RTOS-UH
-
Ein Überblick

Andreas Hadler

1 Inhaltsverzeichnis

2 Echtzeit-Systeme

RTOS-UH ist ein Echtzeit-Multitasking-Betriebssystem und als solches schon vom Konzept her anders als die meisten üblichen Betriebssysteme. Das Betriebssystem versucht, den speziellen Anforderungen der Meß-, Steuer- und Regelungstechnik mit zwei speziellen Strategien gerecht zu werden. Schon in der Bezeichnung Echtzeit-Multitasking-Betriebssystem wird herausgestellt: das Betriebssystem ist echtzeitfähig und kann mit mehreren Tasks arbeiten.

In einer etwas laxen Definition kann man sagen:

Echtzeitfähig wird ein System dann genannt, wenn es jederzeit in ausreichender Schnelligkeit auf äußere Ereignisse reagieren kann.

Dies ist zwar eine Definition, die jedem Puristen den Spaß an Fachdiskussionen nimmt, aber als pragmatischer Ansatz trifft. Normalerweise interessiert nicht, ob das System den Richtlinien des *hard realtime computing*, folgt, also harten Echtzeitanforderungen genügt, oder ein *real realtime system*, ein echtes Echtzeitsystem, ist. Üblicherweise vielmehr gilt der pragmatischere Ansatz: "Ist das System für meine Anforderungen schnell genug? Kann ich mit dem System meine Maschine/Anlage o.ä. steuern? Kann ich schnell genug auf Grenzschalter, Eingaben etc. reagieren?". In den Anfangszeiten des Einsatzes von Computern in der Meß-, Steuer- und Regelungstechnik, vor der Gewöhnung an eine spezifische Fachterminologie, wurde der viel einprägsamere Begriff der *schritthaltenden Datenverarbeitung* geprägt. Hier ist klar erkennbar, was gemeint ist: der Rechner kann mit seiner Umwelt Schritt halten, d.h. so schnell auf äußere Vorgäng reagieren, daß er nichts versäumt.

2.1 Lösungsansätze

Zur Erfüllung dieser Anforderungen an Reaktions- und Rechengeschwindigkeit sind im wesentlichen zwei unterschiedliche Ansätze verfolgbar:

- "brute force"

  Mit Einsatz reiner Rechengewalt kann man ein allen Echtzeitanforderungen genügendes Verhalten erzwingen. Ist ein Rechner nicht schnell genug zur Lösung eines Problems, so wird die nächstschnellere Variante eingesetzt. Reicht das nicht, so werden halt 2, 4, oder in gigantischen Ansätzen bis zu 65536 Rechner eingesetzt, und das Problem so umdefiniert, das es in unabhängigen Teilen lösbar wird.

  Durch schlichten Materialeinsatz läßt sich somit jedes Problem lösen, das eine prinzipielle Lösbarkeit besitzt, es sei denn, die Kommunikation und Synchronisation dieser ggf. zahllosen Rechner erfordert überproportionale Rechenkapazität und läßt somit einen Punkt erkennen, an dem eine Erhöhung der Rechnerzahl sogar eine Verringerung der Reaktionsfähigkeit herbeiführt.

- Algorithmische Intelligenz

  Die intellektuell und, wenn man will, auch ästhetisch befriedigendere Lösung besteht in der Entwicklung geeigneter und effizienter Algorithmen. Als Beispiel mag die Multiplikation zweier Zahlen dienen: die Multiplikation "7*3" läßt sich zurückführen auf die Addition "3+3+3+3+3+3+3" (mit ein wenig Überlegung ist natürlich "7+7+7" die bessere Lösung). Aber erst bei "17839*15757" wird klar, welchen Vorteil die Logarithmierung mit anschließender einzelner Addition bietet: statt 15757 Additionen durchzuführen, kann das Problem durch die Bildung zweier Logarithmen und deren Addition gelöst werden.

Beide Lösungswege können natürlich zum Erfolg führen, der eine mit höherem Material- und der andere mit höherem Denkaufwand.

2.2 Reaktionsgeschwindigkeit und Interrupts

Echtzeit-Systeme versuchen hierbei, dem Anwender einen Teil des Denkaufwandes abzunehmen. Das Hauptproblem bei der Realisierung von Steuerungen und Regelung ist die Gewährleistungen einer zur sicheren Beherrschung der Anlage ausreichenden Reaktionsgeschwindigkeit. Neben der Berechnung von häufig recht rechenzeitaufwendigen Regelungs-Algorithmen müssen Grenzschalter überwacht, Grenzwerte für Druck, Temperatur etc. überprüft und auf deren Erreichen in vorgeschriebener Zeit reagiert werden. Die erforderlichen Reaktionszeiten sind im Vergleich zum Rechenbedarf der Regelung häufig sehr kurz, so daß bei der Programmierung der Regelung sehr viel Denkaufwand auf den Erhalt der Reaktionsgeschwindigkeit während der Abarbeitung aufwendiger Regelungsverfahren verwandt werden muß. Echtzeitsysteme bieten dem Programmierer hierbei Mechanismen zur einfachen Realisierung hoher Reaktionsgeschwindigkeiten an.

Die Gewährleistung dieser hohen Reaktionsgeschwindigkeit wird in den meisten Fällen durch eine geschickte und effiziente Behandlung von Programmunterbrechungssignalen, den Interrupts, realisiert. Interrupts sind Signale, die von außen an die Zentraleinheit eines Rechners gelangen. Wird eines dieser Signale aktiv, so unterbricht die Zentraleinheit ihre aktuelle Arbeit und verzweigt zu einem speziellen Programmstück, dem Unterbrechungsbehandler oder Interrupt-Handler. Echtzeitsysteme zeichnen sich nun dadurch aus, daß die Programmierung dieser Unterbrechungs-Behandlung besonders effizient und nach einem standardisierten Verfahren erfolgt. Im Gegensatz zu üblichen Systemen kann ein Programmierer eines Echtzeitsystems sich darauf verlassen, das die Programmunterbrechung nahezu verzögerungsfrei bearbeitet wird, und er ist der Mühe enthoben, selbst Mechanismen zur Unterbrechung und Fortführung laufender Programme zu realisieren. Der Programmierer kann seine beiden Probleme, den Regelungsalgorithmus und die Reaktion auf Grenzschalter etc., nahezu unabhängig voneinander programmieren und darauf vertrauen, daß das Echtzeit-System für eine geeignete und schnelle Umschaltung zwischen beiden Programmteilen sorgt. Natürlich muß der Programmierer hierzu dem System mitteilen, welches Programmteil welchem Interrupt zugeordnet ist; dieser Aufwand ist jedoch erheblich geringer als die Programmierung des Interrupt-Handlers selbst.

2.3 Multitasking

Das zweite Element zur vereinfachten Programmierung von Problemen aus der Meß-, Steuer- und Regelungstechnik ist das sogenannten Multi-Tasking. Eine Task (engl. Aufgabe) ist ein Programm, das eigenständig lauffähig ist. In einem Rechner, der Multi-Tasking beherscht, sind mehrere dieser Tasks gleichzeitig vorhanden, und das Betriebssystem übernimmt die Zuteilung der Rechenkapazität auf die einzelnen Tasks. Die einfachste Zuteilungsstrategie ist unter dem Begriff *time-sharing* bekannt: jede Task erhält für eine feste Zeitdauer, eine sogenannte Zeitscheibe, üblicherweise zwischen ca. 20 Millisekunden und 2 Sekunden, Rechenkapazität zugeteilt und kann dann arbeiten. Nach Ablauf einer Zeitscheibe führt das Betriebssystem einen Task-Wechsel durch und bringt eine andere Task zur Ausführung.

Der ursprüngliche Gedanke, der zur Entwicklung des Multitasking geführt hat, war die Nutzung eines Rechners durch mehrere Anwender, die alle gleichzeitig an einem Rechner arbeiten können und ihre Programme quasi gleichzeitig ausführen sollten. Diese sogenannten Mehrnutzer-Systeme, engl. Multi-User-Systems, ermöglichen die gleichzeitige Benutzung eines Rechners durch mehrere Anwender und ersparen so die Anschaffung individueller Rechner für die einzelnen Anwender. Als Nebeneffekt ist damit natürlich auch verbunden, daß ein einzelner Anwender zu

Zeiten, in denen wenige andere Awender an diesem Zentralrechner arbeiten, wesentlich mehr Rechenkapazität zur Verfügung hat, als wenn er nur seinen Arbeitsplatzrechner hätte.

Auch für die Programmierung von Anlagensteuerungen bietet das Multi-Tasking Vorteile: die oben beschriebenen Aufteilung einer Steuerungsaufgabe in aufwendige Algorithmen und reaktionskritische Grenzwertüberprüfungen kann effizient und elegant durch Ausnutzen des Multi-Tasking erfolgen. Beide Problemlösungen werden in getrennten Tasks formuliert und unabhängig voneinander programmiert. Das Betriebssystem übernimmt die Umschaltung zwischen den einzelnen Tasks.

Zur Sicherstellung einer ausreichend schnellen Reaktion des nun aus mehreren Tasks bestehenden Programmpaketes auf äußerer Ereignisse muß der Task-Umschalt-Mechanismus des Betriebssystems entsprechend den speziellen Anforderungen der Echtzeit-Programmierung angelegt werden. Hierzu bietet sich die Ausnutzung der Interrupts, der von außen angelegten Unterbrechungssignale, an. Man erhält hierdurch gegenüber der reinen Zeitscheibensteuerung gleich zwei Vorteile: zum einen finden Taskwechsel nur dann statt, wenn sie auch tatsächlich erforderlich sind, und zum andern laufen Tasks, die aufwendige Algorithmen bearbeiten und nicht auf Interrupts reagieren wollen, ungestört.

2.4 Das Angebot von RTOS-UH

RTOS-UH versucht, mit einer möglichst schnellen und einfach nutzbaren Reaktion auf Interrupts sowie einem leicht überschaubaren Konzept des Multitaskings die Werkzeuge zur Verfügung zu stellen, die einem Programmierer die Lösung der speziellen Probleme der Meß-, Steuer- und Regelungstechnik vereinfacht. Das Multitasking, das unter anderen Systemen nur auf Systemebene zur Verfügung steht, wird unter RTOS-UH jedem Programmierer direkt zugänglich gemacht und kann auf eine sehr übersichtliche Weise auch zur Lösung kleinerer Aufgaben genutzt werden. RTOS-UH unterstützt, insbesondere auch mit der Programmiersprache PEARL (Process and Experiment Automation Realtime Language), die Aufteilung einer Problemlösung in überschaubare Einzelprogramme (Tasks), die unabhängig voneinander programmiert, getestet und ausgeführt werden können. Mit der Möglichkeit, einzelnen Tasks Prioritäten zuzuordnen, wird für einen problemangepaßten Ablauf des gesamten Programmsystems gesorgt. Das Betriebssystem teilt den einzelnen Tasks Rechenkapazität in der Reihenfolge ihrer Priorität zu. Zusammen mit der sehr schnellen Reaktion auf Interrupts und der Möglichkeit, Taskwechsel als Reaktion auf Interrupts einzuplanen und durchzuführen, ergibt sich ein sehr leistungsfähiges Werkzeug.

Da der Einsatz dieses Systems vom Programmierer einen zunächst ungewohnten Arbeitsansatz erfordert - die Aufteilung eines Problems in kleinere, unabhängige Teile, die Definition der einzelnen Tasks mit ihren Prioritäten, die Erkennung ggf. erforderlicher Synchronisation der einzelnen Tasks, die Zuordnung einzelner Programmteile zu einzelnen Interrupts, usw. -, ist die Beschäftigung mit den grundlegenden Funktionsprinzipien des RTOS-UH zur erfolgreichen und effizienten Nutzung des Systems sehr sinnvoll.

Ausgehend von einer Beschreibung der verschiedenen Taskeigenschaften und -zustände soll im Folgenden zunächst die Bedeutung der schon fest im System vorhandenen Tasks sowie deren Zusammenspiel mit Interruptroutinen erläutert werden. Nachdem damit die prinzipielle Funktion des RTOS-UH bekannt ist, kann das im Vergleich zu üblichen Betriebssystemen sehr anders geartete I/O-System und seine Arbeitsweise beschrieben werden. Hierauf aufbauend wird anschliessend die Bedienung des RTOS-UH-eigenen Editors erläutert, um in einigen Programmbeispielen Erläuterungen zum bisher erklärten zu geben. Eine kurze Beschreibung der Programmiersprache PEARL wird diesen Überblick über RTOS-UH abschließen.

Die Art der Beschreibung wird sehr gemischt sein: kursorische Überblicke und Detail-Erläuterungen werden bunt gewürfelt erscheinen. Zumindest das Studium der jeweils ersten Sätze

eines Kapitels ist nötig, um einen groben Überblick über das Betriebssystem und seine Besonderheiten zu erhalten. Detaillierte Erklärungen können zunächst gefahrlos überlesen werden, sind aber für ein effektives Arbeiten mit dem Betriebssystem sehr wichtig.

Dieser Text soll nicht das Studium des Systemhandbuches erübrigen, und wird daher an einigen Stellen nicht in korrekter Ausführlichkeit Anweisungen zum Handeln geben, er hilft aber sicherlich zum Verständnis der häufig eher abstrakten Erläuterungen des Handbuches.

In diesem Überblick werden Theorie und Praxis getrennt behandelt, da der Autor der Ansicht ist, tatsächliches Arbeiten am Rechner setzt zumindest einige Kenntnisse über die Funktion des Betriebssystems voraus. Zusätzlich soll so vermieden werden, daß das Arbeiten am Rechner zu viele aus der Theorie noch nicht bekannte Phänomene aufzeigt, über die zunächst mit viel Eleganz hinweggesehen werden müßte.

3 RTOS-UH - Das Betriebssystem

RTOS-UH soll als Betriebssystem die Resourcen eines Rechners (Speicher, Prozessor, I/O-Schnittstellen) verwalten und Anwenderprogrammen Dienstleistungen zur einfachen Nutzung dieser Resourcen zur Verfügung stellen. Um dies Aufgabe effizient zu erfüllen, bedient sich RTOS-UH dreierlei Hilfsmittel: es besteht aus

- einem Betriebssystemkern, dem sog. Nukleus, der über Systemaufrufe, Traps, die Manipulation von Tasks kontrolliert und den Systemspeicher verwaltet,
- aus Interrupt-Routinen, die grundlegende I/O-Operationen abwickeln, und
- aus Systemtasks, die komplexere Verwaltungsaufgaben erledigen.

Da der Begriff einer Task fundamental für das Verständnis von RTOS-UH ist, soll er zunächst genauer ausgeführt werden. Anschliessend wird erläutert, wieso Interrupt-Routinen Bestandteil des Betriebssystems sind und wie diese mit den Systemtasks zusammenarbeiten.

3.1 Tasks und Tasking

Grundlegend für RTOS-UH ist der Begriff *Task*, der ein eigenständig ablauffähiges Programm kennzeichnet. Unter RTOS-UH können diese Tasks verschiedene Eigenschaften haben und sich in verschiedenen Zuständen befinden. Das Betriebssystem verteilt die verfügbare Rechenkapazität auf die einzelnen Tasks (Multi-Tasking), und zwar derart, das der Wechsel von einer Task zur anderen jederzeit möglich ist (Echtzeit-Fähigkeit). Bei Mehrprozessor-Rechnern können durchaus mehrere Tasks gleichzeitig aktiv sein, bei einem Einprozessor-System kann zu einem Zeitpunkt natürlich nur eine Task exekutiert werden. Im folgenden Text wird stets von einem Einprozessor-System ausgegangen, die analoge Übertragung auf ein Mehrprozessor-System ist relativ einfach.

3.1.1 Task-Eigenschaften

Unter einer Task versteht RTOS-UH ein eigenständig arbeitendes Programm. Um unter RTOS-UH als Task anerkannt zu werden, muß dieses Programm einige Eigenschaften haben. Diese Eigenschaften muß die jeweilige Task gemäß den Spielregeln von RTOS-UH in einem Verwaltungsblock, dem *Taskkopf*, notieren.

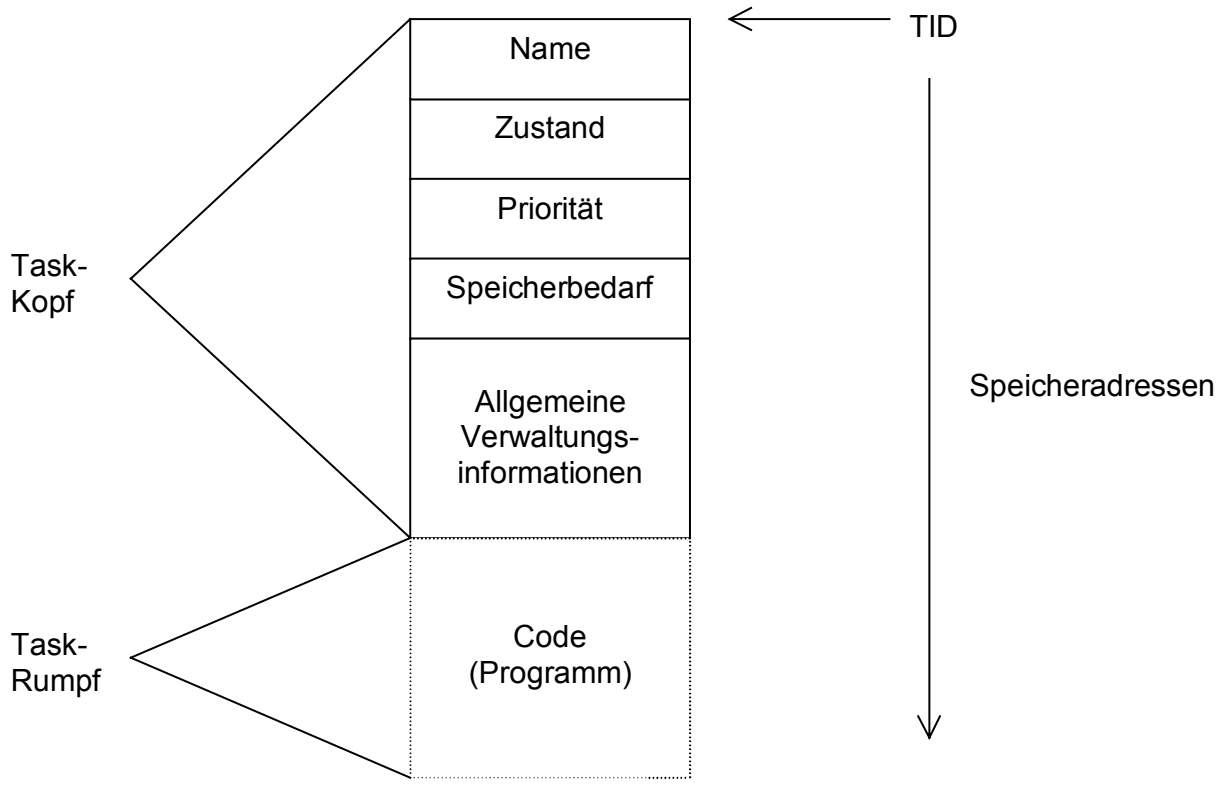

Abbildung 3-1: Aufbau einer Task

Hochsprachen-Programmierer brauchen sich um die Generierung dieser Taskköpfe normalerweise keine Gedanken machen, dies übernimmt der Compiler, Assembler-Programmierer müssen diesen Taskkopf jedoch selbst generieren. Nach dem Laden eines Programms muß dieser Taskkopf im Speicher vorhanden sein; spricht man unter RTOS-UH von der Adresse einer Task, so meint man die Adresse des Taskkopfes, die sog. TID (Task-Identification). Neben den hier explizit aufgeführten Informationen sind im Taskkopf noch wesentlich mehr zum Teil nur betriebssysteminterne Daten enthalten.

3.1.1.1 Taskname

Der Taskname ist, neben der TID, ein weiteres wichtiges Kennzeichen einer Task. Alle Tasks in einem Rechner sollten nicht nur über ihre TID, sondern auch über den Namen eindeutig voneinander unterscheidbar sein. Probleme bei Namensgleichheit treten spätestens dann auf, wenn eine Task über ihren Namen gestartet werden soll.

Ein Taskname darf z. Zt. aus max. 24 Zeichen bestehen.

3.1.1.2 Zustand

Der Zustand einer Task gibt an, ob und ggf. wie eine Task in irgendwelche Aktivitäten verwickelt ist. Der Zustand kann vom Bediener über Bedienbefehle sowie von Tasks und dem Betriebssystem über Tasking-Operationen gesteuert werden.

3.1.1.3 Priorität

Die wichtigste Eigenschaft einer Task ist für RTOS-UH ihre Dringlichkeit, die Priorität. RTOS-UH verteilt die Prozessorkapazität unter den lauffähigen Tasks gemäß deren individueller Priorität.

Prioritäten werden als Ganzzahl im Bereich -32768 .. 32767 angegeben, in der Reihenfolge sinkender Priorität. Eine Task mit der Priorität 1 ist also dringlicher als eine Task mit der Priorität 5. Im Gegensatz zu den Zahlenwerten sagt man auch, die Task mit der Priorität 1 habe eine höhere Priorität als die Task mit der Priorität 5. Negative Werte sind, zumindest von der Intention her, dem Betriebssystem selbst vorbehalten.

Jedesmal, wenn es für RTOS-UH eine Veranlassung gibt, den Prozessor einer Task zuzuteilen, d.h. eine Task auszuführen, inspiziert der Prozeßumschalter des RTOS-UH, der sog. *Dispatcher*, eine Liste der vorhandenen Tasks und teilt den Prozessor der laufwilligen Task mit der höchsten Priorität zu. Für den Fall, das einmal keine Task laufwillig sein sollte, enthält RTOS-UH eine Leerlauf-Task, die Task mit dem Namen **IDLE**, die die niedrigstmögliche Priorität besitzt und stets laufwillig ist. Diese Task erfüllt keinerlei Aufgaben, ist aber für die Funktion des Betriebssystems sehr wichtig.

3.1.1.4 Speicherbedarf

Um möglichst viele Tasks im Rechner halten zu können, teilt RTOS-UH den Tasks Speicher für lokale Variable erst beim Starten zu. Daher muß die Größe des erforderlichen Task-Arbeitspeichers, des sog. Task-Workspace (**TWSP**), im Taskkopf eingetragen sein. Beim Starten einer Task, unter RTOS-UH spricht man vom Aktivieren einer Task, teilt RTOS-UH dieser Task dann Speicherplatz in der erforderlichen Größe zu. Beendet eine Task ihre Arbeit, unter RTOS-UH spricht man vom Terminieren einer Task, so wird dieser **TWSP** wieder zu freiem Speicher und kann vom Betriebssystem anderweitig zugeteilt werden.

3.1.1.5 Residente Tasks

Erklärt sich eine Task als resident, so wird Ihr der **TWSP** nur bei der allerersten Aktivierung vom Betriebssystem neu zugeteilt. Sie behält diesen Speicherbereich, auch wenn sie terminiert ist. Dieses Verfahren hat den Vorteil, das bei häufig aktivierten Tasks nicht jedesmal neu Speicher zugeteilt werden muß (Zeitgewinn) und ermöglicht andererseits einer Task, sich statische Variablen zu halten, d.h. Variablen, deren Wert über eine Terminierung bis zur nächsten Aktivierung erhalten bleibt.

3.1.1.6 Autostart-Fähigkeit

Gerade für stand-alone-Systeme wichtig: erklärt sich eine Task als Autostart-Task, so wird sie direkt nach dem Start des Betriebssystems von diesem aktiviert. Alle Tasks, die diese Eigenschaft nicht besitzen, werden nicht automatisch aktiviert. Sie können nur von anderen Tasks oder über Bedienbefehle gestartet werden.

Unter RTOS-UH gibt es keine andere Möglichkeit für Tasks, aus eigenem Antrieb lauffähig zu werden.

3.2 Multi-Tasking

Das Konzept von RTOS-UH beruht auf der Idee des Multi-Tasking, der Fähigkeit des Betriebssystems, mehrere eigenständige (und allein lauffähige) Programme (quasi-) gleichzeitig betreuen zu können. Diese Tasks können gleichzeitig im Rechner vorhanden sein, und das Betriebssystem verteilt die verfügbare Prozessorkapazität auf die laufwilligen Tasks.

3.2.1 Task-Zustände

Das kleine Wörtchen *laufwillig* kennzeichnet schon, daß RTOS-UH für eine Task unterschiedlichste Betriebszustände kennt. RTOS-UH verwaltet die Tasks in ihren unterschiedlichen Betriebszuständen und stellt Mechanismen zur Veranlassung und Kontrolle von Zustandsübergängen bereit. In der Grafik Abbildung 3-2: Zustandsübergange

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

Neben den grundsätzlichen Taskzuständen **DORM** (schlafend), **RUN** (laufwillig), **SUSP** (ausgesetzt), **SCHD** (eingeplant) und **SEMA** (warten auf Synchronisation), die direkt durch Task- oder Bedieneroperationen erzielt werden können, werden hier auch die eher systeminternen Zustände **I/O?** (wartend auf Abschluß einer I/O-Operation), **PWS?** (wartend auf Speicher), **????** (Mehrfachblockierung) und **CWS?** (warten auf I/O-Speicher) beschrieben.

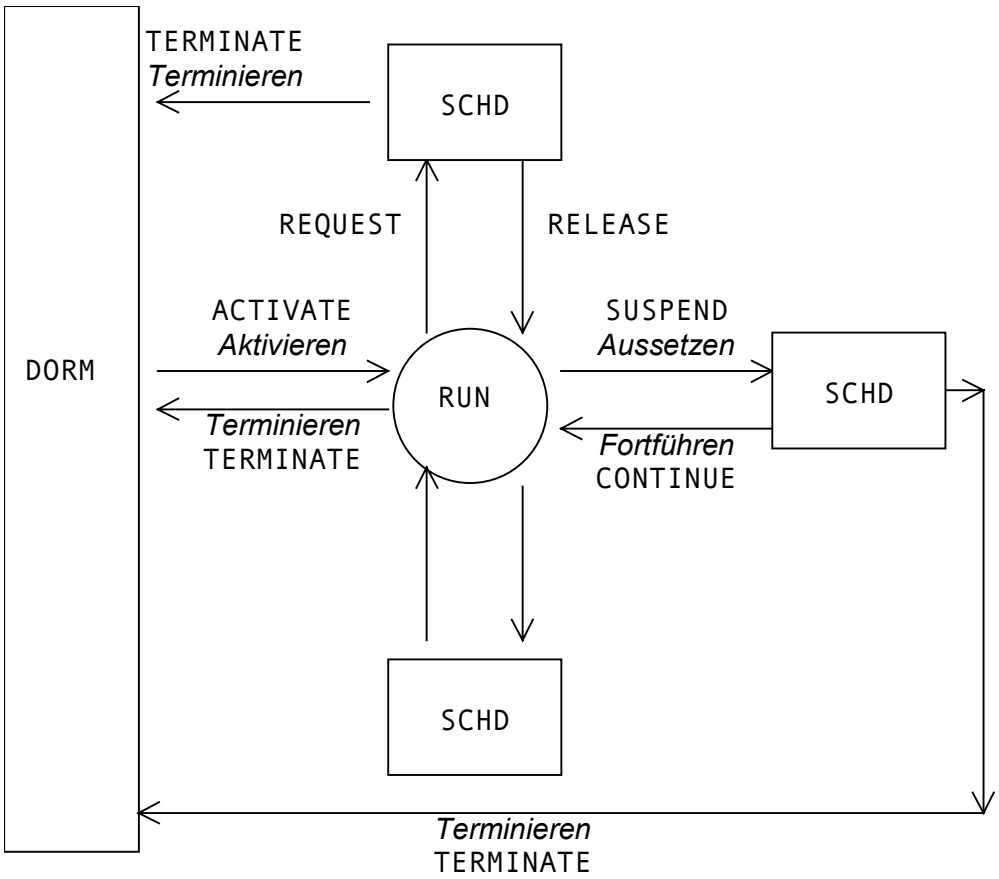

Abbildung 3-2: Zustandsübergange

Kursiv: Bezeichnung der Zustandsübergänge

`Gerade`: Bedienbefehle zur Auslösung des betreffenden Übergangs

3.2.1.1 DORM

Der einfachste Betriebszustand wird **DORM** genannt, engl. dormand, dt. schlafend. Eine Task im Zustand **DORM** ist im Rechner geladen, aber nicht laufwillig. Sie wird nie Prozessorkapazität erhalten, es sei denn ihr Betriebszustand ändert sich durch äußere Einwirkung. Eine schlafende Task beansprucht vom Betriebssystem lediglich Speicherplatz für ihren Code und ggf. (residente Tasks !) für statische Variablen. In allen anderen Betriebszuständen besitzt eine Task mit Sicherheit Speicherplatz für ihren **TWSP** (einzige Ausnahme s. Übergang Aktivierung).

3.2.1.2 RUN

Im Zustand **RUN** ist eine Task laufwillig. Sobald keine höher priorisierte Task laufwillig ist, wird diese Task den Prozessor erhalten und damit ihre Aufgabe bearbeiten können. Der übergang vom Zustand **DORM** in den Zustand **RUN** wird Aktivierung dieser Task genannt. Sobald die Task den Prozessor zugeteilt erhält, beginnt sie ihre Abarbeitung beim Programmanfang. Die Aktivierung kann nur von außen, d. h. von einer anderen Task oder vom Bediener über den Befehl **ACTIVATE** *Taskname* erfolgen. Es ist klar, daß eine Task sich nicht von selbst aus dem Zustand **DORM** in den Zustand **RUN** bewegen kann, da sie ja im schlafenden Zustand keine Prozessorkapazität erhält und somit ihre eigene Aktivierung nicht veranlassen kann.

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

Eine Task im Zustand **SUSP**, engl suspended, dt. unterbrochen, ist zwar im Prinzip laufwillig und hat auch schon ein- oder mehrmals Prozessorkapazität erhalten, will aber zur Zeit keine Prozessorkapazität erhalten. Sie hat die Bearbeitung ihrer Aufgabe freiwillig oder von außen gesteuert für eine undefinierte Zeitdauer ausgesetzt. Bei der Verteilung der Prozessorkapazität übergeht RTOS-UH eine suspendierte Task.

Der Zustand **SUSP** kann vom Bediener über den Befehl **SUSPEND** *Taskname* hergestellt werden. Die betroffene Task setzt ihre Ausführung an der Stelle, an der sie gerade ist, aus und kann später über den Befehl **CONTINUE** *Taskname* wieder fortgesetzt werden.

3.2.1.4 SCHD

Eine Task ist engl. scheduled, dt. eingeplant. Der Zustand ist dem **SUSP** insofern ähnlich, als diese Task im Prinzip laufwillig ist, jedoch zur Zeit keine Prozessorkapazität erhalten will. Allerdings ist für diese Task schon eine Bedingung definiert, unter der sie in den Zustand **RUN** überführt wird. Diese Bedingung kann sowohl eine zeitliche Einplanung (z.B. vom Bediener **AT 17:00:00 ACTIVATE** *Taskname*) als auch eine Einplanung auf einen Interrupt (z.B. vom Bediener **WHEN EV 00000001 ACTIVATE** *Taskname*) sein; genaueres zu Einplanungen im entsprechenden Kapitel. Der Übergang in den Zustand **SCHD** ist aus allen anderen Zuständen mit Ausnahme des Zustandes **SEMA** möglich. Insbesondere ist allein durch den Zustand **SCHD** nicht gesagt, ob diese Task schon einmal Prozessorkapazität erhalten hat, oder ob sie erst durch die Einplanung aktiviert wird. Wird die gegebene Einplanungsbedingung erfüllt, überführt RTOS-UH die entsprechende Task automatisch in den Zustand **RUN** und teilt ihr den Prozessor zu, falls keine höher priorisierte Task laufwillig ist.

3.2.1.5 I/O?

Eine Task wartet auf den Abschluß einer Ein- oder Ausgabe. Unter RTOS-UH werden die tatsächlichen Ein- und Augabeoperationen nicht direkt von der veranlassenden Task ausgeführt (s. Kap. 5, Das I/O-System), sondern von sog. Betreuungstasks. Eine Task im Zustand **I/O?** hat sie eine Ein- oder Ausgabeoperation angestoßen und wartet auf deren Abschluß. Für diese Zeit erhält sie keine Prozessorkapazität zugeteilt. Sobald die I/O-Operation abgeschlossen ist, überführt RTOS-UH diese Task wieder in den Zustand **RUN**.

3.2.1.6 PWS?

Eine Task wartet auf die Zuteilung von Speicher, i.a. Prozedurworkspace, **PWSP**. Sie hat vom Betriebssystem Speicher in einer Größe verlangt, die zur Zeit nicht zur Verfügung steht. Da RTOS-UH den Speicher dynamisch verwaltet, kann jedoch jederzeit wieder ausreichend freier Speicher entstehen. RTOS-UH befriedigt dann diese Speicheranforderung (bei mehreren wartenden Tasks in der Reihenfolge der Priorität) und überführt die Task automatisch wieder in den Zustand **RUN**.

3.2.1.7 CWS?

Eine Task wartet auf die Zuteilung von Speicherbereich für I/O-Operationen. Dies ist ein Sonderfall des Status **PWS?**, der nicht daher rührt, daß im System nicht mehr genügend freier Speicher vorhanden ist, sondern aus dem Überschreiten des Speicherkontingents für I/O-Operationen resultiert. RTOS-UH begrenzt den für jede einzelne Task bei jeder einzelnen Aktivierung für I/O-Operationen bereitgestellten Speicherplatz auf 2 kB, um zu verhindern, daß eine einzelne Task durch zahlreiche Ausgaben den Systemspeicher übermäßig verkleinert.

3.2.1.8 ????

Eine Task ist mehrfach blockiert, z.B. wenn sie im Zustand **PWS?** noch vom Bediener suspendiert wurde. Die Ausgabe von **????** als Taskzustand ist hierbei nur als Verzweiflungstat des Betriebssystems zu werten, daß den Taskzustand nicht mehr in einem Wort berichten kann. Intern wird die Kombination unterschiedlicher Zustände korrekt behandelt.

Allerdings kann die Ausgabe von **????** als Taskzustand auch auf eine Notwehrreaktion des Betriebssystems hindeuten: hat eine Task eine unerlaubte Aktion durchgeführt, die das Betriebssystem nicht mehr abfangen, sondern nur beobachten konnte (z.B. Speicherzugriffsfehler bei privilegierten Speicherzugriffen), so versetzt es diese Task in den Zustand **????**, um ein Weiterlaufen der Task zuverlässig zu verhindern.

3.2.2 Taskzustands-Übergänge

Nachdem die einzelnen Zustände, in denen sich Tasks befinden können, nun geklärt sind, werden hier die Möglichkeiten, Zustandsübergänge zu veranlassen und zu beeinflussen, erläutert. Dies betrifft nur die grundsätzlichen Taskzustände **DORM**, **RUN**, **SUSP**, **SCHD** und **SEMA**. Die Operationen, die Zustandsübergänge veranlaßen können, sind Aktivieren, *Terminieren*, *Aussetzen*, *Fortführen*, *Einplanen*, *Ausplanen* und Synchronisationsoperationen sowie das Eintreten von Zeitpunkten und Interrupts.

Hier werden die Wirkung der Operation sowie die veranlassenden Bedienbefehle und PEARL-Anweisungen grob beschrieben, detailliertere Erläuterungen finden sich im Handbuch, Kapitel Bedienbefehle, sowie für Assemblerprogrammierer im Handbuch, Kapitel Traps.

3.2.2.1 Aktivieren

Aktivieren hat nur auf Tasks im Zustand **DORM** direkte Einwirkung: sie werden in den Zustand **RUN** versetzt, und die beginnt bei der Zuteilung des Prozessors am Programmanfang. Eine Aktivierung einer Task in einem anderen Zustand ist möglich und hat eine sog. gepufferte Aktivierung zur Folge. RTOS-UH merkt sich (im Taskkopf, wo sonst), daß diese Task aktiviert wurde, und startet sie von Programmanfang an neu, sobald sie wieder in den Zustand **DORM** geraten will.

Bei der Aktivierung teilt RTOS-UH zunächst der Task den Speicherplatz für ihre lokalen Variablen zu (**TWSP**), anschliessend wird die Task vom Anfang an gestartet. Steht im System nicht genügend Speicherplatz zur Verfügung, so wird die Task dennoch in den Zustand **RUN** versetzt. RTOS-UH überprüft dann bei jeder Änderung der Speicherbelegung, ob die Anforderung bedient werden kann, und teilt den Prozessor erst zu, wenn der Task auch **TWSP** zur Verfügung steht. Die Aktivierung kann sowohl vom Bediener als auch von einer beliebigen Task veranlaßt werden, Tasks können sowohl sich selbst als auch andere Tasks aktivieren.

Eine Aktivierung wird über den Bedienbefehl

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 Terminieren

Terminierung ist die zur Aktivierung entgegengesetzte Operation. Die Ausführung einer Task wird an der Stelle, an der sie sich gerade befindet, abgebrochen. Die Task wird sofort in den Zustand **DORM** überführt, es sei denn, eine gepufferte Aktivierung liegt vor. Bei der Terminierung wird der Task jeglicher zugeteilter Speicher entzogen (Ausnahme: **TWSP** residenter Tasks und I/O-Speicherbereiche) und wieder zu freiem Speicher erklärt. Hat eine Task schon I/O-Operationen angestoßen, die noch nicht beendet sind, so wird wie folgt verfahren:

- 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.

Eine Terminierung ist in jedem Taskzustand möglich. Auch eine Task, die schon **DORM** ist, kann terminiert werden; das Betriebssystem führt in diesem Fall allerdings keine Aktionen durch. Die Terminierung kann sowohl vom Bediener als auch von einer Task veranlasst werden, für Tasks ist sowohl Selbst-Terminierung als auch Fremdterminierung zulässig.

Die Terminierung wird durch den Bedienbefehl

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

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

3.2.2.3 Aussetzen

Durch Aussetzen geht eine Task vom Zustand **RUN** in den Zustand **SUSP** über. Die Ausführung der Task wird für eine beliebige Zeit ausgesetzt, die Task sozusagen in ihrem gegenwärtigen Zustand eingefroren. Eine ausgesetzte Task wird bei der Zuteilung von Prozessorkapazität übergangen.

Alle lokalen Variablen bleiben erhalten, jeder zugeteilte Speicher ebenfalls. Von der Task angestossene I/O-Operationen werden ausgeführt.

Das Aussetzen kann sowohl vom Bediener als auch von Tasks veranlaßt werden, Tasks können sowohl sich selbst als auch fremde Tasks in der Ausführung aussetzen. Für eine einzelne Task ist auch das Aussetzen, verbunden mit der Definition einer Fortsetzbedingung, möglich, s. Einplanen mit Aussetzen. Das Aussetzen wird veranlaßt durch den Bedienbefehl

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

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

3.2.2.4 Fortsetzen

Fortsetzen ist die zum Aussetzen entgegengesetzte Operation. Eine Task wird aus dem Zustand **SUSP** in den Zustand **RUN** gebracht, d. h. sie wird ab sofort wieder bei der Zuteilung der Prozessorkapazität berücksichtigt. Die Task nimmt ihre Tätigkeit an genau der Stelle, an der sie ausgesetzt wurde, wieder auf. Für die Task selbst ist bis auf eine Änderung der Uhrzeit kein Unterschied zu ununterbrochener Ausführung merkbar. Die Fortsetzung kann sowohl vom Bediener als auch von einer anderen Task veranlasst werden. Klar dürfte sein, daß eine ausgesetzte Task keine Möglichkeit hat, sich selbst wieder fortzusetzen, es sei denn, sie beginnt das Aussetzen mit einer Einplanung zur Fortsetzung (s. Einplanen). Die Fortsetzung wird vom Bedienbefehl

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

der PEARL-Anweisung **CONTINUE** und den Traps **CON** und **CONQ** veranlaßt.

3.2.2.5 Einplanen

Einplanungen sind das Herz von RTOS-UH. Die Fähigkeit, Aktivierung und Fortsetzung von Tasks von dem Eintreten unterschiedlichster Ereignisse abhängig zu machen, ist ein wesentliches Merkmal des Betriebssystems und ermöglichen die Erstellung von Programmsystemen, die zum einen sehr schnell und zum anderen sehr flexibel auf äußere Situationen reagieren können.

Einplanungen lassen sich zum einen nach der Art des Ereignisses, Zeit oder Interrupt, und zum anderen nach der Art der eingeplanten Operation, Aktivierung oder Fortsetzung, unterscheiden.

Allen Einplanungen gemeinsam ist die Änderung des Taskzustands von **DORM**, **RUN** oder **SUSP** in den Zustand **SCHD**. Die Einplanungen können vom Bediener oder von einer Task vorgenommen werden, auch ein selbstständiges Einplanen einer Task ist möglich (natürlich nicht, wenn sich diese Task im Zustand **DORM** befindet).

- zeitlich eingeplante Aktivierung

  Eine zeitlich eingeplante Aktivierung führt zum Start einer Task zu einem angegebenen Zeitpunkt oder nach einer angegebenen Zeitdauer. Es kann festgelegt werden, daß diese Aktivierung in bestimmten Zeitabständen zu wiederholen ist, und diese zyklische Einplanung kann auf eine bestimmte Zeitdauer eingegrenzt werden. RTOS-UH verwaltet Uhrzeit und Zeitdauer mit der Auflösung von 1 ms, die längste verarbeitbare Zeitdauer sind ca. 24 Tage.

  Die Einplanung auf einen Zeitpunkt erfolgt mit dem Bedienbefehl

  AT *Uhrzeit* **ACTIVATE** *Taskname*,

  die Einplanung über eine Zeitdauer mit dem Befehl

  AFTER *Zeitdauer* **ACTIVATE** *Taskname*.

  Die PEARL-Anweisungen entsprechen den Bedienbefehlen.

- zyklische Einplanung

  Eine zyklische Einplanung erfolgt mit dem Bedienbefehl

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

  und kann in der Form

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

  auf eine bestimmte Zeitspanne eingegrenzt werden. Statt der Eingrenzung auf eine bestimmte Zeitdauer kann auch eine Uhrzeit für die letzte Aktivierung in der Form

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

  gegeben werden. Die PEARL-Anweisungen entsprechen den Bedienbefehlen. Zyklische Einplanungen in der einfachen Form haben eine sofortige Aktivierung zur Folge, d.h. die erste der zyklischen Aktivierungen findet sofort statt. Ist dies nicht erwünscht, so kann die zyklische Einplanung mit der Einplanung auf einen Zeitpunkt oder über eine Zeitdauer kombiniert werden, z.B.

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

  oder

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

  Auch hier sind die PEARL-Anweisungen dentisch den Bedienbefehlen. Assemblerprogrammiern stehen die Traps **TIAC** und **TIACQ** zur Verfügung, die für alle zeitlich eingeplanten Aktivierung einsetzbar sind.

- auf Interrupt eingeplante Aktivierung

  Sehr wichtig für ein Echtzeit-Betriebssystem ist die Fähigkeit, schnell auf externe Ereignisse reagieren zu können. Externe Ereignisse sind Ereignisse, die asynchron zum Programmablauf auftreten und dem Prozessor von der Peripherie signalisiert werden, sog. Unterbrechungen (im regulären Programmablauf) oder Interrupts. RTOS-UH kann die Aktivierung von Tasks vom Auftreten dieser Interrupts abhängig machen, d. h. tritt ein Interrupt auf, so wird die hierauf eingeplante Task aktiviert. Hat diese Task eine hohe Priorität, so wird ihr sofort der Prozessor zugeteilt, und Tasks niedriger Priorität können die Behandlung dieser Unterbrechungssituation nicht behindern.

  Die Einplanung auf einen Interrupt erfolgt mit dem Befehl

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

  der ebenso lautenden PEARL-Anweisung oder den Traps **ACTEV** oder **EVACTQ**.

- zeitlich eingeplante Fortsetzung

  Zeitlich eingeplante Fortsetzungen entsprechen den zeitich eingeplanten Aktivierungen, wobei weder eine zyklische Einplanung noch die Begrenzung auf einen bestimmten Zeitraum oder bis zu einem bestimmten Zeitpunkt möglich sind. Eine zeitlich eingeplante Fortsetzung wird mit den Bedienbefehlen

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

  oder

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

  den analogen PEARL-Anweisungen oder den Traps **TICON** oder **TICONQ** definiert.

Will eine Task selbst ihre Ausführung für eine bestimmte Zeitdauer aussetzen, so kann sie dies mit den PEARL-Anweisungen

AT *Uhrzeit* **RESUME**

und

AFTER *Zeitdauer* **RESUME**

oder dem Trap **TIRE** erreichen. Die Task überführt sich hierbei selbst in den Zustand **SCHD** und verzichtet bis zum Erreichen des gewünschten Zeitpunktes oder dem Verstreichen der gewünschten Zeitdauer auf die Zuteilung von Prozessorkapazität.

- auf Interrupt eingeplante Fortsetzung

  Die auf einen Interrupt eingeplante Fortsetzung ist analog der auf einen Interrupt eingeplanten Aktivierung zu verstehen. Für eine Task wird definiert, daß sie beim Eintreten eines Interrupts fortgesetzt werden soll. Voraussetzung ist natürlich, daß diese Task bis zum Eintreten des Interrupts ausgesetzt ist, andernfalls erfolgt beim Eintreten des Interrupts die Fehlermeldung **Taskname NOT SUSPENDED**, da die Fortsetzung einer nicht ausgesetzten Task unmöglich ist. Die Einplanung wird über den Bedienbefehl

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

  die analoge PEARL-Anweisung oder die Traps **CONEV** oder **EVCONQ** definiert.

  Will eine Task sich selbst gleichzeitig aussetzen und zur Fortsetzung bei einem Interrupt einplanen, so exitiert hierzu die PEARL-Anweisung

  WHEN *Interruptkennzeichnung* **RESUME**,

  bzw. die Kombination der Traps **CONEV** oder **EVCONQ** mit dem Trap **SUSP**.

3.2.2.6 Ausplanen

Ausplanen ist die zum Einplanen entgegengesetzte Operation. Ausplanungen können für eine Task nur pauschal vorgenommen werden, d.h. eine Differenzierung nach der Art der Einplanung ist nicht möglich. Durch eine Ausplanung werden alle für eine Task vorgenommenen Einplanungen gelöscht, der Zustand der Task nach der Ausplanung ergibt sich aus dem sinnvoll aus dem Taskzustand bei der Einplanung.

Die häufgsten Übergänge sind **SCHD** → **DORM**, falls eine Task zur Aktivierung eingeplant war, und **SCHD** → **SUSP**, falls eine Task zur Fortsetzung eingeplant war. Eine Task kann auch ihre eigenen Einplanungen löschen.

Die Ausplanung wird durch den Bedienbefehl

PREVENT *Taskname*,

die PEARL-Anweisung **PREVENT**, ggf. mit Taskname, und die Traps **PREV** und **PREVQ** erzielt.

3.2.3 Synchronisationsoperationen

In einem Multi-Tasking-Betriebssystem tritt häufig der Fall ein, daß mehrere Tasks auf den gleichen Datenbestand zugreifen müssen. Ändert eine Task diesen Datenbestand, so muß gewährleistet sein, daß die anderen Tasks, die den Datenbestand nur auslesen, stets konsistente Daten erhalten. Diese Forderung kann nur allein über die Prioritätenwahl nicht erfüllt werden:

- Hat die ändernde Task die höchste Priorität, so kann sie zwar stets alle Änderungen ungestört vornehmen, unterbricht aber möglicherweise eine auslesende Task mitten im Lesevorgang, was bei der auslesenden Task zu inkonsistenten Daten führt.
- Hat die ändernde Task die niedrigste Priorität, so kann sie mitten während einer Datenänderung unterbrochen werden, wodurch eine auslesende Task wiederum inkonsistente Daten erhält.

Zur Lösung dieses Problems stellt RTOS-UH Synchronisationsvariable, sog. Semaphore, ital. für Ampeln, und Bolts, engl. für Riegel, zur Verfügung.

3.2.3.1 Semaphore

Semaphore lassen sich in ihrer Funktion gut durch die Analogie zu Ampeln erklären. Im Beispiel ist eine Engstelle in der Fahrbahn gegeben, die durch beidseitige Ampeln gegen das Einfahren von Fahrzeugen sperrbar ist. An beiden Einfahrstellen sind jeweils in Fahrtrichtung Induktionsschleifen installiert, über die ein Einfahrwunsch gemeldet und das Verlassen der Engstelle erkannt werden kann.

Der einfachste zu betrachtende Fall besteht aus zwei Fahrzeugen, die sich aus Gegenrichtung der Engstelle nähern. Das Fahrzeug, das als erstes seine Einfahrt-Induktionsschleife errreicht, erhält Grün für die Weiterfahrt und setzt automatisch Rot für die Gegenseite, so daß der Einfahrtwunsch der Gegenseite abgelehnt wird. Erst mit dem Überfahren seiner Austritts-Induktionsschleife gibt das Fahrzeug die Engstelle frei und erwirkt Grün für das wartende Fahrzeug.

Übertragen auf Semaphore unter RTOS-UH erfordert nur eine neue Terminologie: das Erfragen der Einfahrterlaubnis durch Überfahren der Einfahrt-Induktionsschleife wird **REQUEST**-Operation genannt, das Verlassen der Engstelle **RELEASE**-Operation. Die einzelnen Tasks sind wie die Fahrzeuge zu betrachten: Task A führt ein **REQUEST** auf die Semaphore *Engstelle* durch und belegt die *Engstelle* hiermit. Diese Operation hat keine Auswirkung auf den Zustand der Task A. Führt nun Task B, die z.B. durch ein Unterbrechungssignal oder einen Bedienereingriff lauffähig geworden ist und wegen ihrer höheren Priorität den Prozessor zugeteilt hat, ebenfalls eine **REQUEST**-Operation durch, so wird sie suspendiert, d.h. ihre Ausführung wird trotz der höheren Prioritätausgesetzt und ihr Zustand auf **SEMA**, d.h. wartend auf das Freiwerden einer Semaphore, gesetzt. Es wird eine Neuzuteilung des Prozessors notwendig, und Task A hat gute Chancen, nun die höchstpriorisiert, laufwillige Task zu sein und den Prozessor zugeteilt zu erhalten.

Verläßt Task A die Engstelle, so führt sie eine **RELEASE**-Operation auf die Semaphore *Engstelle* durch. Hiermit wird die Engstelle wieder freigegeben. Auf den Computer übertragen, bedeutet dies die Notwendigkeit einer erneuten Prozessorzuteilung, und hierbei wird Task B auf Grund ihrer höheren Priorität den Prozessor zugeteilt erhalten.

Umgesetzt von Fahrbahn-Engstellen auf Datenbereiche ist durch den Einsatz der Synchronisations-Operationen **REQUEST** und **RELEASE** die Integrität des Datenbestandes gewahrt geblieben.

Von der Bedienerebene her sind Semaphore nur über ihre Adressen mit den Anweisungen

REQUEST *Semaphoradresse*

und

RELEASE *Semaphoradresse*

ansprechbar; zur Vereinfachung ist auch eine Anweisung

RELEASE *Taskname*

möglich, die die Semaphore, auf die eine Task im Zustand **SEMA** wartet, freigibt. Man beachte, daß die letztgenannte Form der Anweisung zum einen nicht spezifisch eine bestimmte Semaphore anspricht, da die Adresse der betroffenen Semaphor-Variablen nicht angegeben wird, zum anderen aber stets genau eine und nur eine Semaphore betrifft, da eine Task im Zustand **SEMA** stets nur auf eine Semaphore warten kann.

Von PEARL her stehen die gleichlautenden Anweisungen zur Verfügung, die entsprechenden Traps heißen **REQU** und **RELEA**.

In Erweiterung zu dem bisher Geschilderten sind Semaphore unter RTOS-UH mehrwertig implementiert, d.h. je nach Vorbesetzung der Semaphore können auch mehrere **REQUEST**-Operationen ohne Blockierung durchgeführt werden. Damit ist dann auch die Synchronisation zweier als Erzeuger und Verbraucher tätigen Tasks möglich: Führt der Verbraucher vor Verwendung der erzeugten Daten (die z.B. in einem Ringpuffer abgelegt werden können) eine **REQUEST**-Operation durch, und führt der Erzeuger stets nach Bereitstellung eines Datensatzes eine **RELEASE**-Operation durch, so ist die Synchronisation beider gewährleistet. Der Verbraucher stets erst dann lauffähig, wenn zu bearbeitende Daten erzeugt wurden.

3.2.3.2 Bolts

Bolts arbeiten gegenüber Semaphoren wesentlich differenzierter. Sie können zwischen nur lesenden Tasks und Tasks, die ggf. auch Daten modifizieren, unterscheiden. Lesende Prozesse können durch die Operation

```
ENTER Boltvariable
```

den Beginn des Datenzugriffs und mirt der Operation

```
LEAVE Boltvariable
```

das Ende ihres Zugriffs auf den kritischen Datenbereich signalisieren. Schreibende Prozesse, d.h. Tasks, die Daten modifizieren, benutzen statt dessen die Operation

```
RESERVE Boltvariable
```

beim Eintritt und

```
FREE Boltvariable
```

beim Verlassen des kritischen Pfades.

Ziel dieser unterschiedlichen Operationen ist es, lesende Prozesse nicht zu behindern, wenn kein Schreiber die Daten benutzen will. Solange kein Schreiber eine **RESERVE**-Operation durchgeführt hat, dienen die **ENTER**- und **LEAVE**-Operationen nur dazu, die Benutzung der Daten zu markieren. Erst bei der Abarbeitung einer **RESERVE**-Operation greift der mit den Bolts verknüpfte Synchronisationsmechanismus. Ist zu dieser Zeit ein lesender Prozeß im kritischen Pfad, so wird die Ausführung des **RESERVE**ierenden Schreibers ausgesetzt, bis der oder die Leser den kritischen Pfad verlassen haben. Gleichzeitig wird der Eintritt in den kritischen Pfad für Leser und weitere Schreiber gesperrt. Sind keine Leser mehr im kritischen Pfad, d.h. haben alle Leser ihre **LEAVE**-Operation durchgeführt, so wird der Schreiber wieder in den Zustand lauffähig versetzt und erhält ggf. den Prozessor zugeteilt. Nach Abarbeitung der dem **RESERVE** zugeordneten **FREE**-Operation wird der kritische Pfad wieder freigegeben.

3.2.4 Ereigniseintritt

Ein Übergang aus dem Zustand **SCHD** in den Zustand **RUN** ist nur durch den Eintritt des bei der Einplanung festgelegten Ereignisses möglich, sei es durch das Erreichen einer Uhrzeit, durch das Verstreichen einer Zeitdauer oder durch das Auftreten eines Interrupts.

Der Bediener kann hierauf nur begrenzt Einfluß nehmen. Eine Beeinflußung der Uhrzeit ist mit dem Befehl **CLOCKSET** möglich (kein PEARL-Äquivalent, kein Trap); ein Interrupt kann mit dem Befehl

TRIGGER *Interruptkennzeichnung*

simuliert werden. Die PEARL-Anweisung entspricht dem Bedienbefehl, der entsprechende Trap heißt **TRIGEV**.

3.2.5 Überblick über Taskzustandsübergänge

Hier die möglichen Zustandsübergänge mit veranlassenden Operationen im Überblick:

DORM	→	**RUN**	:	Aktivieren
DORM	→	**SCHD**	:	Einplanen
RUN	→	**DORM**	:	Terminieren
RUN	→	**SUSP**	:	Aussetzen
RUN	→	**SCHD**	:	Einplanen mit Aussetzen
RUN	→	**SEMA**	:	Synchronisationsoperation REQUEST
SUSP	→	**DORM**	:	Terminieren
SUSP	→	**RUN**	:	Fortsetzen
SUSP	→	**SCHD**	:	Einplanen
SCHD	→	**DORM**	:	Ausplanen
SCHD	→	**RUN**	:	Ereigniseintritt
SCHD	→	**SUSP**	:	Ausplanen
SEMA	→	**DORM**	:	Terminieren
SEMA	→	**RUN**	:	Synchronisationsoperation RELEASE
SEMA	→	**SCHD**	:	unmöglich
SCHD	→	**SEMA**	:	unmöglich
SUSP	→	**SEMA**	:	unmöglich
DORM	→	**SEMA**	:	unmöglich
DORM	->	**SUSP**	:	unmöglich

Tabelle 3-1: Task-Zustandsübergänge

3.3 Interrupt-Routinen

Nach dem Konzept der Tasks ist das zweitwichtige Element für die Effizienz von RTOS-UH der planvolle Einsatz von Interrupt-Routinen.

Interrupts, Unterbrechungen, sind Ereignisse, die von der Peripherie bzw. den Treiberbausteinen für Preipheriegeräte unter bestimmten Umständen ausgelöst werden. Sie unterbrechen den regulären Ablauf eines Programms und werden von speziellen Programmsegmenten, den Interruptroutinen, bearbeitet. Folgendes Bild mag den üblichen Programmablauf und die Bearbeitung eines Interrupts verdeutlichen:

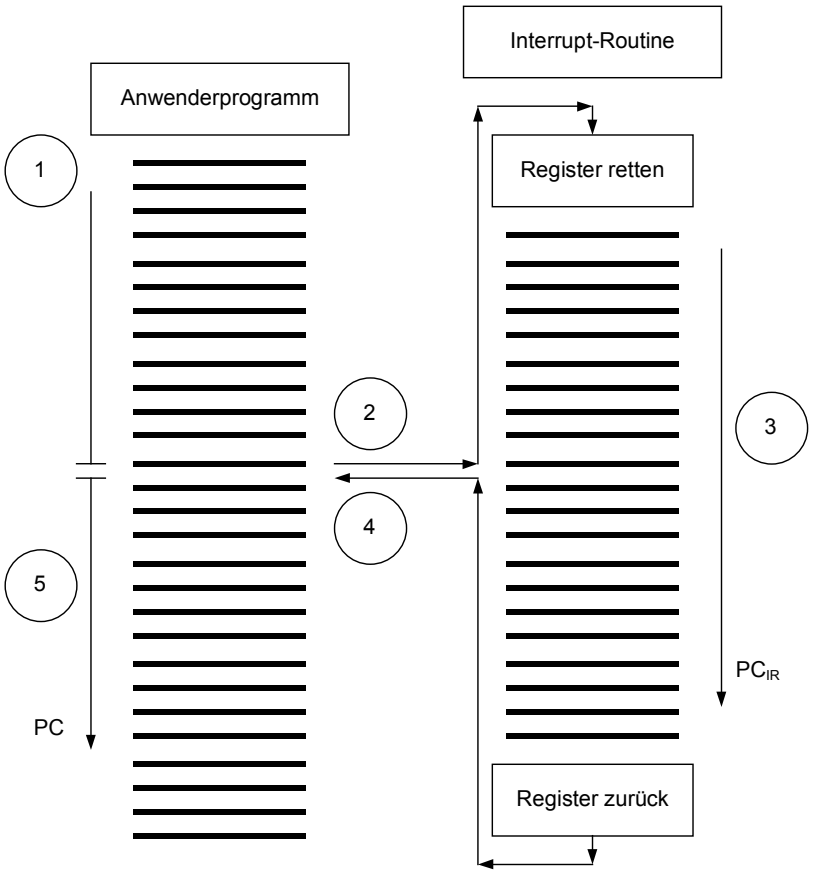

Abbildung 3-3: Programmunterbrechung durch Interrupt

1. Anwenderprogramm läuft
2. Unterbrechung – Retten des Prozessorzustands
3. Abarbeitung der Interrupt-Routine
4. Restaurieren des Prozessorzustands
5. Anwenderprogramm läuft

Die Ursache für den Einsatz von Interrupt-Routinen liegt in der im Vergleich zum Prozessor langsamen Arbeitsweise von Peripheriegeräten. Bei einer seriellen Schnittstelle dauert z.B. die Übertragung eines Zeichens bei 9600 Baud ca. 1 ms, der Prozessor könnte die Daten jedoch um einen Faktor von ca. 1000 schneller senden oder empfangen. Da die Übertragung eines Zeichens von

der Hardware selbstständig erledigt wird und den Prozessor nicht benötigt, ist es sinnvoll, den Prozessor in dieser Zeit andere Aufgaben erledigen zu lassen. Erst bei Ende der Übertragung eines Zeichens wird der Prozessor wieder benötigt, sei es, um ein neues Zeichen bereitzustellen oder um das Ende der Übertragung zu erkennen. RTOS-UH nutzt diese Pausen durch Verteilung der Prozessorkapazität an laufwillige Tasks. Lediglich dann, wenn der Prozessor für die Betreuung eines I/O-Bausteins erforderlich ist, wird er auch hierfür eingesetzt. Die I/O-Bausteine lösen in diesem Fall einen Interrupt aus, der die regulär arbeitenden Programme unterbricht. RTOS-UH hält für diese Fälle spezielle Interruptroutinen zur Betreuung der Bausteine bereit. Hiermit wird erzielt, das an keiner Stelle des Betriebssystems Prozessorkapazität durch wiederholtes Abfragen von Peripheriebausteinen verschwendet wird.

Interrupt-Routinen liegen außerhalb des Task-Konzeptes von RTOS-UH, da die Interruptbeantwortung schon im Prozessor als Sonderfall angelegt ist. Für die Dauer der Bearbeitung einer Interrupt-Routine ist der Taskwechsel-Mechanismus von RTOS-UH paralysiert. Um diese Paralyse-Zeiten möglichst gering zu halten, arbeiten sämtliche Interrupt-Routinen von RTOS-UH mit Systemdiensten zusammen, die im Regelfall als Tasks angelegt sind. Auch hier werden die RTOS-UH-eigenen Mechanismen der Taskzustandsänderung ausgenutzt: die Systemtasks setzen ihre Abarbeitung aus oder planen sich ein, um dann von den Interrupt-Routinen wieder fortgeführt zu werden. Kontrollierten Zugriff auf diese Interrupt-Routinen hat lediglich der Systemprogrammierer, Anwenderprogramme bedienen sich hierzu stets der Systemtasks.

3.3.1 Timer-Interrupt

Eine Ausnahme bildet hier lediglich die Systemuhr: ein periodischer Interrupt, der in den meistens RTOS-UH-Systemen jede Millisekunde ausgelöst wird, verwaltet die Systemzeit selbstständig. Die hierzugehörende Interrupt-Routine prüft bei jedem Uhr-Interrupt, auch Clock-Tick genannt, ob ein Zeitpunkt vorliegt, zu dem eine Einplanung existiert. Ist dies der Fall, so werden die erforderlichen Maßnahmen zur Taskzustandsänderung durchgeführt und der Taskumschalter gestartet, um den Prozessor der nun höchstpriorisierten Task zuzuteilen.

3.3.2 Schnittstellen-Interrupt

Wie schon oben erwähnt, ist die Ausgabe von Zeichen über die meisten seriellen und parallelen Schnittstellen langsam im Vergleich zur Prozessorgeschwindigkeit. Daher gibt es in RTOS-UH für jede Schnittstelle eine Interrupt-Routine, die die Ein- und Ausgabe von Zeichen unabhängig von jeder Task durchführt. Lediglich für Anfang und Ende des Ein- oder Ausgabevorgangs arbeiten diese Interruptroutinen mit ihren Systemtasks zusammen. Für die Dauer der tatsächlichen Ein- oder Ausgabe setzen diese Systemtasks ihre Abarbeitung aus (sind im Zustand `SUSP`) und werden von den Interrupt-Routinen nach Beendigung der Ein-/Ausgabe wieder fortgesetzt.

3.3.3 Floppy-Interrupt

Auch beim Betrieb von Massenspeichern treten längere Wartephasen auf. Daher existieren auch hier im Regelfall Interrupt-Routinen, die es ermöglichen, das der Prozessor in Wartephasen den laufwilligen Tasks zugeteilt wird. Die zugehörige Systemtask hält für diese Zeit ihre Abarbeitung ebenfalls an und wird bei Bedarf von der Interrupt-Routine fortgesetzt.

3.4 Systemtasks

Nachdem nun geklärt ist, was Tasks sind und was mit ihnen gemacht werden kann, soll hier die Funktion der fest zu RTOS-UH gehörenden Systemtasks erläutert werden. Hier wird als erstes der

praktische Nutzen des Task-Konzeptes sichtbar, sicherlich viele Eigenarten des Betriebssystems werden verständlich und auch als in sich schlüssig anerkannt.

Die Funktion des RTOS-UH beruht auf einer Reihe von Systemtasks, die neben den im Betriebssystemkern realisierten Dienstleistungen eine Reihe von Schlüsseldiensten anbieten. Alle jeweils vorhandenen Systemtasks sind für die Funktion des Betriebssystems erforderlich und erfüllen ihre Aufgabe ohne Eingriffe des Anwenders. Daher sind sie auch schon von in ihrer Namensgebung vor unerlaubten Eingriffen geschützt: das Zeichen **#**, mit dem ihre Namen beginnen, kann nicht vom Bediener als Bestandteil eines gültigen Tasknamens eingegeben werden. Der Einfachheit halber werden die Tasks im Folgenden aber ohne dieses führende **#** benannt. Weiterhin werden im Folgenden auch nicht alle denkbaren bzw. schon realisierten Arten von Systemtasks aufgeführt. Insbesondere bei den Betreuungstask werden nur die typischen Vertreter vorgestellt. Andere Systemtasks mit ähnlichen Aufgaben verhalten sich ähnlich und werden auch ähnlich realisiert.

3.4.1 Betreuungstasks und Datenstationen

Eine Betreuungstask ist zuständig für die Verwaltung aller Anforderungen, die vom System oder von Anwenderprogrammen an ein Peripheriegerät gestellt werden. RTOS-UH enthält für jedes physikalisch unabhängige Gerät eine Betreuungstask. Diese Geräte werden unter RTOS-UH als Datenstationen bezeichnet, da sie Ziel oder Quelle für Datenübertragungen sind. Datenstationen haben einen Namen, der gemäß der RTOS-UH-Nomenklatur in der Form /Datenstation/ angegeben wird. Die genauere Bedeutung der Betreuungstasks wird im Kapitel I/O-System erläutert.

3.4.1.1 IDLE

Eine Task mit diesem Namen ist in jedem RTOS-UH-System vorhanden. Es ist die sog. Leerlauf-Task des Systems, die jederzeit laufwillig ist. Diese Task besitzt die niedrigste mögliche Priorität und erhält den Prozessor stets dann zugeteilt, wenn keine andere Task laufwillig ist. Die Task **IDLE** erfüllt keine weiteren Aufgaben außer ihrer jederzeitigen Verfügbarkeit. Der von der **IDLE** verarbeitete Code besteht normalerweise aus einer Endlosschleife. Es gibt jedoch auch Systeme, in denen die **IDLE** den Prozessor bis zum Auftreten eines Interrupts anhält. Hierdurch kann bei batteriegetriebenen Rechner Strom gespart werden.

3.4.1.2 USER

Für den Anwender zunächst am wichtigsten ist die Task mit dem Namen **USER**. Diese Task verkörpert den Bedieninterpreter eines RTOS-UH-Systems und ist für die Eingabe, Auswertung und Durchführung von Bedienbefehlen zuständig. Einmal aktiviert, zeigt sie ihre Empfangsbereitschaft für Befehle durch die Ausgabe des Zeichens ***** als Prompt. Sie wartet dann auf die Eingabe eines Kommandos. Nach Abschluß der Eingabe analysiert sie den eingegebenen Text und führt die erhaltenen Befehle durch. Der Name **USER** rührt daher, das diese Task in einem RTOS-UH-System die systeminterne Representation des Anwenders ist.

Die Task **USER** arbeitet auf einer sehr hohen Priorität, um Bedienereingriffe auch bei hochpriorisierten Dauerläufer-Tasks empfangen und bearbeiten zu können. Weiterhin besitzt diese Task jeglichen für sie erforderlichen Speicherplatz auch im Zustand **DORM**. Daher kann sie auch noch bei einem unter akutem Speichermangel leidenden Rechner gestartet werden, z. B. um die Ursache des Speichermangels per Bedienereingriff zu beheben (Terminieren eines Speicherfressers o. ä.).

Für umfangreichere Befehle, so z. B. das Übersetzen eines Programmes, generiert sie eigenständige Tasks, sog. Subtasks. Diese Tasks werden dynamisch erzeugt und arbeiten komplexe Be-

dienbefehle mit niedrigerer Priorität eigenständig ab. Dadurch wird der Bedieninterpreter frühzeitig wieder bereit zum Empfang neuer Befehle.

Ein gutes Beispiel ist hier der **COPY**-Befehl: Nach dem Absetzen eines **COPY**-Befehls wird sofort eine Subtask generiert, die den Kopiervorgang tatsächlich ausführt. Da die Task **USER** ihre Arbeit mit der Erzeugung der Subtask getan hat, kann z.B. sofort ein neuer **COPY**-Befehl abgesetzt werden. Beide Kopiervorgänge laufen dann, von zwei getrennten Subtasks bearbeitet, quasi-parallel ab. In den Wartezeiten der einen **COPY**-Subtask kann die andere **COPY**-Subtask arbeiten und umgekehrt. Neben dem psychologischen Effekt, daß Warten auf den Rechner vermieden wird, kann so auch kürzere Kopierzeit als bei der zeitlich aufeinanderfolgenden Bearbeitung von Kopiervorgängen resultieren. Aber Vorsicht: wird von oder zum Massenspeicher kopiert, kann es auch passieren, daß die insgesamt benötigte Kopierzeit größer wird als die Summe der Zeiten für einzelne **COPY**-Befehle, da auf dem Massenspeicher wesentlich mehr Positioniervorgänge stattfinden müssen.

Subtasks erhalten ihren Namen aus dem Bedienbefehl, dem eine vom System fortlaufend vergebene Nummer angehängt wird (**COPY/*xx***, die Nummer ***xx*** wird durch den Schrägstrich abgetrennt). Nach Beendigung ihrer Aufgabe verschwinden sie selbstständig wieder aus dem System.

Besitzt ein RTOS-UH System mehrere Tasks, die mit dem Namen **USER** beginnen und durch eine noch folgende Ziffer unterschieden sind, so stehen mehrere Einsprünge in den Bedieninterpreter zur Verfügung. Dies ist normalerweise bei Systemen mit mehreren seriellen Schnittstellen der Fall. Jede einzelne Task **USER*x*** ist hierbei für Befehlseingaben von einer Schnittstelle zuständig. Da für jede Schnittstelle ein **USER** existiert, können auf allen Schnittstellen unabhängig voneinander Bedienbefehle eingegeben werden (Mehrnutzer-Fähigkeit). Das hat zur Folge, das an einem Rechner mehrere unabhängige Nutzer arbeiten können.

Allerdings ist RTOS-UH kein echtes Mehrnutzer-Betriebssystem: die einzelnen Nutzer sind in keiner Form voreinander geschützt, Dateien haben keine Nutzerzugehörigkeit, Tasks können gegenseitig manipuliert werden etc. Die Task **USER1** hat im Regelfall die höchste Priorität aller **USER**-Tasks.

3.4.1.3 XCMMD

Eine Task mit dem Namen **XCMMD** ist ebenfalls ein Bedieninterpreter in einem RTOS-UH-System. Sie erhält ihre Befehle allerdings nicht von Bedienereingaben, sondern über Ausgaben anderer Programme. Jede Task kann durch Ausgaben an die Datenstation **/XC/**, die von der **XCMMD** betreut wird, Bedienbefehle absetzen. So können von Programmen auch Operationen durchgeführt werden, die nicht zum Umfang der jeweiligen Programmiersprache gehören.

3.4.1.4 ACIA, SCC, RS232

Eine Task mit dem Namen **ACIA** oder **SCC** ist für die Betreuung einer seriellen Schnittstelle zuständig. Sie empfängt von anderen Systemtasks oder von Anwendertasks Ein- und Ausgabeaufträge, verwaltet den Betriebszustand der seriellen Schnittstelle, erfüllt die Ein-/Ausgabeaufträge in Zusammenarbeit mit der ihr zugehörenden Interrupt-Routine und gibt die erfüllten Aufträge an den Auftraggeber zurück. Eine **ACIA** kann nur nicht mehrere Ein- oder Ausgabeaufträge gleichzeitig bearbeiten.

Der Name **ACIA** oder **SCC** rührt von dem Namen typischer Bausteine zum Betrieb serieller Schnittstellen her.

Dieser Aufgabenbereich ist typische für eine sog. Betreuungstask. Im Verlauf eines Ein- oder Ausgabeauftrags durchläuft eine **ACIA** verschiedene Taskzustände. Normalerweise **DORM**, wird sie vom System beim Eintreffen eines Auftrages aktiviert. Im Zustand **RUN** erfüllt sie Verwaltungsauf-

gaben und übergibt den Auftrag an die zugehörige Interrupt-Routine. Die **ACIA** begibt daraufhin in den Zustand **SUSP**. Bei Beendigung der Ein- oder Ausgabe wird sie von der Interrupt-Routine fortgesetzt. Erneut im Zustand **RUN** erledigt sie weitere Verwaltungsaufgaben und gibt den Auftrag an den Auftraggeber zurück. Anschliessend terminiert sie sich selbst.

Sind in einem System mehrere serielle Schnittstellen vorhanden, so existieren auch mehrer Tasks **ACIA*x***, wobei ***x*** kennzeichnend für die zugehörige Schnittstelle ist. Die zu einer seriellen Schnittstelle zugeordnete **USER**-Task hat die gleiche Kennzeichnung ***x*** wie die entsprechende **ACIA**. Die Bezeichnung der zugeordneten Datenstation lautet **/Ax/**. Die von der **ACIA1** betreute Schnittstelle wird als Systemschnittstelle bezeichnet, da sie den Zugang zu dem höchstpriorisierten **USER** besitzt.

3.4.1.5 SOUT

Im Regelfall existiert zu jeder **ACIA*x*** auch eine **SOUT*x***. Eine Task **SOUT** ist für die Ausgabe von Daten über eine serielle Schnittstelle im Voll-Duplex-Betrieb erforderlich, d. h. wenn gleichzeitig über eine Schnittstelle empfangen und gesendet werden soll, erfolgt die Ausgabe über die **SOUT** und die Eingabe über die **ACIA**. Eine **SOUT** kann keine Eingabeanforderungen bearbeiten, ihr Name rührt aus der Charakterisierung *serial out*.

Werden von **ACIA** und **SOUT** gleichzeitig Ausgaben verlangt, so regeln beide Tasks ihren Zugriff auf die Schnittstelle mit Synchronisationsvariablen, Semaphoren. Daher kann sich eine **SOUT** auch im Zustand **SEMA** befinden.

Die Bezeichnung der betreuten Datenstation lautet **/D*x*/**, wobei *x* wie bei der **ACIA** beschrieben gewählt ist.

3.4.1.6 EDFM

Ein RTOS-UH-System, das eine Task **EDFM** besitzt, enthält eine RAM-Disk. **EDFM** ist die Betreuungstask für diese dateiorientierte Datenverwaltung im Systemspeicher. Der Name **EDFM** stammt aus der Bezeichnung **ED**it-**F**ile-**M**anager, die andeutet, das es in RTOS-UH einen Editor gibt, der direkt auf der Ramdisk arbeiten kann. Die betreute Datenstation, die Ramdisk, heißt **/ED/**. **EDFM** hat die gleichen Aufgaben wie eine **ACIA**, arbeitet jedoch nicht mit einer Interruptroutine zusammen, da es im Speicher keine Wartephasen auf die Bereitschaft eines Peripheriegerätes gibt. Sobald **EDFM** einen Arbeitsauftrag erhalten hat, führt er ihn aus, ohne sich zwischenzeitlich in einen anderen Zustand als **RUN** zu begeben.

3.4.1.7 ERROR

Die **ERROR** ist die höchstpriorisierte Task in einem RTOS-UH-System. Sie ist für die Bearbeitung von Fehlermeldungen zuständig. Durch betriebssysteminterne Sonderbehandlung der **ERROR** kann diese nie derart blockiert werden, daß sie laufunfähig wird.

Die **ERROR** besitzt einen zyklischen Puffer, in dem Fehlermeldungen mit einer Fehlernummer und ggf. Klartext abgelegt werden. Direkt nach dem Eintragen einer Fehlermeldung wird die **ERROR** aktiviert. Sie besorgt die Bereitstellung einer Fehlermeldung zur Ausgabe über die **ACIA** und ggf. zusätzliche Maßnahmen zum fehlertoleranten Weiterlaufen des Betriebssystems. Zusätzlich ist die **ERROR** noch für die Aktivierung der **USER** zuständig: empfängt die Interrupt-Routine einer **ACIA** ein Sonderzeichen, das **^A**, **^B**, **^C** oder ein BREAK-Signal, und ist ein Aktivieren des **USER** gestattet, so übergibt sie der **ERROR** eine spezielle Meldung. Die **ERROR** aktiviert hierauf den **USER**. Dieser auf den ersten Blick etwas umständliche Weg wurde gewählt, da die **ERROR** stets lauffähig ist. So kann sie auch in dem Fall, daß ein **USER** wegen illegaler Operationen vom System zwangsweise suspendiert wurde oder aus welchen Gründen auch immer in eine Endlosschleife geraten ist, die-

sen aus seinem Schlamassel befreien und über einen Neustart des **USER**s zumindest einen Notbetrieb herstellen.

3.4.1.8 UHFM

UHFM ist die Betreuungstask für Massenspeicher, d. h. ein RTOS-UH-System, das eine Task **UHFM** besitzt, verfügt über einen Treiber für Massenspeicher (Floppy, Festplatte). Der Name stammt aus der Bezeichnung **U**niversity **H**annover **F**ile **M**anager. Der **UHFM** ist zuständig für Ein- und Ausgaben über den Massenspeicher sowie die Verwaltung der Dateistruktur auf dem Massenspeicher. Da bei Floppys und Festplatten auch Wartephasen anfallen (Positionierung der Schreib- Leseköpfe, Motoranlaufphasen), arbeitet der **UHFM** auch mit einer Interrupt-Routine zusammen. Der **UHFM** kann sich daher auch in die Zustände **SUSP** oder **SCHD** begeben.

Da RTOS-UH die Verwaltung der Daten auf Massenspeicher nach unterschiedlichen Arten ermöglicht (z. Zt. DOS-kompatible oder RTOS-UH-eigene Verwaltung), benutzt der **UHFM** unterschiedliche Verwaltungsroutinen, je nach angeforderter Verwaltungsart. Diese Verwaltungsroutinen können über externe Sprungleisten angeschlossen werden, s. Handbuch. Die betreuten Datenstation heißen **/F0/**, **/M0/**,...

3.4.1.9 VDATN

VDATN ist die Betreuungstask für ein speicherinternes, FIFO (first in, first out)- ähnlich organisiertes Ein-/Ausgabegerät. Die Datenstationen heißen **/VO/** als Ziel für Ausgaben und **/VI/** als Quelle für Eingaben.

VDATN wird insbesondere für die unsynchronisierte Kommunikation zwischen Tasks genutzt: eine Task gibt zu beliebiger Zeit eine Meldung zu VDATN hin aus, eine andere Task liest diese Meldung, wiederum zu beliebiger Zeit, ein. Hat die schreibende Task noch keine Ausgabe gemacht, so wird die lesende Task vom System solange ausgesetzt (Zustand **I/O?**), bis der Eingabewunsch befriedigt werden kann. Die schreibende Task kann auf jeden Fall ungehindert weiterarbeiten. Mehrere Ausgaben der schreibenden Task werden in der Reihenfolge ihres Eintreffens bei **VDATN** an die lesende Task weitergereicht (FIFO-Struktur). **VDATN** kann beliebig viele dieser I/O-Kanäle über Dateinamen unterscheidbar verwalten.

Erfolgen in einen I/O-Kanal allerdings mehrere Ausgaben von Tasks unterschiedlicher Priorität, so wird dem FIFO-Prinzip eine prioritätsgemäße Ordnung derart überlagert, daß die Ausgaben der höher priorisierten Tasks sich in der Reihenfolge der Priorität an den Anfang des FIFOs vordrängeln. Ebenso verdrängen Eingabeanforderungen höher priorisierter Tasks Anforderungen von Tasks niedrigerer Priorität. Der Name **VDATN** rührt von der Kennzeichnung als **V**irtuelle **DAT**e**N**-station.

3.4.1.10 NIL

NIL ist die Betreuungstask für die Datenstation **/NIL/**, die eine ideale Datenquelle und -senke darstellt. Ausgaben zur Station **/NIL/** werden stets erfolgreich durchgeführt, beim Lesen von der Station **/NIL/** erhält man stets ein CR zurück.

3.4.1.11 PPROT

PPORT ist die Betreuungstask für eine Centronics-Schnittstelle. Über **PPROT** ist nur Ausgabe zulässig, ansonsten sind die Ausführungen bei der **ACIA** hier voll übernehmbar. Der Name ist ein Kürzel für **P**rinter-**PORT**, die Datenstation heißt **/PP/**.

4 Erste Schritte

Nach der Theorie soll nun erst mal ein wenig Praxis folgen. Mit den bisher gewonnenen Kenntnissen ist eine Beobachtung des Verhaltens von RTOS-UH schon sinnvoll möglich, bevor jedoch erste Beispiele zur Demonstration des bisher Geschilderten folgen, müssen erst einmal die Grundzüge der Systembedienung sowie einige elementare Bedienbefehle dargestellt werden. Hier sollte allerdings gleich auch der Rechner benutzt werden - je eher man ihn benutzt, desto mehr Fehler kann man ungefährdet machen, bevor es ernst wird.

4.1 Systemstart

Die erste Hürde beim Start eines RTOS-UH-Systems ist genommen, wenn die Systemmeldung auf dem Bildschirm erscheint. Für Rechner, die mit angeschlossenem Terminal arbeiten, kann dies schon schwieriger sein, Rechner mit eingebautem Terminal sind da meist einfacher zu handhaben.

4.1.1 Rechnerkonfiguration mit externem Terminal

Das Terminal muß an die Hauptschnittstelle des Rechner angeschlossen sein, meist mit A oder 1 gekennzeichnet. Bei 25 p Sub-D-Verbindungen müssen die Pins 2, 3 und 7 (RX, TX und GND) verbunden sein. Die Pins 4 und 5 (RTS und CTS) sollten der Einfachheit halber zunächst an beiden Kabelenden gebrückt werden.

Das Terminal sollte TELEVIDEO-kompatibel sein, zur Not geht auch VT52. Die erforderliche Baudrate ist aus dem RTOS-UH-Handbuch, Kap. Implementierungsbesonderheiten, ersichtlich. Sollte sie unbekannt sein, so kann erst mal mit 9600 Baud gestartet werden. Die weiteren Übertragungsparameter sind: 8 Datenbits, 1 Stopbit, keine Parität. Allerdings kann genauso mit 7 Datenbits und 2 Stopbits gearbeitet werden, da RTOS-UH nur 7 Datenbits ausnutzt. Als Protokoll sollte X_{ON}/X_{OFF} gewählt werden. Das Terminal sollte keine Zeichenumsetzung vornehmen (CR=CR), bei Empfang eines LF in der untersten Bildschirmzeile automatisch scrollen und bei Überschreitung der Zeilenlänge automatisch auf den Beginn der nächsten Zeile positionieren. Das Terminal sollte kein lokales Echo vornehmen.

Sind diese Grundeinstellungen vorgenommen, kann der Rechner mit eingesetzten RTOS-UH-EPROMs oder eingelegter Bootdisk gestartet werden. Spätestens ca. 10 sec nach Rechnerstart bzw. Ende des Bootvorgangs sollte die Systemmeldung erscheinen.

Passiert auf dem Bildschirm gar nichts, so sollten an einem Ende des Terminalverbindungskabel die Verbindungen zu Pin 2 und 3 vertauscht und der Vorgang wiederholt werden. Es schadet auch nichts, zweimal die **`No scroll`**- oder **`Hold screen`**-Taste des Terminals zu betätigen. Auch ein X_{ON} (Control- und Q-Taste gleichzeitig drücken, **`^Q`**) hat noch keinen Rechner verärgert.

Erscheinen auf dem Bildschirm nur merkwürdige Zeichen (unter der Maßgabe, daß eine RTOS-UH-Systemmeldung keine Anhäufung merkwürdiger Zeichen darstellt), so stimmt wahrscheinlich die Baudrate nicht - 19200 Baud versuchen.

Hilft das alles nichts, so hilft auch diese kurze Einführung nicht weiter. Kurz vor Feierabend oder Freitag nachmittag sollte der Rechner dann am besten in Ruhe gelassen werden - vielleicht überlegt er es sich ja noch anders. Andernfalls sollte ein in der Inbetriebnahme serieller Schnittstellen erfahrener Kollege und/oder Freund zu Rate gezogen werden. Rausreden auf defekte EPROMs oder Bootdisk ist zwar praktisch - aber meist nicht sinnvoll, da jedes ausgelieferte RTOS-UH-System schon einmal gelaufen ist.

Erscheint die Systemmeldung, so funktioniert zumindest die Datenausgabe. Wenn nach gleichzeitigem Drücken der Tasten Control und A auf dem Bildschirm ein ***** erscheint, funktioniert die Gegenrichtung ebenfalls, erscheint der Stern nicht, so liegt es meistens am Kabel. Entweder fehlen hier noch Verbindungen oder Brücken, oder die Leitung ist fehlerhaft.

4.1.2 Rechnerkonfiguration mit integriertem Terminal

Rechner mit integriertem Terminal brauchen meist nur mit eingesetzten EPROMs oder eingelegter Bootdisk gestartet werden. Um inhaltlich mit den Kollegen mit externen Terminals gleichzuziehen, sollten hier nun auch die Tasten Control und A gleichzeitig gedrückt werden.

4.2 Kontaktaufnahme

Durch das Drücken der Tasten Control und A gleichzeitig ist der erste Kontakt mit dem Rechner hergestellt. Der Rechner hat den Tastendruck mit der Ausgabe eines ***** als Prompt quittiert und wartet nun auf die Eingabe eines Befehls.

Bevor die direkte Arbeit mit dem Rechner beginnt, noch einige Konventionen: gleichzeitiges Drükken der Control-Taste zusammen mit einer anderen Taste, z.B. dem A, wird im Folgenden durch die Notation **^A** angegeben. Weiterhin bedeutet *Absetzen eines Befehls* die Eingabe von **^A**, dem Befehlstext und abschliessend einem **CR**, entweder über die ***RETURN***- oder ***ENTER***-Taste oder als **^M**.

Der Zugang zum Bedieninterpreter ist in RTOS-UH grundsätzlich unterschiedlich zu anderen Betriebssystemen: vor Eingabe eines Befehls muß **^A** gegeben werden.

Der Grund hierfür ist die Blockierung der Terminalausgabe bei anhängiger Eingabe. Würde der Bedieninterpreter stets auf Eingaben warten, könnten Ausgaben von Tasks nicht auf dem Bildschirm erscheinen, es sei denn, sie würden über den Duplex-Kanal (Systemtask **SOUT**) ausgegeben. Dies wiederum hätte zur Folge, das der Eingabetext durch die eingestreuten Ausgaben unleserlich wird. Um diesen Effekt zu vermeiden, erfolgen Ein- und Ausgaben normalerweise nur über die Systemtask **ACIA**, so daß während der Bearbeitung einer Eingabe keine Ausgaben auf dem Bildschirm erscheinen können.

4.2.1 Die Systemmeldung

Auf dem Bildschirm ist immer noch die Systemmeldung vom Systemstart zu sehen. Ihr können wichtige Information über die aktuelle Konfiguration des Betriebssystems entnommen werden. Eine typische Systemmeldung sieht in etwa so aus:

```
                          ============================
                           > > >  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.
```

Neben Überschrift, Versions- und Lizenznummer sind hier die einzelnen Systembestandteile mit ihren eigenen Versionsnummer angegeben. Das hier gezeigte System, eine Version für einen 68000-Einplatinencomputer, der auch im Weiteren als Musterrechner herhalten muß, enthält folgende Bestandteile:

Nuc	:	Betriebssystemkern, Nukleus genannt, in der Version 6.5-E
Imp	:	Implementierungsscheibe mit ACIA und Uhr-Interrupt, in der Version 2.N
Pbus	:	Treiberroutinen für das Bussystem des Rechners, in der Version 1.1
Error	:	Systemtask ERROR, in der Version 0.6
EdFm	:	Systemtask EDFM, in der Version 0.5
Vi/Vo	:	Systemtask VDATN, in der Version 0.5
sh/ext	:	Bedieninterpretererweiterungen, in der Version 1.0
Sh/sr	:	Unterprogrammpaket für den Bedieninterpreter, in der Version 3.2-H
Shell	:	Bedieninterpreter mit USER, in der Version 3.2-G
Hyp	:	Laufzeitpaket für Hochsprachprogramme, in der Version 12.3-D
Dev	:	diverse Hilfsroutinen, in der Version 3.2
Math	:	Gleitkommaarithmetik, in der Version 1.B
assign	:	Ausgabenumlenkung, in der Version 0.7
r/w	:	binäre Ein-/Ausgabe, in der Version 1.2
Loader	:	Linker und Lader, in der Version 4.5-F
Help	:	Bedienbefehl HELP, in der Version 1.6c
Prom	:	Utility zur Erstellung ROM-residenter Programme, in der Version 2.8
RTC/XPAC, Pwfail, ADDint	:	rechnerspezifisch
P	:	PEARL-Compiler, in der Version Mini-12.3-E

Tabelle 4-1: Bestandteile des Betriebssystems

Anschliessend folgt die Meldung **RESET**, die anzeigt, das RTOS-UH gerade einen Kaltstart durchgeführt hat. An dieser Stelle kann auch die Meldung **ABORT** erscheinen. In diesem Fall hat RTOS-UH eine Warmstart durchgeführt, d. h. alle Taskaktivitäten geordnet beendet, einen Test auf korrekten Aufbau der Speicherverwaltung durchgeführt und alle Systembestandteile neu aufgesetzt. Ein **ABORT** ist eine gemilderte Form der Systeminitialisierung, bei der ggf. geladen Tasks und Dateien in der Ramdisk nicht verloren gehen. Die Auslösung eines **ABORT**s kann allerdings bei Erkennung von Fehlern in der Speicherverwaltung auch zu einem **RESET**, der kompletten Systeminitialisierung mit Verlust sämtlicher im RAM befindlicher Daten, führen.

Bei den meisten Rechnern können **RESET** und **ABORT** über spezielle Tasten ausgelöst werden.

4.2.2 Der Speicheraufbau

RTOS-UH ist als ein Betriebssystem konzipiert, in dem der Nutzer möglichst weitgehende Informationen über den aktuellen Zustand des Systems erhalten kann. Als erstes Beispiel hierzu soll die Speicherbelegung des Beispielrechners nach dem Einschalten betrachtet werden: mit dem Befehl `S (^A S CR`) erhält man folgende Speicherbelegungsliste:

```
*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					Adresse Speichersegments
	Ende				des Speichersegments
		Typ			des Speichersegments
			zus. Info		(optional)
				Name des Besitzers	

Tabelle 4-2: Speicherbelegung

Folgende Typen von Speichersegmenten können auftreten:

Mnemo		Bedeutung
MARK	:	speziell markiertes Segment, Anfang oder Ende des Systemspeichers
TASK	:	Taskkopf; der Code der Task kann, aber muß nicht in diesem Segment liegen. Besitzer des Segments ist die Task selbst.

ATSK : ist kein Schreibfehler, sondern der Taskkopf einer Autostart-fähigen Task.

TWSP : Task-Workspace. Der Speicherbereich für lokale Variablen einer Task. Besitzer ist diese Task.

PWSP : Prozedur-Workspace. Zusätzlich von der Task angeforderter Speicherbereich für prozedurlokale Variable. Besitzer ist auch hier die Task.

CWSP : Communication-Workspace. Von einer Task zur Durchführung von Ein- oder Ausgaben angeforderter Speicherbereich. Besitzer ist die Task. Sollte die Task terminiert werden, so geht der Speicherbereich bis zum Ende der I/O-Operation in den Besitz von RTOS-UH (Namensangabe **RTOS**) über.

MDLE : Modul. Datenbereich für permanente Daten, z.B. bei gemeinsamer Nutzung durch mehrere Tasks. Ein Modul hat einen eigenen Namen und ist selbst Besitzer des Speichersegments. Ein spezielles Modul mit dem Namen **#NORAM** signalisiert eine Lücke im Speicherausbau des Rechners. Auf den betroffenen Bereich kann nicht zugegriffen werden.

EDTF : Edit-File. Es handelt sich um ein Segment der Ramdisk. Als Besitzer ist der Dateiname angegeben.

FREE : Es handelt sich um freien Speicher.

PMDL : Prom-Modul. Ein bei der Erstellung PROM-residenter Programme entstehendes Speichersegment.

SMDL : Shell-Modul. Das Modul enthält einen in Hochsprache codierten Bedienbefehl. Die Generierung ist über PEARL möglich.

???? : Ein Speichersegment ist nicht mehr identifizierbar. Die im Speichersegment notierte Typ-Kennung wurde durch unerlaubte Operationen, Programmfehler o. ä. zerstört.

Tabelle 4-3: Typangabe für Speichersegmente

Im Beispiel sind nur Speicheranfang, Systemtasks (z.T. Autostart-fähig, aber alle mit der Eigenschaft **Resident**), freier Speicher und das Speicherende vorhanden. Während der weiteren Arbeit werden auch noch andere Segmente erscheinen. Der Bereich vor dem Speicheranfang ist mit RTOS-UH-internen Verwaltungsinformationen sowie dem **TWSP** der residenten Systemtasks belegt und steht nicht zur weiteren Verfügung.

Der Befehl **S** erlaubt durch Optionen eine Begrenzung der Ausgabe auf Segmente bestimmten Typs, s. RTOS-UH-Handbuch.

4.2.3 Die Taskzustände

Eine übersicht über die Zustände aller Tasks im System kann man durch den Befehl "L" (List Tasks) erhalten. Im Musterrechner sieht es jetzt so aus:

```
*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							Adr. des Taskkopfs
	Prio						Priorität (hex)
		User					USER, über den eine Aktivierung erfolgte
			Zustand				Zustand der Task
				TWSP-Adr.			Adr. des TWSP
					PC		Adr. des gerade ausgeführten Befehls
						Taskname	

Tabelle 4-4: Ausgabe der Taskliste

Die Priorität wird hexadezimal mit eingeschränktem Zahlenbereich ausgegeben.

Die Angabe der Adressen des Task-Workspace sowie des Programm-Counters der Task ermöglichen Insidern weitere Untersuchungen über die aktuelle Tätigkeit einer Task. Insbesondere kann hier unter Umständen beobachtet werden, daß eine Task zwar laufwillig ist (Zustand **RUN**), aber noch keinen **TWSP** und keinen **PC** zugeteilt erhalten hat. Eine derartige Task wartet noch auf die Zuteilung von **TWSP**.

Die hier erhaltene Liste erklärt sich aus dem Zusammenspiel der Systemtasks wie folgt: **IDLE** ist laufwillig (wie es auch sein sollte), **ACIA1** ebenso, da sie gerade mit der Ausgabe der Taskliste beschäftigt ist. **USER1** ist laufwillig, da er die Taskliste für die Ausgabe durch die **ACIA1** erstellen muß. Alle anderen Systemtask haben zur Zeit keine Aufgaben zu erfüllen und sind daher **DORM**. Lediglich die **ERROR** ist zur Behandlung von Fehlermeldungen eingeplant.

Sehr gut zu verfolgen ist auch die Prioritätenordnung der Systemtasks: die **ERROR** besitzt die höchste Priorität (-10), gefolgt vom **USER1**, der als **USER** an der Systemschnittstelle ausgezeichnet ist und eine höhere Priorität als der **USER2** besitzt. Die **ACIA**s und **SOUT**s sind auf der nächsten Prioritätsstufe angesiedelt, da sie als Betreuungstasks für die Ein- und Ausgabe der **USER** im System wichtiger sind als die Betreuungstasks für die restlichen Datenstationen.

Die Beobachtung der Taskzustände ist mit folgenden Befehlen möglich:

L	-	Ausgabe der Liste aller Tasks
LU	-	Ausgabe nur der Anwendertasks (List Usertasks)
SHOW	-	zur gezielten Betrachtung einer einzelnen Task (**SHOW** *Taskname*)

Tabelle 4-5: Befehle zur Ausgaben der Taskzustände

Der **L**-Befehl kann durch Optionen zur Ausgabe nur nach bestimmten Kriterien selektierter Tasks veranlaßt werden.

4.3 Einige Beispiele

Hier werden einige Beispiele für das Verhalten von Tasks unter RTOS-UH gegeben. Da die Erstellung von Programmen, insbesondere die Bedienung des Editors und Behandlung von Dateien, mit den bisherigen Erläuterungen noch nicht möglich ist, beschränkt sich dieses Kapitel auf Eingriffe über Bedienbefehle. Ziel ist die Erläuterung der Systembedienung einschließlich einiger Grundbefehle sowie die praktische Beobachtung der theoretischen Kenntnisse des Task-Verhaltens.

4.3.1 Eingabe von Befehlen

RTOS-UH erfordert vor der Eingabe eines Befehls den Start des Bedieninterpreters mit **^A**. Der aktivierte Bedieninterpreter meldet sich mit einem ***** als Prompt und akzeptiert die Eingabe von Befehlen.

Während der Befehlseingabe sind folgende Editier-Möglichkeiten gegeben:

- Cursor links, Backspace (**^H**) oder **DEL** (Delete) löschen ein Zeichen nach links.
- Cursor rechts (**^L**) restauriert das Zeichen an der aktuellen Position aus der vorhergehenden Eingabe.
- **CR**, **RETURN** oder **ENTER** (**^M**) sowie **LF** (**^J**) beenden die Eingabe.

Die Eingabe von Befehlen ist in Klein- oder Großbuchstaben, auch gemischt, möglich, es findet keine Unterscheidung der Schreibweise statt. Task-, Modul- und Dateinamen müssen allerdings unter Berücksichtigung der Groß-/Kleinschreibung angegeben werden.

Parameter werden von den Befehlen durch Leerzeichen getrennt.

Eine Befehlszeile kann max. 128 Zeichen lang sein.

Ein Befehl endet mit einem Semikolon **;** oder einem doppelten Bindestrich **--**, diese Endekennung kann entfallen, wenn in einer Zeile nur ein Befehl abgesetzt wird. Durch Semikolon getrennte Befehle werden vom System nach Möglichkeit parallel abgearbeitet (z. B. können so zwei Dateien gleichzeitig kopiert werden), durch **--** getrennte Befehle werden in zeitlicher Reihenfolge abgearbeitet (Befehlsverkettung). Hierbei werden jeweils folgende Befehle nicht bearbeitet, wenn im bei der Bearbeitung des vorigen Befehls ein Fehler aufgetreten ist. Beim Befehl **P--LOAD** (PEARL übersetzen mit anschliessendem Laden des Programms) wird daher der Ladebefehl nur ausge-

führt, wenn die Übersetzung fehlerfrei erfolgt ist. Zur Kontrolle der Ausgabe stehen folgende Zeichen zur Verfügung:

- X_{OFF} (^S) hält die Ausgabe an, d. h. auch die Ausgabe des Prompts nach demDrücken von ^A unterbleibt.
- X_{ON} (^Q) läßt sie wieder weiterlaufen.

Da X_{OFF} sehr leicht aus Versehen statt ^A gedrückt wird, kann in unklaren Situationen ein X_{ON} nie schaden. Eventuell ist dann der Zugang zum Bedieninterpreter schon wieder frei.

X_{OFF} und X_{ON} werden nie in den Eingabestring übernommen.

4.3.2 Erzeugung von Tasks

Tasks werden normalerweise durch Hochsprach- oder Assemblerprogrammierung erzeugt oder vom Bedieninterpreter automatisch zur Bearbeitung komplexer Aufgaben generiert. Mit dem **DEFINE**-Befehl kann der Bedieninterpreter allerdings auch zur direkten Taskerzeugung gezwungen werden. Der Befehl

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

hat folgende Wirkung:

Der Bedieninterpreter erzeugt eine Task mit dem Namen **X**, deren Aufgabe die Ausführung des anschliessenden (**--**, Befehlsverkettung) **SHOW**-Befehls ist. Diese Konstruktion ermöglicht die Definition feststehender, eigener Bedienbefehlsequenzen.

Intern erfolgt die Erzeugung der Task ebenso wie z. B. die Subtaskerzeugung beim **COPY**-Befehl. Der erzeugten Subtask wird durch den Zusatz **.Name** hinter dem Bedienbefehl ein vorgeschriebener Subtaskname gegeben. Vor dem Punkt dürfen Leerzeichen erscheinen. Wird die Namensgebung nicht vorgeschrieben, so generiert das System den Namen aus dem Bedienbefehl, gefolgt von einem **/** und einer zweistelligen, fortlaufend vergebenen Nummer. Nach Eingabe des Befehls **DEFINE.X -- SHOW X** antwortet das System mit der zum Befehl **SHOW** gehörenden Ausgabe. Mit **L** und **LU** kann festgestellt werden, daß die Task **X** nun als Anwendertask im System vorhanden ist.

4.3.3 Zeitliche Einplanungen

Mit dieser Task **X** können nun einige Beispiele für Aktivierungen und Einplanungen durchgeführt werden. Mit dem Befehl

```
X
```

(kurz für **ACTIVATE X**) wird **X** aktiviert, und die Ausgabe erscheint erneut. Mit dem Befehl

```
ALL 5 SEC X
```

(kurz für **ALL 5 SEC ACTIVATE X**) wird **X** zyklisch eingeplant. **X** wird sofort und folgend im 5 Sekunden-Rhythmus aktiviert, erkennbar an der erscheinenden Ausgabe. Zwischen den Ausgaben kann mit dem **LU**-Befehl festgestellt werden, daß **X** im Zustand **SCHD**, eingeplant, ist.

Trotzdem kann **X** auch asynchron von Hand gestartet werden: Die Eingabe von **X** als Befehl führt nur zu einer zusätzlichen Aktivierung und Ausgabe, ändert aber nichts am Rhythmus der regelmäßigen Ausgaben. Abgestellt werden kann dieser Poltergeist nur durch

```
PREVENT X,
```

eine Ausplanung der Task **X**.

Eine schwierigere Einplanung läßt sich mit dem Befehl

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

beobachten: nachdem 10 Sekunden nichts passiert, erscheinen 6 Ausgabezeilen, und daraufhin herrscht wieder Ruhe. Mit **LU** findet man **X** im Zustand **DORM**, sie hat ihre Aufgabe erfüllt, wurde selbstständig ausgeplant und schläft nun friedlich. Versucht man, **X** mit dem Befehl **x** zu wecken, sieht man auch gleich die Reaktion auf Fehleingaben: das System findet keine Task mit dem Namen **x** und beschwert sich mit

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

Fehlermeldungen beginnen mit **>>**, gefolgt vom Tasknamen des Verursachers, und geben die Fehlerursache, hier die Eingabe von **x** als unbekanntes Kommando, an. Wird **X** schneller eingeplant (**ALL 1.0 SEC X**) und die Ausgabe mit X_{OFF} angehalten, so kann die Fähigkeit von RTOS-UH, Taskaktivierungen zu puffern, beobachtet werden. Wird X_{ON} innerhalb von 5 sec nach dem X_{OFF} gegeben, so erscheinen sofort Ausgabezeilen. Deren Anzahl entspricht der Anzahl der Sekunden, für die die Ausgabe blockiert war. Da **X** seine Ausgabe nicht beenden konnte, hat die entsprechende Anzahl von Aktivierungen zwar stattgefunden, konnte aber nicht ausgeführt werden, da die erste Aktivierung nicht beendet wurde (kein Übergang von **RUN** nach **SCHD**). Diese Aktivierungen wurden gepuffert, und werden bei der Freigabe der Ausgabe nachgeholt. Besonders gut zu erkennen ist dies bei einer Task, die die aktuelle Uhrzeit ausgibt:

```
DEFINE.Y—CLOCK,
```

da hier die Uhrzeit des tatsächlichen Tasklaufs ausgegeben wird.

Wird die Ausgabe länger blockiert, so erscheint bei Freigabe der Ausgabe eine Zeile mit der Ausgabe der ersten Aktivierung, dann eine Fehlermeldung (ggf. auch mehrere)

```
>> X:INTRPT OVERFLOW (ACT).
```

und dann die Ausgaben der restlichen, gepufferten Aktivierungen. Die Fehlermeldung gibt an, das der Aktivierungspuffer der Task **X** übergelaufen ist, verursacht von einer Interrupt-Routine (hier dem Clock-Tick).

4.3.4 Interrupt-Einplanungen

RTOS-UH kennt 32 unterschiedliche Interruptquellen, die durch die Bezeichnung **EV** ***xxxxxxxx*** (**EV** wie engl. Event, dt. Ereignis) benannt werden. ***xxxxxxxx*** ist hierbei eine hexadezimal notierte Bitmaske, in der der gemeinte Interrup mit einer **1**, alle anderen mit einer **0** notiert werden. Als Beispiel soll hier der Interrupt **EV 00080000** dienen.

Die Task **X** kann mit

```
WHEN EV 00080000 activate X
```

aus dem Zustand **DORM** in den Zustand **SCHD** gebracht werden. Mit der simulierten Auslösung des Interrupts durch

```
TRIGGER EV 00080000
```

passiert allerdings noch gar nichts. RTOS-UH sperrt bei der Initialisierung nämlich das Auftreten der Interrupts. Erst nachdem der Interrupt mit

```
ENABLE EV 00080000
```

freigegeben wurde, kann **X** mit dem **TRIGGER EV 00080000**-Befehl gestartet werden. **X** wird bei jedem Auftreten des Interrupts neu aktiviert. Die Einplanung kann durch **PREVENT X** wieder gelöscht werden. Die Aktivierung wird jedoch auch schon durch die Interruptsperre

```
DISABLE EV 00080000
```

verhindert.

4.3.5 Aussetzen und Fortführen

Diese Operationen können an einer Endlosschleife der Art

```
DEFINE.endlos—endlos
```

mit den Befehlen **SUSPEND** (kurz **SU**) und **CONTINUE** (kurz **C**) gut beobachtet werden. Ganz nebenbei wird hier auch der Sinn der hohen Prioritäten für **ACIA** und **USER** klar: obwohl die Task **endlos** auf der Priorität -1 in einer Endlosschleife läuft, können noch Bedienbefehle abgesetzt werden. Der Rechner verhält sich für die höher priorisierten Tasks so, als ob die niedrigeren Prioritätsstufen gar nicht existierten.

Nach einem

```
T endlos
```

(Terminieren der Task **endlos**) kann dies noch besser mit Hilfe der Task **Niedrig** aus

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

(die Task Niedrig erhält die Priorität 10) und der Task **endlos** gezeigt werden. Wird **endlos** aktiviert, während **Niedrig** läuft, so verschwinden zunächst die Ausgaben von **Niedrig**, da sie keine Prozessorkapazität mehr erhält. **Niedrig** wird jedoch weiterhin regelmäßig aktiviert, und daher erscheinen nach ca. 10 sec Fehlermeldungen, die auf das Überlaufen des Aktivierungspuffers von **Niedrig** hinweisen. Der Effekt ist hier identisch zum Anhalten der Ausgabe im Kapitel "Zeitliche Einplanungen".

5 Das I/O-System

Das I/O-System von RTOS-UH unterscheidet sich vom Konzept her wesentlich von anderen Betriebssystemen. Zur Erläuterung wird hier zunächst der Grundgedanke, der hinter der Gestaltung des I/O-Systems steht, geschildert. Darauf folgt eine detailiertere Erläuterung des Funktionsprinzip. Obwohl hier zum Teil sehr ins Detail gegangen wird, ist ein Studium dieser Beschreibung zum Verständnis der Möglichkeiten, die RTOS-UH bietet, sehr wichtig. Eine Kurzbeschreibung der wichtigsten, in einem RTOS-UH-System vorhandenen Geräte und Datenstationen schließt dieses Kapitel ab.

5.1 Warteschlangen

RTOS-UH koppelt die Ausführung von E/A-Operationen von den veranlassenden Tasks ab. Eine Task, die eine Ein- oder Ausgabe durchführen möchte, fordert sich vom Betriebssystem zunächst Speicher für E/A-Operationen an, den sog. Communication-Workspace (**CWSP**). Ein derartiges Speichersegment wird auch Communication-Element, CE, genannt. In dieses CE trägt die Task ein, auf welchem Gerät und ggf. welcher Datei welche E/A-Operation durchgeführt werden soll, und übergibt das ausgefüllte CE dem Betriebssystem.

Das Betriebssystem reiht dieses CE in die Warteschlange des betroffenen Gerätes ein. Die Einordnung erfolgt in der Reihenfolge der Prioritäten der veranlassenden Tasks, bei gleicher Priorität in der Reihenfolge des Eintreffens. Gleichzeitig aktiviert es die zu diesem Gerät gehörende Betreuungstask, falls diese noch nicht aktiv ist.

Die Betreuungstask entnimmt daraufhin der Warteschlange ein CE und bearbeitet es. Nach der Bearbeitung wird es dem Betriebssystem zurückgegeben. Ist die veranlassende Task an dem Resultat der E/A-Operation interessiert, gibt das Betriebssystem dieses CE der Task zurück, andernfalls wird der benutzte Speicherbereich automatisch wieder zu freiem Speicher erklärt. Die Betreuungstask terminiert sich daraufhin, falls kein weiteres CE in ihrer Warteschlange vorhanden ist.

Wird eine Task, die noch nicht abgeschlossene Ein- oder Ausgaben veranlasst hat, terminiert, so werden deren noch in Warteschlangen befindlichen Eingabe-CE‘s sofort aus der Warteschlange genommen und zu freiem Speicher erklärt. Schon in der Bearbeitung durch eine Betreuungstask befindliche CE's sowie Ausgabe-CE's werden im System belassen und zu Ende bearbeitet.

Die Ausführung der E/A-Operation ist so von der Abarbeitung der veranlassenden Task abgekoppelt. Eine Task kann eine Ausgabe veranlassen, indem sie die Ausgabedaten in einem CE bereitstellt und diese CE dem Betriebssystem übergibt. Hat die Task kein weiteres Interesse am Resultat der Ausgabe (es können ja z.B. Fehler auftreten), so kann sie sofort nach der Übergabe weiterlaufen. Gerade die Zeiten, in denen eine Betreuungstask ihre Ausführung aussetzt, können so auch von der veranlassenden Task genutzt werden.

Ähnlich verhält es sich bei Eingaben: eine Task kann eine Eingabeanforderung absetzen, anschliessend in ihrer Abarbeitung fortfahren, und erst später Interesse an den eingelesenen Daten zeigen. Sollten die Daten zu diesem Zeitpunkt allerdings noch nicht zur Verfügung stehen, so wird die Task in den Zustand **I/O?** versetzt und in ihrer Ausführung bis zum Ende der E/A-Operation ausgesetzt.

Der Befehl **COPY** nutzt diese Fähigkeit aus, indem er mit zwei CE's einen Ping-Pong-Betrieb durchführt: Ein CE fordert eine Eingabe von der Kopierquelle an, ein weiteres CE wird für die Ausgabe auf das Kopierziel benutzt. Sowohl die Eingabe- als auch die Ausgabeanforderungen können nun vom System gleichzeitig bearbeitet werden. Die generierte Subtask **COPY/*xx*** dreht nach er-

folgter Ein- und Ausgabe lediglich die Datenrichtung der CE's um, so daß die erhaltene Eingabe als Ausgabe weitergeleitet und das alte Ausgabe-CE nun für die Eingabe genutzt wird.

5.2 LDN's, Drives und Gerätenamen

RTOS-UH verwaltet für jedes E/A-Gerät eine eigene Warteschlange. Betriebssystemintern wird jede Warteschlange durch eine Nummer, die **LDN** (**L**ogical **D**evice **N**ame, logischer Gerätename) gekennzeichnet. Besitzt ein Gerät noch eigene Untergliederungsmöglichkeiten (z.B. kann das Gerät *Massenspeicher* unterschiedliche Laufwerke verwalten), so kann, ggf. muß, noch eine Laufwerksnummer, der sog. Drive, als genaue Spezifikation angefügt werden. RTOS-UH verwaltet die einzelnen Drives jedoch nicht in eigenen Warteschlangen. Die Unterscheidung nach Drives muß in der Betreuungstask erfolgen.

Zur Vereinfachung der Systembedienung hält das Betriebssystem für viele **LDN**-Drive-Kombinationen symbolische Gerätenamen bereit. Bei der Eingabe von Bedienbefehlen kann so eine bestimmte **LDN**-Drive-Kombination durch die Angabe lediglich des Gerätenamens gekennzeichnet werden. Die in einem RTOS-UH-System verfügbaren **LDN**-Drive-Kombinationen können mit dem Befehl **HELP -D** angezeigt werden. Bei dem Musterrechner erscheint daraufhin folgende Ausgabe:

```
*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
```

In dem System sind enthalten:

LDN	Drive	Gerätename	Funktion	Betreuungstask
0	0	**/A1/**	Systemschnittstelle	**#ACIA1**
0	2	**/B1/**	dto., andere Betriebsart	"
0	6	**/C1/**	dto., andere Betriebsart	"
1	0	**/ED/**	Ramdisk, ASCII-Daten	**#EDFM**
1	1	**/EDB/**	Ramdisk, binäre Daten	"
2	0	**/A2/**	2. ser. Schnittstelle	**#ACIA2**
2	2	**/B2/**	dto., andere Betriebsart	"
2	3	**/UL/**	dto., andere Betriebsart	"
2	6	**/C2/**	dto., andere Betriebsart	"
7	0	**/VO/**	virt. Datenstation, Ausgabe	**#VDATN**
8	0	**/VI/**	virt. Datenstation, Eingabe	"
9	0	**/XC/**	Bedieninterpreter	**#XCMMD**
10	0	**/PP/**	Centronics-Schnittstelle	**#PPROT**

11	0	**/D1/**	Systemschnittstelle, Duplex	**#SOUT1**
12	0	**/D2/**	2. ser. Schnitttelle, "	**#SOUT2**
15	0	**/NIL/**	ideale Datensenke/quelle	**#NIL**

Tabelle 5-1: Zuordnung LDN/Drive und Mnemos der Datenstationen

Die weiterhin angegebenen Geräte **/TY/**, **/TYB/**, **/TYC/** und **/NO/** haben Sonderfunktionen. Ihre direkte Benutzung ist nicht mögliche, ihre Bedeutung wird später erklärt.

Bei der Eingabe von Bedienbefehlen können Geräte sowohl über die mnemonischen Kürzel (z.B. **/A1/**) oder über die Angabe von **LDN** und Drive (z. B. **/LD/0,0/** für **/A1/** oder **/LD/0,2/** für **/B1/**) spezifiziert werden.

5.3 Aufbau und Benutzung von CE's

In diesem Kapitel wird der direkte Umgang mit CE's geschildert, um die Möglichkeiten und Fähigkeiten dieses I/O-Konzepts zu erläutern. Obwohl die Beispiele hier eher nur für Assembler-Programmierer interessant sind (Hochsprach-Programmierern wird diese Arbeit durch ein der Hochsprache angepaßtes Laufzeitsystem abgenommen), sollten sie dennoch gelesen werden, da das Verhalten der Laufzeitunterstützung für Hochsprach-Programme nur dadurch verständlich wird.

5.3.1 Anforderung eines CE's

Ein CE wird vom Betriebssystem über den Trap **FETCE** angefordert. Hierzu muß die Größe des gewünschten Datenpuffers im Prozessorregister D1 eingetragen werden. Der Trap besorgt sich vom Betriebssystem ein Speichersegment vom Typ **CWSP** in der erforderlichen Größe und parametriert daß CE vor. Nach der Rückkehr aus dem Trap zeigt das Prozessorregister A1 auf dieses Speichersegment. Das Speichersegment hat folgenden Aufbau (Offset in Byte relativ zu A1, Name entsprechend den RTOS-UH-Konventionen):

Offset	Name	Funktion
0 (0x00)	**FORL**	Vorwärtszeiger der Speicherverwaltung, zeigt auf das nächste Speichersegment
4 (0x04)	**BACKL**	Rückwärtszeiger der Speicherverwaltung, zeigt auf das vorhergehende Speichersegment
8 (0x08)	**OWNER**	Nummer des veranlassenden **USER**s
9 (0x09)	**TYPE**	Typkennzeichnung des Speichersegmentes
10 (0x0A)	**FORT**	Vorwärtszeiger für Verkettung der Taskzugehörigkeit
14 (0x0E)	**BACKT**	Rückwärtszeiger für Verkettung der Taskzugehörigkeit
18 (0x12)	**TIDO**	TID der veranlassenden Task
22 (0x16)	**FORS**	Vorwärtszeiger für Warteschlangenverkettung Steht auf 1, wenn das CE in der Bearbeitung durch eine Betreuungstask ist.
26 (0x1A)	**BACKS**	Rückwärtszeiger für Warteschlangenverkettung

30 (0x1E)	PRIO	Priorität des CE's (für Warteschlange)
32 (0x20)	BUADR	Zeiger auf den Datenbereich des CE's
36 (0x24)	RECLEN	bei Ausgabe: Anzahl der auszugebenden Bytes bei Eingabe: Länge des verfügbaren Datenpuffers
38 (0x26)	STATIO	Statusinformationen über das CE
39 (0x27)	LDN	Nummer der Warteschlange, für die das CE bestimmt ist
40 (0x28)	MODE	Kodierung der Art des I/O-Auftrags
42 (0x2A)	DRV	Dauer eines Time-Outs für die Bearbeitung dieses CE's (nur bei geeigneten Betreuungstasks sinnvoll)
43 (0x2B)	DRIVE	Nummer des Drives, für das das CE bestimmt ist
44 (0x2C)	FNAME	Dateiname, für den das CE bestimmt ist (wird nicht von allen Betreuungstasks ausgewertet)
	IOBUF	Datenpuffer. Die Anzahl der verfügbaren Bytes wird beim Aufruf von FETCE bestimmt. BUADR zeigt auf diesen Datenbereich.

Tabelle 5-2: Bedeutung der einzelnen Felder eines CE

Der Anwender kann hierbei auf die Elemente ab PRIO Einfluß nehmen. Die Elemente bis einschl. BACKS dienen der systeminternen Verwaltung und dürfen nicht benutzt werden.

Ein mit FETCE beschafftes CE wurde vom System wie folgt parametriert:

- PRIO wurde mit der Priorität der anfordernden Task besetzt
- BUADR zeigt auf IOBUF
- STATIO enthält 0
- FNAME enthält 8 Leerzeichen

5.3.2 Ausfüllen eines CE's

Bevor ein CE genutzt werden kann, müssen alle für die Ein- oder Ausgabe wichtigen Daten eingetragen werden. Die Bedeutung der einzelnen Elemente sowie die möglichen Eintragungen werden im folgenden erläutert.

Name	Bedeutung des Feldinhalts
PRIO	Die Eintragung der CE's in eine Warteschlange erfolgt in der Reihenfolge der PRIO der CE's. Standardmäßig wird hier die Priorität der veranlassenden Task benutzt (Vorbesetzung). Diese Angabe kann umgesetzt werden.
BUADR	BUADR zeigt auf den Datenbereich des CE's, nach der Anforderung mittels FETCE also auf IOBUF, den Bereich hinter FNAME. Dort stehen so viele Bytes wie angefordert zur Verfügung. BUADR kann umgesetzt werden. Stehen Ausgabedaten schon fertig zur Verfügung, kann BUADR so gesetzt werden, das er auf diese Daten zeigt. In diesem Fall kann auch die Anforderung eines CE's mit einer Datenfeldlänge 0 sinnvoll sein.
RECLEN	Gibt die Anzahl der Datenbytes an. Bei Ausgaben muß RECLEN auf die Maximalzahl der auszugebenden Bytes gesetzt werden, bei Eingaben auf die Maximalzahl der einzulesenden Bytes.

Die tatsächliche Anzahl der gelesenen bzw. geschriebenen Bytes wird zusätzlich von **MODE** beeinflußt.

STATIO Im Statusbyte des CE's stehen nur die Bits 1 (Maske 0x02) und 3 (Maske 0x08) zur Verfügung. Bit 1 ist das sog. Verschrottungsbit (**STABRE**), wird dieses Bit gesetzt, so wird das CE nach beendeter Ein- oder Ausgabe sofort zu freiem Speicher erklärt und nicht der veranlassenden Task zurückgegeben (kann bei Ausgaben sinnvoll sein).

Bit 3 hat keine definierte Funktion und kann frei verwendet werden.

LDN Hier muß die Warteschlangennummer des Zielgerätes eingetragen werden.

MODE Gibt spezifische Anweisungen für die Ausführung des I/O-Auftrags an. Die wesentlichen Punkte werden in der folgenden Tabelle erläutert.

DRV Enthält die Dauer eines Time-Outs für die Ein- oder Ausgabeoperation in Vielfachen einer betreuungstaskspezifischen Grundzeit. Nach Ablauf des Time-Outs wird der E/A-Vorgang mit einer Fehlermeldung abgebrochen. Diese Angabe wird nur von einigen Betreuungstasks ausgewertet. Für Betreuungstasks, die kein Time-Out kennen, sollte DRV zu 0 gesetzt werden.

DRIVE Muß die ggf. mögliche Laufwerksunterteilung eines Gerätes enthalten.

FNAME Bei Geräten, die dateiorientiert arbeiten, muß hier der Pfad der Datei angegeben werden. Die Pfadangabe muß mit **0xFF** beendet werden. Unter **FNAME** ist ausreichend Platz für jede unter RTOS-UH zulässige Pfadangabe vorhanden, z.Zt. max. 24 Zeichen). Die Gerätebezeichnung darf nicht mit angegeben werden, da sie schon in **LDN** und **DRIVE** kodiert ist.

Betreuungstasks, die keine Dateien verwalten, ignorieren eine evtl. Dateiangabe.

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.
- Die niederwertigen 5 Bit kodieren 32 unterschiedliche Aufträgen.

Die am häufigsten benötigten Aufträge sind:

	Kürzel	Auftrag
0	**READ/WRITE OLD**	Lesen oder Schreiben einer schon vorhandenen Datei. Wird versucht, in eine noch nicht vorhandene Datei zu schreiben, so wird dies als fehlerhaft abgewiesen. Ist die Datei noch nicht geöffnet, so wird sie geöffnet und auf den Dateianfang positioniert. Eine weitere Positionierung erfolgt nicht. Wird beim Lesen das Dateiende überlaufen, so wird sie in Abhängigkeit vom Bit **MODMSC** von **MODE** (s.u.) ggf. automatisch geschlossen. Wird am Dateiende weitergeschrieben, so wird die Datei verlängert. Kennt das betroffene Gerät keine Dateiverwaltung, wird die Dateianga-

		be ignoriert.
6	**CLOSE**	Schließen einer Datei. Eventuell unter **BUADR** angegebene Daten werden ignoriert. Die Datei wird geschlossen. Sollte nur als Eingabe-CE durchgeführt werden.
7	**READ/WRITE ANY**	Lesen oder Schreiben einer Datei. Ist die betroffene Datei noch nicht vorhanden, so wird sie neu angelegt und auf den Dateianfang positioniert. Die weiteren Bemerkung aus Nr. Auftrag 0, **READ/WRITE OLD**, gelten hier entsprechend.
8	**REWIND OLD**	Positionieren einer schon vorhandenen Datei auf den Dateianfang. Ist die Datei noch nicht vorhanden, wird eine Fehlermeldung erzeugt. Sollte nur als Eingabe-CE durchgeführt werden.
9	**APPEND**	Ausgaben werden an eine bestehende Datei angefügt. Vor diesem Schreibauftrag wird an das Dateiende positioniert. Die Datei wird durch diesen Schreibauftrag verlängert. Nur als Schreibauftrag möglich.
21	**REWIND ANY**	Positionierung auf den Dateianfang. Ist die Datei noch nicht vorhanden, wird sie vorher angelegt. Sollte nur als Leseauftrag durchgeführt werden.
22	**REWIND NEW**	Erzeugung einer neuen Datei. Ist die Datei schon vorhanden, erfolgt eine Fehlermeldung. Sollte nur als Leseauftrag durchgeführt werden.

Tabelle 5-4: Auftragsnummern für I/O-Aufträge

Die weiteren Auftragsnummern werden bei den entsprechenden Betreuungstasks erläutert. Erhält eine Betreuungstask Auftragsnummern, die sie nicht kennt, erzeugt sie eine Fehlermeldung.

Die Einzelbit-Parameter sind in MODE wie folgt abgelegt (Bitnummer 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.	Kürzel	Funktion
MODMWA	Wait-Bit	Ist dieses Bit gesetzt, so wird die veranlassende Task bis zum Ende der Bearbeitung des I/O-Auftrages in den Zustand **I/O?** versetzt und bei der Prozessorzuteilung nicht berücksichtigt. Nicht sinnvoll mit gesetztem Bit 1 von **STATIO**.
MODMOU	Output-Bit	Wird dieses Bit gesetzt, so wird das CE als Ausgabe-CE be-

		trachtet; ist das Bit nicht gesetzt, erfolgt eine Eingabe.
MODMCR	End on **CR**	Ist diese Bit gesetzt, so wird eine Aus- oder Eingabe vor Erreichen von **RECLEN** beendet, falls ein **CR (^M**) ausgegeben oder eingelesen wurde. Das **CR** wird mit übertragen. Ist das Bit nicht gesetzt, erfolgt keine Sonderbehandlung für **CR**.
MODMLF	End on **LF**	Ist diese Bit gesetzt, so wird eine Aus- oder Eingabe vor Erreichen von **RECLEN** beendet, falls ein **LF** (**^J**) ausgegeben oder eingelesen wurde. Das **LF** wird mit übertragen. Ist das Bit nicht gesetzt, erfolgt keine Sonderbehandlung für **LF**.
MODMEO	End on **EOT**	Ist diese Bit gesetzt, so wird eine Aus- oder Eingabe vor Erreichen von **RECLEN** beendet, falls ein **EOT** (**^D**) ausgegeben oder eingelesen wurde. Das **EOT** wird mit übertragen. Ist das Bit nicht gesetzt, erfolgt keine Sonderbehandlung für **EOT**. **EOT** dient unter RTOS-UH als Dateiendekennung.
MODMSC	Suppress-command	Ist dieses Bit gesetzt, führen die Dateiverwaltungen von RTOS-UH kein automatisches Schliessen einer Datei bei Überlesen des Dateiendes durch, und die seriellen Schnittstellen gestatten keinen Zugang zum Bedieninterface mehr über **^A** und **^B**.
MODMNE	No echo	Nur bei seriellen Schnittstellen: bei gesetztem Bit wird kein automatisches Echo der Eingabezeichen erzeugt.
MODBIN	Binär-Bit	Nur bei seriellen Schnittstellen: bei gesetztem Bit wird binärer Datentransfer durchgeführt, d.h. das oberste Datenbit wird nicht zwangsweise auf 0 gesetzt und das X_{ON}/X_{OFF}-Protokoll ist abgeschaltet (nur noch Hardware-Handshake über RTS/CTS möglich).
IOCEF		Nicht für die CE-Parametrierung sondern bei Rückerhalt des CE's nach erfolgter Eingabe: Bei einem Lesevorgang wurde das Dateiende errreicht.
NERR	No-error-messages	Bei gesetztem Bit erzeugen Betreuungstasks keine Fehlermeldungen auf dem Bildschirm, sondern tragen eine Fehlernummer in **RECLEN** ein.
EXCLU	Exclusive-Bit	Ist diese Bit gesetzt, so soll der Schreib- oder Lesezugriff exklusiv, d.h. andere Tasks ausschliessend, erfolgen. Sobald eine Task eine Datei exklusiv benutzt hat, werden Lese- oder Schreibzugriffe anderer Tasks nicht mehr zugelassen, bis die Task die Datei geschlossen hat.

Tabelle 5-6: Bedeutung der MODE-Bits

5.3.3 Ausführung der Ein- oder Ausgabe

Nach der Parametrierung kann das CE dem Betriebssystem mit Hilfe des Traps **XIO** zur Bearbeitung gegeben werden. Die Adresse des CE's muß hierzu vor dem Trap-Aufruf im Prozessorregister A1 eingetragen werden (Ausgaberegister von **FETCE**). RTOS-UH unternimmt nun alle weiteren erforderlichen Schritte. Modifikationen an dem CE dürfen jetzt nicht mehr durchgeführt werden! In Abhängigkeit vom Wait-Bit (**MODE** Bit 15) wird die veranlassende Task nun in ihrer Ausführung

ausgesetzt (Zustand **I/O?**, bei gesetztem Wait-Bit) oder kann ungehindert weiterlaufen (Wait-Bit 0).

Wurde das **STABRE**-Bit (Bit 1 von **STATIO**) gesetzt, so sind von nun an keine weiteren Operationen mit dem CE zulässig. Das CE wird nach erfolgter Ein- oder Ausgabe vom Betriebssystem automatisch vernichtet, d.h. der **CWSP** wieder zu freiem Speicher erklärt.

5.3.4 Auswertung und Freigabe eines CE's

Wurde das Wait-Bit eines CE's nicht gesetzt, so kann die veranlassende Task mit dem Trap **IOWA** das Ende der Bearbeitung eines CE's abwarten. Die Adresse des CE's muß hierzu vor dem Trap-Aufruf im Prozessorregister A1 eingetragen werden (Ausgaberegister von **FETCE**). Die veranlassende Task wird in den Zustand **I/O?** gebracht, falls die Bearbeitung des CE's noch nicht beendet ist. Nach Bearbeitung eines CE's durch eine Betreuungstask können dem CE folgende Informationen entnommen werden:

- Eingabe-CE

  BUADR zeigt auf die eingelesenen Zeichen.

  Es wurden **RECLEN** Zeichen eingelesen.

  Wurde beim Einlesen das Dateiende erreicht (genau!, kann mit End-on-EOT erreicht werden), so ist **IOCEF** in **MODE** gesetzt.

- Ein- und Ausgabe-CE

  Ist **RECLEN** 0 oder negativ, so ist die I/O-Operation nicht erfolgreich durchgeführt wurden. Im Fehlerfall enthält **RECLEN** bei gesetztem **NERR** im **MODE** eine Fehlernummer (s. Handbuch S. C-IV-6 ff) mit gesetztem höchstwertigem Bit, bei nicht gesetztem **NERR** wurde schon eine Fehlermeldung von der Betreuungstask ausgegeben, und **RECLEN** enthält 0.

  Das CE kann jetzt entweder neu verwendet oder mit dem Trap **RELCE** dem System zurückgegeben werden. Nach Absetzen des Traps **RELCE** (A1 muß auf das CE zeigen !) ist der Speicherbereich des CE's wieder zu freiem Speicher erklärt und darf nicht weiter verwendet werden.

5.4 Beispiel

Die Parametrierung und das Verhalten der CE's kann an dem Musterrechner sehr einfach beobachtet werden. Mit dem Befehl

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

wird eine Kopier-Subtask (**CP**: Kurzform von **COPY**) erzeugt, die von der 2. ser. Schnittstelle in die Datei **MIST** der Ramdisk kopiert. Da an der Schnittstelle keine Datenquelle angeschlossen ist, erfolgt kein Datentransfer, und die erzeugten CE's lassen sich mit dem Befehl

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

im Speicher dingfest machen. **S-C** ist der Befehl **S** mit der Option **-C**, die die alleinige Anzeige von **CWSP** gestattet.

Die beiden CE's können mit dem **DM**-Befehl (engl. Dump Memory, dt. Speicherausgabe) genauer betrachtet werden. Das erste CE ergibt sich zu

```
*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 ................
```

Die Ausgabe des **DM**-Befehls besteht in einer Zeile aus Startadresse des Speicherauszuges und dem Inhalt des Speicherbereiches von Startadresse bis einschl. Startadresse+15, einmal in hexadezimaler und einmal in ASCII-Notation.

Das zweite CE ergibt

```
*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........
```

Eine Übertragung der Werte in die Struktur eines CE's ergibt folgende Inhalte:

CE-Element	CE1	Bedeutung	CE2	Bedeutung
FORL	1FDBA	Die Speicherverkettung kannmit S überprüft werden	1FE9E	
BACKL	358A		1FCF4	
OWNER	00	ist stets um 1 kleiner als die Angabe x in USERx	00	
TYPE	04	entspricht CWSP	04	
FORT	1FDBA	RTOS-UH-intern verwendet	1FE9E	
BACKT	1FE9E	"	1FCF4	
TIDO	1FF44	mit LU als TID erkennbar	1FF44	
FORS	0	nicht in Bearbeitung	1	in Bearbeitung durch Betreuungstask
BACKS	C20	systemintern	C2A	
PRIO	14	entspricht Task-Priorität	14	
BUADR	1FD38	zeigt auf jeweiligen IOBUF	1FDFE	
RECLEN	2	z.Zt. ohne Bedeutung, da CE gerade unbenutzt	2	Eingabe nicht beendet
STATIO	0		0	
LDN	1	/ED/	2	/A2/
MODE	8015	REWIND ANY, Eingabe m. WAIT (von vorheriger Nutzung beim Anlegen der Datei MIST)	C007	READ/WRITE ANY, Eingabe mit WAIT u. End on CR
DRV	0	kein Time-Out	0	
DRIVE	0	Drive 0	0	
FNAME	MIST	wie beim Befehl angegeben	-	Standard bei fehl. Angabe
IOBUF	-leer-	(noch nicht benutzt)		LF u. =

Tabelle 5-7 : Inhalt der Beispiel-CE's

Die nicht benutzten Stellen des CE's, insbesondere der Rest von FNAME und der komplette IOBUF, können beliebige Werte aus vorheriger Speichernutzung aufweisen.

Mit den nun bekannten Inhalten eines CE kann bei unklaren Situationen ein eigenes Studium des Zustands von I/O-Operationen erfolgen.

5.5 Geräte und Datenstationen

Hier werden nur kurze Hinweise über die wichtigsten Geräte in einem RTOS-UH-System gegeben. Detailierte Beschreibungen folgen in einem späteren Kapitel, lediglich die seriellen Schnittstellen werden hier ausführlich beschrieben.

5.5.1 Geräteparameter

Jedes in einem RTOS-UH-System vorhandene Gerät stellt einen kurzen Parameterblock zur Beschreibung seiner Fähigkeiten zur Verfügung. Dieser kann mit dem Befehl

`DD` *`Gerätenamen`*

betrachtet werden. **DD** bedeutet engl. "Dump Device", dt. "Ausgabe Gerät"

Der Befehl DD gibt die Adresse des Parameterblocks sowie 16 Bits, die die einzelnen Gerätefunktionalitäten charakterisieren, aus. Die einzelnen Bits haben folgende Bedeutung:

Maske	Bedeutung (bei gesetztem Bit)
0x8000	Gerät erlaubt Positionieren auf Dateianfang.
0x4000	Vor der Benutzung muß ein Öffnen (unter RTOS-UH nur als REWIND möglich) einer Datei erfolgen, nach der Benutzung ein Schließen.
0x2000	Beendet ein CR eine Zeile, so soll es durch CR+LF ersetzt werden (nur bei Ausgaben auf dieses Gerät).
0x1000	Das Gerät ist dialogfähig (ser. Schnittstelle, Terminal).
0x0800	Bei ser. Schnittstellen: es soll kein Echo erzeugt werden.
0x0400	Löschen von Dateien ist möglich (Befehle **RM** und **ERASE**).
0x0200	Über das Gerät ist Datenausgabe möglich.
0x0100	Über das Gerät ist Dateneingabe möglich.
0x0080	Das Gerät kann eine Liste der vorhandenen Dateien liefern (der Befehl **DIR** ist zugelassen).
0x0040	Das Gerät ist formatierbar (Der Befehl **FORM** wird akzeptiert).
0x0020	Das Gerät akzeptiert den Befehl **CF**.
0x0010	Das Gerät kennt Unterverzeichnisse (die Befehle **MKDIR** und **RMDIR** sind zulässig).
0x0008	Auf dem Gerät kann innerhalb von Dateien wahlfrei positioniert werden.
0x0004	reserviert.
0x0002	Für ser. Schnittstellen: Das Terminal macht bei Überschreitung der letzten Spalte einer Zeile keinen automatischen Übergang in die nächste Zeile (kein Auto-Wrap).
0x0001	Für ser. Schnittstellen: Cursorsteuerung soll für VT52 über ESC-Sequenzen erfolgen (Standard ist TELEVIDEO-Kompatibilität).

Tabelle 5-8: Geräteparameter - Die einzelnen Bits

Nach dem Systemstart sind diese Parameterblöcke mit Standardwerten versorgt. Diese Werte können zur Einstellung anderer Betriebsarten im Rahmen der Möglichkeiten der einzelnen Geräte mit dem Befehl

`SD` */Gerätename/ Parameter*

(engl. "Set Device", dt. "Gerät" setzen) geändert werden. Diese Parameter-Bits steuern nicht das Verhalten der einzelnen Geräte (Betreuungstasks), sondern dienen nur zur Information anderer Programme über die Art, wie das Gerät wünscht, behandelt zu werden. Die einzelnen Betreuungstask richten ihr Verhalten lediglich nach den Kontrollinformation, die sie mit einem CE erhalten.

Betriebssystemprogramme fragen diese Parameter ab und richten ihr Verhalten darauf ein, so macht ein **COPY** z.B. bei einer Ramdisk-Datei vor der Benutzung ein REWIND, nach der Benutzung ein CLOSE, bei einer ser. Schnittstelle jedoch nicht, entsprechend den bei dem jeweiligen Gerät gesetzten Parameter-Bits.

5.5.2 Serielle Schnittstellen /Ax/, /Bx/, /Cx/, /Dx/

Mit **/Ax/**, **/Bx/**, **/Cx/** und **/Dx/** werden die seriellen Schnittstellen eines RTOS-UH-Rechners bezeichnet. Betreuungstasks für eine Schnittstelle sind **#ACIA*x*** (**#SCC*x***, **#RS232**) und **#SOUT*x***, die Kennung ***x*** wird hexadezimal für die Zahl der vorhandenen Schnittstellen durchgezählt. Der Buchstabe **A**, **B**, **C** oder **D** kennzeichnet unterschiedliche Betriebsarten der gleichen Schnittstelle.

Eine ser. Schnittstelle **/Ax/** hat beispielhaft die Geräteparameter 3300, d. h. das Gerät ist ein dialogfähiges Datenterminal (0x1000) und erlaubt Aus- (0x0200) und Eingabe (0x0100). Weiterhin ist gewünscht, das Systemprogramme bei Ausgaben über dieses Gerät endende **CR** um ein **LF** ergänzen (0x2000). Das Echo von Eingabezeichen soll erfolgen. Die allgemeine Funktion einer ser. Schnittstelle unter RTOS-UH ist wie folgt:

5.5.2.1 Ausgabe

Die im CE enthaltenen Daten werden ausgegeben. Empfängt die ser. Schnittstelle während der Ausgabe ein X_{OFF}, oder wurde vor Beginn der Ausgabe ein X_{OFF} empfangen, so wird die Ausgabe bis zum Empfang eines X_{ON} angehalten. Die Blockierung über X_{OFF} kann durch den Bedienbefehl

SB ***Gerätename Baudrate***

(engl. "Set Baudrate", dt. "Baudrate setzen") aufgehoben werden. Unterstützt die RTOS-UH-Implementierung das RTS/CTS-Hardware-Protokoll, so wird der Zustand CTS High, d. h. Leitungspegel > 2.2. V, identisch zu einem empfangenen X_{OFF} behandelt.

Ausgaben über **/Dx/** können erfolgen, während über **/Ax/**, **/Bx/** oder **/Cx/** Eingaben aktiv, d.h. Eingabe-CE's in Bearbeitung, sind.

5.5.2.2 Eingabe

Eingaben können nur über die Schnittstellen mit den Kennungen A, B und C erfolgen. Die zuständige Betreuungstask ist **#ACIA*x***. Das Verhalten der Schnittstelle bei einer Eingabe richtet sich nach ihrer Kennung (aus dem **DRIVE** des CE) und dem **MODE** des CE's. Die Kennung ermöglicht folgende Unterscheidungen:

- Kennung A - ungepufferter Betrieb

  Das CE wird mit den Zeichen aufgefüllt, die nach dem Beginn der Bearbeitung des CE's durch die Betreuungstask eintreffen. Die Bearbeitung des CE wird nach dem Eintreffen von **RECLEN** Zeichen oder dem Empfang eines der Zeichen **CR**, **LF** oder **EOT** (je nach **MODMCR**, **MODMLF** und **MODMEO**) beendet. Es wird kein X_{OFF} gesendet. Der Empfangspuffer für den gepufferten Betrieb wird gelöscht.

- Kennung B - gepufferter Betrieb

  Das CE wird zunächst aus einem Empfangspuffer (Länge implementationsabhängig, meist 32 Zeichen), der auch in Abwesenheit eines CE's von der Interruptroutine genutzt wird, gefüllt. Ist der Empfangspuffer leer, wird ein X_{ON} zum Starten des Senders gesendet. Weitere Zeichen werden direkt dem Eingangsdatenstrom entnommen. Die Bearbeitung des CE wird nach dem Eintragen von **RECLEN** Zeichen oder dem Empfang eines der Zeichen **CR**, **LF** oder **EOT** (je nach **MODMCR**, **MODMLF** und **MODMEO**) beendet.

- Kennung C - gepufferter Betrieb, nur Puffer leeren

  Das Verhalten ist wie bei Kennung B, es wird jedoch nicht aus Zeichen aus dem Eingabedatenstrom gewartet. Das CE wird zunächst aus dem Empfangspuffer gefüllt. Wird hiermit schon **RECLEN** oder eine evtl. Abbruchbedingung erfüllt, so wird das CE zurückgegeben, andernfalls wird das CE bis zu **RECLEN** Zeichen mit 0 aufgefüllt und zurückgegeben. Es wird nicht auf das Eintreffen von Zeichen über die Schnittstelle gewartet.

Der unter dem Mnemo D ansprechbare Duplexkanal verdient an dieser Stelle keine besondere Erwähnung.

5.5.2.3 Ruhezustand

Das Verhalten der ser. Schnittstellen im Ruhezustand, d.h. wenn kein CE in Bearbeitung ist, richtet sich nach der Betriebsart der vorherigen Benutzung, d.h. nach dem **MODE** des vorherigen CE's, auch eines vorangegangenen Ausgabe-CE's. Daher sollten auch die Ausgaben mit der Kennung erfolgen, die für die Eingaben gewünscht ist.

Alle eintreffenden Zeichen werden in einen Empfangspuffer übertragen. Ist dieser Puffer voll, wird das letzte empfangene Zeichen überschrieben.

Wurde die Schnittstelle vorher im gepufferten Betrieb benutzt, so wird, beginnend bei halber Pufferfüllung, für jedes empfangene Zeichen ein **X$_{OFF}$** zum Anhalten des Senders gesendet.

War die Schnittstelle in binärem Betrieb genutzt, wird kein **X$_{OFF}$** gesendet; unterstützt die Schnittstelle das RTS/CTS-Protokoll (implementationsabhängig), wird statt dessen RTS auf LOW (Leitungspegel < 0.8 V) gelegt.

5.5.2.4 Der Einfluß von MODE

Die Bedeutung von **MODMWA**, **MODMOU** und **NERR** ist selbsterklärend. **MODMCR**, **MODMLF** und **MODMEO** wirken bei Ein- und Ausgabe. **IOCEF** wird nicht gesetzt, **EXCLU** nicht berücksichtigt. Als Auftragsnummer ist lediglich 7 sinnvoll.

- **MODMSC**

  Ist dieses Bit nicht gesetzt, kann mit der Eingabe von **^A** oder **^B** der Zugang zum Bedieninterpreter angefordert werden (Aktivierung der Task **#USER*x***). Ist gerade ein Eingabe-CE in Bearbeitung, so werden diese Zeichen nicht in das CE übernommen, d.h. die Eingabe wird regulär bis zum Ende abgewickelt. Das vom **USER** generierte Eingabe-CE übernimmt seine Parametrierung aus der Art der Anforderung: wurde **^A** empfangen, so wird die Befehlseingabe im A-Betrieb angefordert, mit **^B** jedoch im B-Betrieb. Hierdurch ist es möglich, schon vor dem Eintreffen des Eingabe-CE's des Bedieninterpreters empfangene Zeichen als Teil eines Befehls zu werten.

- **MODMNE**

  Ist **MODMNE** nicht gesetzt, so werden eingegebene Zeichen bei der Übernahme in ein CE gleichzeitig ausgegeben (s. hierzu auch **MODBIN**). Das Echo erfolgt also z.B. im B-Betrieb nicht bei der Übernahme in den Empfangspuffer (d.h. im Ruhezustand), sondern erst bei der Übertragung in ein CE. Im B-Betrieb werden die empfangenen Zeichen ohne Umwandlung ausgegeben, im A-Betrieb werden Steuerzeichen (Zeichen kleiner als 0x20, dem Leerzeichen) als **@** ausgegeben.

- **MODBIN**

Ist **MODBIN** nicht gesetzt, so ist das X_{ON}-X_{OFF}-Protokoll eingeschaltet. Ferner wird bei empfangenen Zeichen das oberste Bit gelöscht. Ist **MODBIN** eingeschaltet, so ist das X_{ON}-X_{OFF}-Protokoll ausgeschaltet; Betriebssystemimplementierungen, die das RTS/CTS-Protokoll unterstützen, setzen dieses ein. Ferner werden alle empfangenen Zeichen in den Empfangspuffer bzw. das CE, übernommen. Die Editierfunktion (s. u.) der Schnittstellen ist ausgeschaltet, ein Echo eingehender Zeichen findet nicht statt.

5.5.2.5 Editierfunktion

Bei der Eingabe ist die Änderung des Eingabetextes mit folgenden Sonderzeichen möglich:

- Cursor links, Backspace (**^H**) und Delete im CE, falls möglich, um ein Zeichen zurück, d.h. löschen das gerade empfangene Zeichen. Als Echo wird **^H**, ein Leerzeichen und wieder **^H** ausgegeben.
- Cursor rechts (**^L**) läßt das an der aktuellen Position im CE befindliche Zeichen unverändert. Dieses Zeichen wird als Echo auf **^L** ausgegeben.

Diese Sonderzeichen werden nicht ins CE übernommen. Die Editierfunktion ist im Binärbetrieb (**MODBIN**) sowie bei gesetztem **MODMSC** ausgeschaltet.

5.5.3 Die Ramdisk - /ED/, /EDB/

In fast jedem RTOS-UH-System existiert die Ramdisk als dateiorientiertes Speichermedium. Die Dateien der Ramdisk werden durch eine Geräte- und Pfadangabe angesprochen, z.B. spricht **/ED/test** die Datei **test** an. Bei der Pfadangabe können auch Verzeichnisse und Unterverzeichnisse benutzt werden, z.B. spricht **/ED/Heinz/test** die Datei **test** im Verzeichnis **Heinz** an. Verzeichnisse und Unterverzeichnisse brauchen nicht expliziert angelegt werden. Die Nutzung der Ramdisk gestaltet sich mit diesen Kenntnissen relativ einfach:

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

startet einen Kopiervorgang von der Systemschnittstelle im A-Betrieb in die Datei **test**. Die **COPY/*xx***-Subtask untersucht nun als erstes die Geräteparameter der betroffenen Geräte und stellt fest, daß eine Datei der Ramdisk vor der Benutzung geöffnet werden muß (wie im CE1 des obigen Beipiels zu sehen). Weiterhin stellt sie fest, daß es sich bei **/A1/** um ein dialogfähiges Datenterminal handelt, daß mit Echo bearbeitet werden möchte, und gibt daher (zur Erleichterung der Bedienung) ein **=** zur Kennzeichnung der Eingabebereitschaft aus. Anschliessend fordert sie eine Eingabe von **/A1/** an wie im CE2 des Beispiels zu sehen. Im IOBUF von CE2 können auch noch die Reste der Ausgaben des Gleichheitszeichen beobachtet werden. Mit den Eingaben

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

werden die Eingabezeilen in die Datei **test** kopiert. Jedes einzelne eingegebene Zeichen wird auf den Bildschirm ge-"echo"-t; die einzelnen Zeilen werden bei der Eingabe mit **CR** abgeschloßen. Das als **@** ge-"echo"-te Zeichen ist als **^D** (ASCII **EOT**, End-of-Text) eingegeben worden und kennzeichnet das Dateiende. Die Subtask erkennt das (eingegebene) Dateiende, schließt die Datei **test** und beendet ihre Aktivität mit der Meldung

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

Der eingegebene Text ist jetzt in der Datei enthalten und kann z.B. mit

```
TYPE /ED/test
```

auf den Bildschirm ausgegeben werden. Mit **L** oder **LU** kann auch festgestellt werden, daß die **COPY**- und die **TYPE**-Subtask nach Beendigung ihrer Aufgabe nicht mehr im System vorhanden sind. Mit **S** kann festgestellt werden, daß nun ein Speichersegment des Typs **EDTF** vorhanden ist und die Datei **test** enthält. Einen Überblick über die vorhandenen Dateien erhält man mit dem Befehl

```
DIR /ED/
```

Sollte als Abschlußmeldung eines **COPY**- oder **TYPE**-Befehls einmal

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

erscheinen, so ist dies kein Zeichen für Schäden am Betriebssystem, sondern deutet darauf hin, daß das Speichersegment, das die Subtask enthielt, schon vor Ausgabe der Endemeldung anderweitig vergeben und benutzt wurde. Der Name der Subtask kann in diesem Fall nicht mehr identifiziert werden.

Das Löschen der Datei ist mit den Befehlen

```
RM /ED/test
```

(engl. ReMove, dt. Entfernen) oder

```
ERASE /ED/test
```

(dt. auslöschen) möglich.

Dateien in der Ramdisk werden in komprimierter, zeilenorientierter Form abgelegt. Auf Grund des Kompressionsalgorithmus können in Dateien im **/ED/** nur 7-Bit-ASCII-Daten enthalten. Sollen auch andere Daten in Ramdisk-Dateien gespeichert werden, so steht hierzu das Gerät **/EDB/** zur Verfügung. Dieses arbeitet ohne Datenkompression; die Behandlung ist ansonsten identisch zu **/ED/**.

5.5.4 /VI/, /VO/

Die virtuelle Datenstation stellt unter unterschiedlichen Dateinamen unterschiedliche Datenkanäle zur Verfügung. Auch die Angabe von Verzeichnissen und Unterverzeichnissen ist möglich, diese brauchen aber nicht explizit eingerichtet werden.

/VI/ erlaubt nur Eingaben, d.h. auf **/VI/** kann nur lesend zugegriffen werden. Mit

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

wird ein Kopiervorgang gestartet, der aus dem Eingabekanal mit dem Dateinamen **kanal1** ausliest und die gelesenen Zeichen in die Ramdisk, Datei **test2**, kopiert.

/VO/ wiederum ist nur als Ausgabegerät zu benutzen. Ein virtueller Datenkanal wird hierbei durch den gleichen Dateinamen bei **/VI/** und **/VO/** gekennzeichnet. Mit

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

wird also ein Kopiervorgang von der Systemschnittstelle (erkennbar an der Ausgabe des Gleichheitszeichens als Eingabeprompt) in den Datenkanal **kanal1** gestartet. Da die Daten aus diesem Datenkanal mit dem vorigen **COPY**-Befehl in die Datei **test2** der Ramdisk transferiert werden, ist hier nur eine umständlichere Möglichkeit des Befehls **COPY /A1/>/ED/test2** dargestellt.

Mit den Eingaben

```
=MODULE TEST2;
=MODEND;
=@
```

kann nun Text in die Datei **test2** eingegeben werden. Bei Eingabeende durch **^D** (wird auf dem Bildschirm als **@** dargestellt) werden beide **COPY/*xx***-Subtasks terminiert, erkennbar an den zwei **TERMI**-Meldungen.

Einzelne Ein- oder Ausgabekanäle können mit **RM /VI/Kanalname** oder **RM /VO/Kanalname** geleert werden. Die enthaltenen CE's werden in diesem Fall an die veranlassenden Tasks zurückgegeben. In den CE's wird vorher notiert, daß der Ein- bzw. Ausgabevorgang nicht erfolgreich durchgeführt werden konnte.

6 Verzeichnis der Abbildungen

7 Verzeichnis der Tabellen

8 Index

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