up to the RECLEN characters with 0 and returned. Arrival of characters through the interface is not waited for.

The duplex channel addressable under the mnemo D warrants no special mention at this point.

5.5.2.3 Rest Status

The behaviour of the serial interface in the rest status, i.e. if no CE is being processed, depends on the operating mode of the previous use, i..e. after the MODE and DRIVE of the previous CE, also of a previous output CE. The outputs should therefore also be made with the identification desired for inputs.

All incoming characters are transferred to a reception buffer. When this buffer is full, the last received character is overwritten..

If the interface was used previously in buffered operation, an XOFF is sent starting at half buffer filling for every received character to halt the transmitter.

If the interface was used in binary operation, no XOFF is sent; if the interface supports the RTS/CTS protocol (implementation-dependent), RTS is set to LOW (line level <0.8 V) instead.

5.5.2.4 The Effect of MODE

The meaning of MODMWA, MODMOU and NERR is self-explanatory. MODMCR, MODMLF and MODMEO act in input and output. IOCEF is not set, EXCLU not taken into account. Only 7 makes any sense as a job number.

- **MODMSC**

  If this bit is not set, access to the operating interpreter can be requested by entering ^A or ^B (activation of the task #USERx). If an input CE is just being processed, these characters are not transferred to the CE, i.e. the input is completed regularly. The input CE generated by the USER takes its parameterization from the type of request: if ^A was received, the command input is requested in A mode, with ^B, however, in B-mode. This makes it possible to assess the characters received before the input CE of the operating interpreter arrives as part of a command.

- **MODMNE**

If MODMNE is not set, entered characters are output simultaneously when transferring to a CE (see also MODBIN). The echo takes place for example in B mode not with transfer to the reception buffer (i.e. in the rest status) but not until after transfer to a CE.I In B mode the received characters are output without transformation, in the A mode, control characters (characters less than $20, the blank) are output as " ".

- **MODBIN**

  If MODBIN is not set, the XON/XOFF protocol is switched on. In addition, the top bit is deleted for received characters. If MODBIN is switched on, the XON/XOFF protocol is switched off; operating system-internal implementations which support the RTS/CTS protocol use this. In addition all received characters are transferred to the receive buffer or the CE. The Edit function (see below) of the interfaces is switched off, an echo of incoming characters does not take place.

5.5.2.5 Edit Function

In the input, it is possible to change the input text with the following special characters:

- Cursor left, backspace (^H) and Delete position one character back in the CE if possible, i.e. delete the character which has just been received. As an echo, ^H, a blank and ^H again are output.
- Cursor right (^L) leaves the character in the CE unchanged at the current position. This character is output as an echo to ^L.

These special characters are not transferred to the CE. The Edit function is in binary mode (MODBIN) and switched off for set MODMSC.

5.5.3 The Ramdisk - /ED, /EDB

In almost every RTOS-UH system, the Ramdisk exists as a file-oriented memory medium. The files of the Ramdisk are addressed by a device and path specification, e.g. /ED/test addresses the test file. In the path specification, directories and subdirectories can also be used, e.g. /ED/Heinz/test addresses the test file in the Heinz directory. Directories and subdirectories do not need to be created explicitly. The use of Ramdisk is relatively easy with this information:

```
*COPY /A1/>/ED/test
```

starts a copying process from the system interface in A mode to the test file. The COPY/xx subtask then examines the device parameters of the devices concerned first and determines that a file of the Ramdisk needs to be opened before use (as in CE1 of the above example). It also determines that /A1 is a dialog-capable data terminal which wants to be processed with echo and therefore outputs a = for identifying the readiness for input (to simplify operation). Then it requests an input from /A1 as in CE2 of the example. In the IOBUF of CE2 the rest of the outputs of the equal sign can be observed. With the inputs

```
=MODULE TEST;
=MODEND
=@
```

the input lines are copied into the test file. Every single entered character is echoed on the screen; the individual lines are terminated with CR in the input. The character echoed as is entered as ^D (ASCII EOT, End of Text) and identifies the end of file.. The subtask recognizes the (entering) file end, closes the test file and ends its activity with the message

```
>> COPY/xx: (TERMI).
```

The entered text is now contained in the file and can for example be output on the screen with

```
TYPE /ED/test
```

With L or LU it can also be determined that the COPY and TYPE subtask are no longer available in the system at the end of their task. With S it can be determined that a memory segment of the EDFT type now exists and contains the file test.. An overview of the existing files can be obtained with the command

```
DIR /ED/
```

If

```
>> --??--: (TERMI).
```

should appear as a terminating message of a COPY or TYPE command this is not a sign of damage to the operating system but points to the fact that the memory segment containing the subtask has already been assigned and used otherwise before the end message was output. The name of the subtask can no longer be identified in this case.

It is possible to delete the file with the commands

```
RM /ED/test
```

(engl. ReMove, dt. Entfernen) oder

```
ERASE /ED/test
```

Files in the Ramdisk are stored in compressed, line-oriented form. Because of the compression algorithm only 7-bit ASCII data may be contained in files in the /ED. If other data are also to be stored in Ramdisk files, the device /EDB is available for this. This operates without data compression; the handling is otherwise identical with /ED.

5.5.4 /VI/, /VO/

The virtual data station provides data channels under different file names. The specification of directories and subdirectories is also possible, these do not need to be set up explicitly however.

/VI only allows inputs, i.e. only read access is possible to /VI. With

```
*CP /VI/kanal1 > /ED/test2
```

a copying process is started which reads out of the input channel with the filenames channel1 and the read characters copied to the Ramdisk, file test2.

/VO on the other hand can only be used as an output device. A virtual data channel is identified by the same file name for /VI and /VO. With

```
*COPY /A1/> /VO/kanal1
```

a copying process from the system interface (recognizable from the output of the equal sign as an input prompt) to data channel channel1 is started. Since the data from this data channel are trans-

ferred to the file test2 of the Ramdisk with the previous COPY command, this represents only a complicated alternative of the command COPY /A1/>/ED/test2.

With the inputs

=MODULE TEST2;

=MODEND;

=@

text can then be input in the file test2. The end of input by ^D (is displayed as on the screen) both COPY/xx subtasks are terminated, recognizable from the two TERMI messages.

Single input and output channels can be emptied with RM /VI/channel name or RM/VO/channel name. The CEs contained are returned to the initiating tasks in this case. In the CEs it is noted first that the input or output procedure could not be performed successfully.

7 Figures

8 Tables

IEP GmbH • Am Pferdemarkt 9c • D-30853 Langenhagen • Tel.: +49 (511) 70832-0 • Fax: +49 (511) 70832-99 • E-Mail: info@iep.de
Web: http://www.iep.de