Dipl.-Ing.
Meinhard Wrobel

SIEMENS
Bereich Bauelemente
Mikrocomputer-Entwicklung
Systemberatung-Baugruppen
B WIS P MCS TE ST B
Otto-Hahn-Str. 23
8012 Ottobrunn-Riemerling

Echtzeit - Multitasking - Multiprozessor - Betriebssystem RMOS2

Zusammenfassung

Es handelt sich in der heutigen Zeit um eine allgemein anerkannte Tatsache, daß die Kosten für die Entwicklung, den Weiterausbau sowie den Unterhalt von mikroprozessorgesteuerten Produkten vorwiegend durch die Softwarekosten bestimmt sind. Damit ergibt sich die Notwendigkeit, dem Anwender auch für seine Softwareentwicklung und -wartung geeignete standardisierte "Software-Bausteine" zur Verfügung zu stellen.

Ein solcher Software-Baustein ist das Realtime-Multitasking-Operating-System für 16-Bit-Prozessoren "RMOS2", das in vielen Anwendungsfällen die Entwicklung, Wartung und Pflege von Mikroprozessorsoftware wesentlich beschleunigt und vereinfacht.

Mit dem Betriebssytem RMOS2 kann ein Anwendungssystem aber auch als Entwicklungssystem eingesetzt werden. Neben den Echtzeit-Eigenschaften des Betriebssystems wird damit eine komfortable Entwicklungsoberfläche mit höheren Programmiersprachen und einem bildschirmorientierten Editor zur Verfügung gestellt.

Dieser Vortrag gibt eine kurze Einführung in die Struktur und die Wirkungsweise dieses Realtime-Multitasking-Betriebssystems und beschreibt die wesentlichen Merkmale von RMOS2 mit dem Ziel, so die Vorteile für den Anwender bei der Entwicklung seiner Software unter Einsatz von RMOS2 zu verdeutlichen.

1. Schichtenmodell des Betriebssystems RMOS2

Die Struktur des Betriebssystems RMOS2 ist in Bild 1 in Form eines Schichtenmodells dargestellt.

Die unterste Schicht wird durch die Hardware repräsentiert. RMOS2 ist ein modulares Realtime-Betriebssystem für die Prozessoren 8086/88, 80186/80188 und 80286 (real mode) und unterstützt dabei die Verwendung der Arithmetik-Prozessoren 8087/80287.
Wird als Prozessor ein 8086/88 oder 80286 verwendet, so müssen zur einwandfreien Funktion von RMOS2 noch mindestens
- ein Zeitgeber-Baustein: SAB8253, SAB8254 oder Am9513 und
- ein Interrupt-Steuerbaustein: SAB8259 oder Am9519A enthalten sein.
Bei der Verwendung einer CPU 80186/80188 können die integrierten Zeitgeber- und Interrupt-Steuerbausteine benutzt werden.

Eine erweiterte Version des Betriebssystems (RMOS286) erlaubt den Betrieb des 80286 im sog. protected mode. Damit wird ein physikalischer Adreßraum von 16 MByte zur Verfügung gestellt. Außerdem unterstützen zusätzliche Betriebssystem-Aufrufe die Verwaltung von Deskriptoren und Zugriffsrechten der Speichersegmente. RMOS286 überprüft bei jedem Speicher-Zugriff die Segment-Länge und die Zugriffsart und bricht im Fehlerfall den laufenden Prozeß mit einer Meldung ab. Damit sind Fehler, die zum Stack-Überlauf, unerwarteten Interrupt oder Code-Zerstörung führen, ausgeschlossen.

Die erste auf der Hardware aufgesetzte Schicht realisiert die verschiedenen Treiberfunktionen (s. 6.).
Die Software-Treiber stellen - in Zusammenarbeit mit dem Betriebssystem - den Tasks eine einfache, normierte Schnittstelle für den Daten-Ein/Ausgabe-Verkehr mit komplexen Peripheriegeräten zur Verfügung.

Im Kern des RMOS2-Betriebssystems befinden sich die gesamten Prozeduren für die Multitasking-Multiprozessor Funktionen. Er teilt die verschiedenen Tasks über ein Prioritätsschema der CPU zu und ermöglicht die Kommunikation und Synchronisation der verschiedenen Tasks über Events, Semaphore und Mailboxes. Weiterhin führt er die gesamte Interruptverarbeitung, die Speicherverwaltung und die Zeitverwaltung durch.

Als Erweiterung des RMOS2-Nucleus durch die Bereitstellung zusätzlicher Betriebssystem-Aufrufe (Supervisor Call,SVC) können
- die lose Rechnerkopplung mittels globaler Datenbereiche (IPK) und
- das kompakte, leistungsfähige Dateiverwaltungssystem (HSF) für den Betrieb externer Massenspeicher angesehen werden.

Der Schicht oberhalb des RMOS2-Nucleus ist das Sprachinterface zugeordnet, das die Betriebssystem-Aufrufe von PL/M-86-, C-, FORTRAN- oder PASCAL-Programmen in die Assemblerschnittstelle des Betriebssystemkerns umwandelt.

Darauf aufbauend werden verschiedene RMOS2-Systemerweiterungstasks zur Verfügung gestellt:
- ERROR LOGGER zur Bearbeitung von Fehlermeldungen während der Laufzeit des Betriebssystems,
- DEBUGGER als effektive Hilfe beim Austesten von Tasks und
- RESOURCE-REPORTER zur Bestandsaufnahme der Zustände von betriebssystem-internen Betriebsmitteln bzw. Datenstrukturen.
- Mit Hilfe der optionalen SDLC-Folgesteuerung kann ein RMOS2-System als Secondary Station für bitorientierte Steuerungsverfahren (SDLC/HDLC) in einer IBM-SNA-Umgebung realisiert werden.

Dieser Ebene kann man ebenfalls die Schnittstelle UDI (Universal Development Interface = SRI86R) zuordnen, die den Ablauf der gesamten höheren Programmiersprachen und Software-Tools erlaubt. Diese Schicht ist voll multitask-fähig, d.h. es können mehrere SW-Tools (z.B. Compiler) gleichzeitig auf einem oder mehreren Prozessoren laufen. Die Echtzeit-Fähigkeit des Betriebssystems ist dabei immer gewährleistet.

2. Multitask-Verarbeitung bei RMOS2

"Multitasking" bedeutet bei einer im System enthaltenen Zentraleinheit die "quasi-simultane" Bearbeitung zweier oder mehrerer Programme (Tasks). Die Motivation für diese Form der CPU-Benutzung entsteht aus dem Wunsch nach optimaler Ausnutzung des Zentralrechners. Die quasi-parallele Abarbeitung mehrerer Tasks erfordert einen speziellen Task-Wechsel-Mechanismus, der sich bei RMOS2 an einer absoluten Vorrangordnung (Prioritäten-Reihenfolge von ϕ niedrigste Priorität .... 255 höchste Priorität) orientiert.

Diese Prioritäten werden zum Zeitpunkt der System-Generierung durch den Anwender statisch definiert und können während der Laufzeit dynamisch verändert (s. 5.1, 9.) werden.
Das Betriebssystem (SCHEDULER) teilt jeweils der Task, die bereit zum Rechnen ist und die höchste Priorität aller bereiten Tasks besitzt, die CPU zu.
Eine rechnende Task wird im Prinzip solange bearbeitet, bis sie selbständig die Kontrolle an das Betriebssystem zurückgibt, bis sie durch eine gerade bereit gewordene Task höherer Priorität oder durch Gerätetreiber-Programme in ihrer Ausführung unterbrochen wird.

2.1. Task-Betriebs-Zustände und Zustandswechsel bei RMOS2

Ein guter Steuerungsmechanismus des Betriebssystems zeichnet sich dadurch aus, daß er einerseits sehr schnell ist (geringer Verwaltungsaufwand) und andererseits vom Benutzer leicht zu steuern ist. Dies ist mit der in RMOS2 realisierten Task-Organisation gegeben. Bild 2 zeigt die elementaren Task-Zustände und deren mögliche Zustands-Wechsel.

- NICHT EXISTENT:
  Existenz der Task ist innerhalb des Betriebssystems nicht bekannt. Der Code der Task kann entweder bereits im Speicher stehen oder von einem externen Massenspeicher-Medium nachgeladen werden.
- SUSPENDIERT:
  Die Task ist im Ruhezustand. Ihre Existenz ist innerhalb des Betriebssystems bekannt; sie kann durch eine andere Task gestartet werden.
- BEREIT:
  Die Task wartet auf die Zuteilung der CPU-Steuerung, d. h. mit Ausnahme der CPU sind alle für die Fortsetzung der Task-Abarbeitung benötigten Betriebsmittel vorhanden.
- RECHNEND:
  Die Task hat die Zentraleinheit zugeteilt bekommen und führt die Abarbeitung der Programmbefehle durch.
- WARTEND:
  (eigentlich mehrere unterschiedliche Wartend-Bedingungen) Die Task ist blockiert, da entweder ein angefordertes Systembetriebsmittel nicht frei ist, oder die Task auf ein Ereignis (z.B. das Ende einer E/A-Operation) wartet, bevor sie wieder "BEREIT" werden kann.

Mit den Zustandswechseln KREIEREN und LÖSCHEN kann der Anwender während der Laufzeit des Systems dynamisch Tasks definieren bzw. löschen und so dafür sorgen, daß RMOS2 immer nur die momentan verwendeten Tasks verwaltet. Durch das Nachladen von Tasks ergibt sich für den Anwender die Möglichkeit einer effizienten Arbeitsspeicher-Auslastung.

Der Zustandswechsel BEREITSTELLEN wird ausgeführt bei:
- dem automatischen Start der sog. Initialisierungstask während der RMOS2-Systeminitialisierung,
- dem System-Aufruf "Task starten" (STRT, QSTRT),
- einer unangeforderten Eingabe an einer Geräteeinheit oder
- der Startanforderung aufgrund eines Interrupts.

Die Zustandswechsel ZUORDNEN bzw. VERDRÄNGEN folgen dem in 2. dargestellten Algorithmus. Der Wechsel ZUORDNEN wird immer dann vom Betriebssystem veranlaßt, wenn sämtliche System- Aktivitäten beendet sind und in den sog. "Applikations-Zustand" umgeschaltet werden soll (s. 3.).
Der Zustandswechsel VERDRÄNGEN erfolgt z.B.:
- nach einem Betriebssystem-Aufruf der rechnenden Task,
  nach einem Systemtakt-Interrupt, wenn eine höherpriore Task BEREIT wird,
  nachdem eine Interrupt-Routine zur Bedienung eines Ein/Ausgabe-Gerätes in den sog. "System-Zustand" (s. 3.) geschaltet hat.

Typische Gründe für den Zustandswechsel BLOCKIEREN sind Betriebssystemaufrufe der rechnenden Tasks mit einer sog. WARTE-Bedingung, z.B.:
- Warten auf die Beendigung einer Ein/Ausgabe-Funktion,
- Warten auf den Ablauf einer Pause,
- Warten auf das Eintreffen einer Nachricht usw.

Analog dazu wird bei der Beendigung der zuvor genannten Warte-Bedingungen vom Betriebssystem der Zustandswechsel DEBLOCKIEREN ausgeführt.

RMOS2 gestattet einer rechnenden Task mit zwei System-Aufrufen den Zustandswechsel BEENDEN:
- Task beenden (ENDT),
  Task beenden und nach Ablauf eines Zeitintervalls erneut starten (ENDR).

Die Tatsache, daß sich eine Task in einem der beschriebenen Zustände befindet, äußert sich in der Einordnung des zugehörigen Task-Kontrollblocks in eine entsprechende Warteschlange, die jedem Prozeßzustand zugeordnet ist. Bei RMOS2 sind diese Warteschlangen in Form von "linearen, einfach verketteten Listen" organisiert und nach fallenden Prioritäten geordnet. Diese Organisation wird unter anderem auch beim Einketten von prioritätsbehafteten "Nachrichten" an internen Kommunikationspunkten (Mailboxes) benutzt.

Dabei gilt als genereller Algorithmus, daß ein neu einzutragender Beschreibungsblock (z.B. Task-Kontrollblock) stets hinter den Beschreibungsblock mit der gleichen Priorität (sofern vorhanden) eingereiht wird.

2.2. Besondere Merkmale der RMOS2-Task-Verwaltung

Ohne eine spezielle Unterstützung durch das Betriebssystem sind Umorganisationen in den beschriebenen Zustands-Warteschlangen (z.B. Aufheben von Dead-Locks durch zwangsweise Aufhebung des WARTEND-Zustands) generell nicht möglich. Aus diesem Grund sind in RMOS2 die nachfolgend beschriebenen Maßnahmen realisiert worden.

2.2.1 Override Zeitverwaltung

Dieser Mechanismus wird verwendet, um zeitlich geplant einen oder alle der folgenden Zustände aufzuheben:

- momentane Task-Priorität (BEREIT-Zustand),
- Anforderung einer Task, ein Semaphor zu testen und zu setzen oder ein Programm mit überwachtem Zugang abzuarbeiten (WARTEND-Zustand),
- Anforderung einer Task zur Reservierung einer Geräteeinheit (WARTEND-Zustand).

Das gewünschte Override-Zeitintervall wird bei der Systemkonfigurierung festgelegt und beginnt zu folgenden Zeitpunkten:

- Override Priorität: Beim Starten einer Task.
  Nach Ablauf des Zeitintervalls wird die Task-Priorität solange um einen Schrittwert erhöht, bis ein Grenzwert der Priorität erreicht ist (Schritt- und Grenzwert konfigurierbar).

- Override Semaphor-Zuteilung, Eintritt in Programm mit überwachtem Zugang oder Reservierung einer Geräteeinheit: Beim Absetzen des jeweiligen Betriebssystem-Aufrufes.
  Nach Ablauf des Zeitintervalls wird die Task mit einer entsprechenden Status-Meldung wieder in den BEREIT-Zustand zurückversetzt.

2.2.2 ROUND-ROBIN-Funktion

Eine CPU-Zuteilung im sog. TIME-SHARING-Verfahren für gleichpriore Tasks in der BEREIT-Warteschlange ist mit Hilfe des ROUND-ROBIN-Zählers möglich. Dieser zwingt die Behandlung des Zeitgeber-Interrupts nach einer konfigurierbaren Anzahl von "TICKS" durch die RMOS2-Instanz SCHEDULER, wodurch ein Rescheduling-Vorgang ausgelöst wird. Mit der in 2.1 beschriebenen Warteschlangen-Organisation erfolgt somit eine Abarbeitung von gleichprioren Tasks im Zeitscheiben-Verfahren (Bild 3).

2.2.3. Automatische Prioritätsanhebung bei Semaphor- und CNTRL-Aufrufen

Klassischerweise dienen Semaphore zum gegenseitigen Ausschluß bei von mehreren Tasks gemeinsam genutzten Datenbereichen und kritische Programmabschnitte (CNTRL) zum gegenseitigen Ausschluß bei gemeinsam genutzten Codebereichen.
Um zu verhindern, daß eine niedrig-priore Task, die in den Besitz eines dieser Betriebsmittel gekommen ist, eine höher-priore Task in ihrer Ausführung behindert, die zu einem späteren Zeitpunkt in den Besitz des gleichen Betriebsmittels gelangen will, hebt das Betriebssystem in diesem Fall vorübergehend die Priorität der niedrig-prioren Task auf die Priorität der höher-prioren Task an.
Damit wird gewährleistet, daß eine zur selben Zeit eingreifende mittelpriore Task (die keinerlei Zugriff auf das Betriebsmittel wünscht) nicht den Ablauf der beiden anderen Tasks blockiert.

2.2.4 Nachladen entrelativierter Tasks

Wird mit einem Start-Aufruf eine nachladbare, entrelativierte Task (Angabe in System-Konfigurierung) aufgerufen, so prüft eine zentrale Verwaltungsinstanz in RMOS2 (OVERLAY-Manager), ob der im Lade-Deskriptor angegebene Overlay-Speicherbereich (muß adressmäßig mit der entrelativierten Task übereinstimmen) gerade von einer anderen Task belegt wird.

In diesem Fall wartet das Betriebssystem mit dem Start der LOADER-Task auf die Freigabe des Speicherbereichs durch einen ENDE-SVC der momentan residenten Task.
Die Loader-Task lädt dann das absolute Objektdateiformat der Ziel-Task (produziert vom Locater-Programm LOC86) in den Arbeitsspeicher und meldet mit einem speziellen Aufruf dem Betriebssystem das Ladeende. RMOS2 startet nun die Zieltask, während sich die Loader Task beendet und auf einen erneuten Start durch RMOS2 wartet. Bild 4 dient zur Erläuterung der beschriebenen Vorgänge.

3. Realtime-Reaktion bei RMOS2

Von außerordentlicher Bedeutung in den meisten Mikroprozessor-Anwendungen ist die schnelle Reaktion auf äußere Ereignisse, die in einem System mit Echtzeitverhalten auf Interrupts des Prozessors abgebildet werden.
Interrupts können im Laufe eines Prozesses zu beliebigen Zeiten eintreffen und direkt bedient werden, wenn die CPU gerade nicht unter Interrupt-Sperre läuft. Eine diesbezügliche Analyse läßt bei RMOS2 den Zustand "RECHNEND" in vier Betriebszustände zerfallen:

- Direkter Interrupt (DI),
- Interrupt (I),
- System (S),
- Applikation (A).

Im Applikations-Zustand werden die Programme der Anwendertasks von der CPU ausgeführt, alle Interrupts sind freigegeben.

Im System-Zustand führt die CPU RMOS2-Routinen oder Programme zur Bedienung von Ein/Ausgabe-Geräten aus. Es sind alle Interrupts freigegeben. Es wird keine Task ausgeführt, bis alle Aktivitäten im S-Zustand abgeschlossen sind.

Im Interrupt-Zustand führt die CPU im allgemeinen Gerätetreiber-Programme aus. Es sind alle Interrupts blockiert, die die gleiche oder eine niedrigere Hardware-Priorität besitzen als der gerade bearbeitete Interrupt.

Im direkten Interrupt Zustand führt die CPU ebenfalls Gerätetreiber-Programme aus, aber alle Interrupts sind blockiert.

Nach dem Verlassen des DI- bzw. I-Zustandes wird immer das Programm (Task, Betriebssystem- oder unterbrochene Interrupt-Routine) fortgesetzt, das zum Zeitpunkt des Auftretens des Interrupts gerade aktiv war. Die Zustandswechsel (DI -- I -- S) können durch das Aufrufen bestimmter Betriebssystem-Routinen veranlaßt werden.

Bei RMOS2 werden die Betriebssystem-Aufrufe in Software-Interrupts abgebildet, bei denen nur die wirklich kritischen Teile unter Interrupt-Sperre laufen (Sperre für max. 100 - 150 µs bei 8086-CPU mit 5 MHz-Takt ohne Speicher-Wartezyklen). Bei einem Interrupt höchster Priorität entsteht keinerlei Annahmeverzug, wenn sich die CPU nicht im DI-Zustand befindet. Für niedrig priorisierte Interrupts muß hier die längste Laufzeit der höher-priorisierten Interrupts einschl. eventueller Schachtelungen kalkuliert werden.

4. Multiprozessor-Unterstützung bei RMOS2

Moderne Mikrocomputersysteme enthalten mehrere Prozessoren, die Leistungsdurchsatz, Verfügbarkeit und Reaktionszeit gegenüber Einzelprozessorsystemen erhöhen. Das Echtzeit-Betriebssystem RMOS2 unterstützt die sogenannten "lose gekoppelten" (loosely coupled) und "eng gekoppelten" (tightly coupled) Multiprozessor-Konfigurationen.

Die Interprozessor-Kommunikation (IPK) ermöglicht den Datenaustausch zwischen mehreren eigenständigen Monoprozessor-RMOS2-Anwendungen über einen globalen, d.h. von allen Prozessoren erreichbaren Speicherbereich. Die Kommunikation basiert auf Senden und Empfangen von Botschaften zu bzw. von diesen sog. globalen Mailboxes (bis zu 256 Mailboxes konfigurierbar).
Die Task-Synchronisierung in dieser lose gekoppelten Multiprozessor-Umgebung kann entweder im Polling-Verfahren oder per Interrupt realisiert werden.

Für den Ablauf der Interprozessor-Kommunikation wird eine der folgenden Hardware-Konfigurationen benötigt:

- Multiprozessorfähiges System mit Prioritätslogik, BUS-Sperrmechanismus und einem globalen/dual ported Speicherbereich (z.B. AMS-M-System, Bild 5).
- Über dual ported Speicherbereiche gekoppelte Monoprozessorsysteme mit BUS-Sperrmechanismus (z.B. zwei SMP-Systeme mit den dual ported Koppelkarten SMP-E150-P1 und SMP-E150-P2).

RMOS2 unterstützt bereits vom Nucleus her das eng gekoppelte Multiprozessor-System als eine einzelne, aus mehreren Prozessoren bestehende, Anwendung. Die Vorteile dieser Organisation liegen darin, daß die Systemlast automatisch auf die einzelnen Prozessoren, je nach laufenden Bedingungen der Anwendung, verteilt wird.
In der in Bild 5 gezeigten multiprozessorfähigen Hardware-Umgebung kann RMOS2 den eigenen und den Task-Code sowohl in lokalen als auch in globalen Speicherbereichen ausführen.
Die Entscheidung, ob eine Task nur auf einer bestimmten oder auf einer beliebigen CPU im System laufen kann, wird bei der System-Konfiguration getroffen.
RMOS2 unterhält für jeden Prozessor eine Warteschlange, die aus allen lokalen Tasks besteht, die sich im Zustand "BEREIT" befinden und für alle Prozessoren eine Warteschlange, die aus allen globalen Tasks besteht, die sich im Zustand "BEREIT" befinden. Der grundlegende Scheduler-Algorithmus zur Abarbeitung dieser Warteschlangen ist im Bild 6 dargestellt.
Das Anfordern von Ein-/Ausgabe-Operationen ist nicht nur auf die Tasks beschränkt, die auf dem Prozessor ablaufen, an dem die Ein-/Ausgabegeräte angeschlossen sind.
Die Abarbeitung solcher Anforderungen wird intern von RMOS2 gesteuert und muß nicht explizit programmiert werden. Der implementierte Mechanismus arbeitet wie folgt: Die Interrupt-gesteuerte Ein-/Ausgabe wird von Treibern bedient, die von jeder Task angesprochen werden können. Bei einem E/A-System-Aufruf prüft RMOS2, ob die Ein-/Ausgabe-Operation von demselben Prozessor, auf dem auch sie abläuft, ausgeführt werden kann. Falls ja, wird die Ein-/Ausgabe-Operation wie üblich ausgeführt. Anderenfalls wird die Anforderung in die Warteschlange des Prozessors eingereiht, der für die Steuerung des Ein-/Ausgabe-Gerätes verantwortlich ist.

Mit anderen Worten ist das RMOS2-Multiprozessor-Modell sowohl in Bezug auf die Ein-/Ausgabe als auch in Bezug auf die restlichen Aspekte für die Tasks vollständig "transparent". Wenn die Anzahl der Prozessoren bei der Konfigurierung genügend hoch gewählt wurde, verlangt in einer späteren Ausführungsphase das Anschließen von weiteren Prozessoren, um den Systemdurchsatz zu erhöhen, keinesfalls Programmänderungen.
Mit den von RMOS2 angebotenen Multiprozessor-Funktionen kann der Anwender sowohl seine "Arbeitslast" (Tasks) als auch seine "Interruptlast" (Peripheriegeräte) so optimal wie möglich auf die im System enthaltenen Zentralprozessoren verteilen.

5. Weitere Merkmale des Betriebssystemkerns von RMOS2

Der Kern des Betriebssystems bietet neben der in 2. und 3. dargestellten MULTITASK- und REALTIME-Unterstützung komfortable Systemdienste zur Task-Verwaltung, Betriebsmittel-Verwaltung und zur Task-Koordination und -Kommunikation.
In Bild 7 sind die vom Betriebssystem zur Verfügung gestellten, grundlegenden Systemdienste genauer aufgegliedert.

Alle Betriebsmittel können dynamisch und statisch vereinbart werden, d.h. die Betriebsmittel werden während der Laufzeit (dynamisch) oder bei der Konfigurierung (statisch) definiert. Die dynamisch definierten Betriebsmittel sind auch löschbar. Es existiert ein CREATE- und DELETE-Aufruf für die Betriebsmittel Speicherpool, Semaphore, Mailbox, Ereignisflag, Diskrete Bytes, Treiber, Overlay Speicherbereich und Task.

Jedem statisch oder dynamisch vereinbarten Betriebsmittel kann mit dem Betriebssystemaufruf CATALOG ein Zeichenstring mit bis zu 12 Zeichen zugewiesen werden. Der Aufruf CATALOG trägt den Zeichenstring und die zugehörige Betriebsmittelkennung (ID) in ein Inhaltsverzeichnis ein. Alle registrierten Betriebsmittelkennungen können mit dem Aufruf LOOK anhand des Zeichenstrings wieder ermittelt werden. Der Anwender hat mit den Aufrufen CATALOG und LOOK die Möglichkeit, alle Betriebsmittel mit logischen Namen anzusprechen.
Der Betriebssystemaufruf GETSTAT liefert die momentane Priorität, Status, CPU-ID und den Stackpointer einer Task zurück.
Mit dem Aufruf SETINT kann einem Interrupt-Vektor ein neuer Interrupt-Handler zugewiesen werden.
Für das Schreiben von Treibern stehen auch Nucleus-Funktionen zur Verfügung:
- Ereignisflag setzen, testen oder rücksetzen,
- Botschaft an eine Mailbox senden bzw. von einer Mailbox empfangen,
- Speicherbereiche anfordern und zurückgeben,
- andere Task starten.

5.1. Task-Verwaltung

Die Task-Verwaltung besteht im wesentlichen aus den Aufrufen:
- Task kreieren, löschen, starten, beenden und zyklischer Task-Start,
- gezieltes Aufheben der Pause einer anderen Task,
- Änderung der eigenen Priorität oder der Priorität einer anderen Task.

Ein weiterer Dienst ist nicht als Betriebssystem-Aufruf ausgeführt, sondern kann bei der System-Generierung gewählt werden:

- Interrupt Task Start
  Damit kann die Behandlung von Hardware- und Software-Interrupts auf die Task-Ebene verlagert werden, indem bei der Software-Konfiguration die Identitäts-Nummer einer Task spezifiziert wird, die beim Auftreten des entsprechenden Interrupts vom Betriebssystem gestartet werden soll.

5.2. Task-Koordination und -Kommunikation

Die Task-Kommunikation wird bei RMOS2 durch den lokalen Botschaftenverkehr und Ereignisflags ermöglicht.
Der Botschaftenverkehr basiert auf sog. "Mailboxes", zu denen rechnende Tasks Nachrichten senden bzw. von diesen empfangen können. Dabei können sie auswählen, ob sie auf das Abholen ihrer Nachricht (bei SEND) oder das Eintreffen einer Nachricht (bei RECV) warten wollen.
Ereignisflags sind durch logische Bit-Einheiten im Speicher realisiert. Von rechnenden Tasks können ein/mehrere Ereignisflag(s) gesetzt, nach einer bestimmten Zeit gesetzt, rückgesetzt und getestet werden. Auch das Warten auf das Setzen eines/mehrerer Ereignisflags ist möglich.

Die Koordination des "wechselseitigen Ausschlusses" von Tasks bei gleichzeitigem Zugriff auf gemeinsam benutzte Betriebsmittel (z.B. Datenstrukturen, Ports,...) wird mit Hilfe von Semaphor-Aufrufen gesteuert.

Als "kritischen Abschnitt" bezeichnet man ein Programm, das von mehreren Tasks benutzt wird, aber nicht reentrant (wiedereintrittsfähig) geschrieben ist. Anfang und Ende eines solchen Programmabschnitts müssen bei RMOS2 durch den CNTRL-System-Aufruf markiert werden.
Damit gestattet das Betriebssystem zu jedem Zeitpunkt nur einer Task die aktuelle Programmausführung.

5.3. Betriebsmittel-Verwaltung

RMOS2 stellt Systemdienste für die Verwaltung der Betriebsmittel Zeit, Speicherplatz und Peripheriegeräte bereit.
Zur Zeitverwaltung benötigt das Betriebssystem ein Taktsignal von einem Zeitgeber-Baustein ("System-Takt" wählbar in ms-Schritten ab 1 ms).

Neben den zeitbezogenen Systemdiensten (zyklischer Wiederanlauf, Pause, usw.) unterhält RMOS2 intern eine Software-Uhr, die Datum- und Uhrzeit-Funktionen (Auflösung in Sekunden) implementiert und die mittels eigener System-Aufrufe gesetzt und abgefragt werden kann.

In der Speicherplatzverwaltung können von Tasks zusammenhängende Speicherbereiche aus "Speicher-Pools" angefordert bzw. zurückgegeben werden. Mit diesen Systemdiensten wird es einer Task ermöglicht, "dynamisch" den benötigten Speicherbereich anzufordern, wenn sie ihn z.B. zur Messwert-Aufnahme und -Analyse benötigt. Anderenfalls wäre der Programmierer gezwungen, der Task den gesamten Datenbereich statisch zu reservieren, was bei einer weniger häufigen Abarbeitung der Task u. U. zu einer schlechten Speicher-Auslastung führen würde.

RMOS2 bietet in der Ein/Ausgabe-Verwaltung Systemdienste zur Behandlung unterschiedlicher Ein/Ausgabe-Operationen.
Besonders in der Prozeßsteuerung treten häufig eine Vielzahl von externen diskreten Eingabe- (Status von Tasten/Schaltern) bzw. Ausgabe-Signalen (Anzeige, mechanische Stellglieder) auf. Mit Hilfe des DSCRTE-Aufrufes wird das Setzen, Rücksetzen bzw. Testen dieser Signale in einer Multitask-Umgebung gesteuert, bei der u. U. mehrere Tasks auf die gleichen Hardware-Signale zugreifen.

RMOS2 steuert die Ein- und Ausgabe für eine Vielzahl von Peripheriegeräten über die in der zweiten Schicht (Bild 1) enthaltenen "Geräte-Treiber" (s. 6.).

Mittels des RIO-Aufrufes (Request I/O) kann eine Task in komfortabler und normierter Art und Weise die Ein- bzw. Ausgabe von Daten (Texten) über das Betriebssystem an einen Geräte-Treiber weitergeben. Die Task kann beim Aufruf entscheiden, ob sie auf das Ende des Auftrages warten möchte oder ob über ein lokales Ereignisflag eine "Fertig-Meldung" an die Task erfolgen soll.

5.4. Kenndaten des Betriebssystemkerns

- Anzahl der Tasks	bis zu	2048
- Prioritätsebenen		256
- globale Ereignisflag-Gruppen (jeweils 8 Flags)	bis zu	16
- lokale (taskbezogene) Ereignisflags (pro Task)		16
- lokale Mailboxes	bis zu	256
- Anzahl der Gerätetreiber	bis zu	256
- Geräte (pro Gerätetreiber)	bis zu	255
- Programme mit überwachtem Zugang	bis zu	256
- diskrete E/A-Bytes	bis zu	8192
- Speicher-Pools	bis zu	64
- Semaphore	bis zu	4096
- Systemtakt	wählbar	min. 1ms

Die typische Ausführungszeit für einen Betriebssystem-Aufruf (incl. der Reschedule-Zeit) liegt für eine CPU SAB8086 (5 MHz) unter 0,6 ms und für den Prozessor SAB80286 (8MHz) unter 0,2 ms.

Der modulare Aufbau des Betriebssystemkerns gestattet trotz der über 50 Betriebssystemaufrufe eine maßgeschneiderte Konfigurierung für die jeweilige Applikation. Der Codebedarf liegt zwischen 5,5 Kbyte und 17 Kbyte.

6. Kenndaten der RMOS2-Standard-Geräte-Treiber

Für die verschiedenen Hardware-Konfigurierungen existieren passende Treiber, wobei jene für den Betrieb von seriellen/parallelen Kommunikationsbausteinen auf die Typen 2661, 8251, 8274, 8530 und 8255 und jene zur Ansteuerung externer Massenspeichermedien speziell auf die Baugruppensysteme AMS-M (Advanced Microcomputer Systems) und SMP (Siemens Mikrocomputer) abgestimmt sind.

- Treiber für byteorientierte Peripheriegeräte (BYT)
  Dieser unterstützt die zentrale Steuerung aller in einer Anwendung vorkommenden byteorientierten Geräte (Datensichtstation, Drucker, usw.). Als besondere Merkmale gestattet der Treiber die Konfiguration eines History-Puffers, um alle früher getätigten Eingaben zu modifizieren oder unverändert nochmals einzugeben (Human Interface) und erlaubt die Notschreiben- und Cancel-Funktion aller Aufträge des Treibers.

- Datensichtgeräte-Treiber (CRT)
  Dieser Treiber bietet eine Funktions-Untermenge des Byte-Treibers und wird in Source-Form geliefert, um dem Anwender eine Anleitung zum Schreiben eigener Treiber zu geben.

- Magnetblasenspeicher-Treiber (BUB)
  Treiber zur physikalischen Steuerung der Magnetblasenspeicher-Baugruppen SMP-E143, -E141.

- Diskettenlaufwerk-Treiber (FD2)
  Treiber zur Steuerung des physikalischen Disketten-Zugriffs mit den Baugruppen SMP-E342 und AMS-M342 zur Datenaufzeichnung auf 3 1/2"-, 5 1/4"- und 8"- Disketten mit einfacher und doppelter Schreibdichte.

- Festplattenspeicherlaufwerk-Treiber (HD6)
  Treiber zum Betrieb von Winchester- und Tape-Laufwerken (mit SCSI-Schnittstelle) an den Baugruppen SMP-E346 und AMS-M342.

- DMA-Treiber (DMA)
  Treiber zur DMA-Unterstützung der RMOS2-Massenspeicher-Treiber mit den Bausteinen AM9517A, 8089, SAB82258 (ADMA) und der DMA-Steuerung des 80186/188.

- Rechner-Rechner-Kommunikationstreiber (COM)
  Universelles Treiberprogramm für die Rechner-Rechner-Kommunikation über eine serielle Schnittstelle. Der Treiber ist vollduplex-fähig mit XON / XOFF - Protokoll und benutzt einen internen 50 Zeichen langen Empfangspuffer.

7. Kenndaten der RMOS2-Systemerweiterungen

- Debugger
  Der Debugger ist eine effektive Hilfe beim Austesten von Tasks unter RMOS2. Zu dem Leistungsumfang des Debuggers zählen unter anderem:
  - Steuerung des Ablaufs und Prüfen des Zustands aller im Echtzeit-Betriebssystem arbeitenden Tasks.
  - Prüfen des gesamten Speicherinhalts und bei Bedarf Ändern des PAM-Inhalts.
  - Setzen von bis zu 15 Unterbrechungspunkten in beliebigen Tasks.
  - Inhalte der Register einer unterbrochenen Task prüfen/ändern.

- Resource-Reporter
  Der Resource-Reporter stellt eine zusätzliche Systemunterstützung in Form einer Task dar, die als wertvolle Ergänzung zum Debugger in der Testphase gesehen werden kann. Mit Hilfe des Resource-Reporters können Bestandsaufnahmen des Zustands der RMOS2-Betriebsmittel bzw. -Datenstrukturen auf dem Bildschirm angezeigt werden. Er umfaßt die Auswertungen für Tasks, Gerätetreiber, Speicherpools, Semaphore, globale Ereignisflags, Programme mit überwachtem Zugang, Mailboxes und Overlay Speicherbereiche.

- Interprozessor-Kommunikation (s. 4.)

- SDLC-Folgesteuerung
  Programm für die Unterstützung einer Folgesteuerung (Secondary Station) für bitorientierte Steuerungsverfahren (SDLC/HDLC) in einer IBM-SNA-Umgebung mit Steuereinheit 3705-II oder jeder SDLC/HDLC-Umgebung, in der eine Leitsteuerung (Primary Station) existiert.

- Datei-Verwaltungssystem
  Das hierarchische, multiprozessorfähige Datei-Verwaltungssystem erleichtert dem Anwender die Erstellung und die Verarbeitung von Dateien, unterstützt echte Parallel-Arbeit von Massenspeichersteuerungen und besitzt neben einer großen Daten-Verwaltungs-Fähigkeit (Unit-Größe max. 256 Mbyte; Datei-Größe max. 32 Mbyte) eine auf hohen Datendurchsatz optimierte (RAM-)Puffer-Verwaltung.
  Es unterstützt neben dem RMOS-Format auch das ISIS-II und MS-DOS Format.

  Zum Lieferumfang des Filesystems gehört ein RAM-Disk-Treiber (erweiterter Adreßraum von 16 Mbyte mit Hilfe des ADMA), der wie ein externes Speichermedium angesprochen wird, aber extrem kurze Zugriffszeiten bietet.

8. Die Entwicklungsoberfläche bei RMOS2

Neben den optimal auf die Hardware abstimmbaren Realtime-Eigenschaften von RMOS2 wird auch eine komfortable Entwicklungsoberfläche mit Hochsprachen, Editor und Debugger angeboten.

Die Anwender-Schnittstelle (Human Interface) wird dabei durch die Kommando-Interpreter-Task (CLI) realisiert und unterstützt folgende Dienste:
- Start von Entwicklungs- bzw. Anwender-Software,
- Externspeicherbedienung (z.B. Formatieren von Disketten),
- Dateisteuerung (z.B. COPY, DELETE, RENAME),
- Systemsteuerung (z.B. TIME, LIST, CPRI),
- Hilfsfunktionen (HELP, VERSION).

Die Kommando-Interpreter-Task wird normalerweise mit niedriger Priorität versehen, so daß die Anwendertasks alle Resourcen nutzen können und die Bearbeitung aller externen Ereignisse (Interrupts) d. h. die Echtzeit-Fähigkeit von RMOS2 voll zur Verfügung steht.

Bild 8 zeigt eine Applikation mit zwei statischen Anwendertasks und einem statisch konfigurierten Kommando-Interpreter. Der CLI kann aber auch zur Laufzeit beliebig oft mit dem START-Kommando als dynamische Task gestartet werden und dient dann als "Träger" für eine Entwicklungs-(CLI1) bzw. eine Applikationsaufgabe (CLI2) im sog. "Hintergrund"-Betrieb.
Eine Entwicklungsaufgabe wäre z.B. das Editieren, Kompilieren, Binden oder Testen einer neuen Anwendertask, die dann - ohne das System abzuschalten oder neu zu generieren - zusätzlich zu den bestehenden Anwendertasks entweder direkt oder mit Hilfe eines neu gestarteten CLI in das System integriert werden kann.

9. Die System-Konfigurierung bei RMOS2

Die System-Konfigurierung von RMOS2 auf eine bestimmte Applikation besteht im wesentlichen aus der Hardware- und Software-Konfiguration des Betriebssystems.

In der Hardware-Konfiguration bestimmt der Anwender, welche Hardwarebausteine (CPU, Timer, Interrupt-Controller) benutzt werden und mit welchem Code (ASM86) diese zu versorgen sind. Bei evtl. notwendigen Hardware-Konfigurationen für Gerätetreiber werden bausteinspezifische Codesequenzen (DMA) oder Bildschirm-Steuerzeichen (BYT) definiert.

Mit Hilfe der Software-Konfiguration werden die im ROM bzw. RAM abgelegten Datenstrukturen (ASM86), die das Betriebssystem zur internen Verwaltung benötigt, generiert. Dazu gehören neben den allgemeinen Parametern (Ereignisflags, Semaphore, lokale Mailboxes, usw.) z.B. die Tabellen, die die eingebundenen Tasks und die Gerätetreiber mit ihren Geräteeinheiten beschreiben. Weiterhin wird die Programmierung der Interrupt-Vektoren, das Vorhandensein von Debugger, Resource-Reporter und der Umfang des Betriebssystemkerns (Anzahl der benutzten SVC's) konfiguriert.

Die so erhaltenen Assembler-Module werden nach ihrer Übersetzung mit den Bibliotheken des Betriebssystems und den übersetzten Anwendertasks zusammen gebunden. Nach dem Entrelativier-Vorgang (LOC86) kann die so generierte RMOS2-Applikation mit Hilfe eines Entwicklungssystems oder EPROM-Burners in die Ziel-Hardware gebracht werden (Bild 9).

Bild 10 zeigt das Prinzip des BOOT-Vorgangs bei RMOS2.

Beim Einschalten des Mikroprozessorsystems (bzw. nach RESET) wird zuerst eine Initialisierungsroutine durchlaufen, die den in der Hardware-Konfiguration generierten Code ausführt, die Interrupt-Vektoren entsprechend der Software-Konfiguration besetzt und alle Bausteine (Units) der angeschlossenen Treiber initialisiert.

Anschließend erfolgt die Initialisierung der in der Software-Konfiguration bereitgestellten RMOS2-internen Datenstrukturen.

Danach wird die bei der Software-Konfiguration spezifizierte Initialisierungstask gestartet, die gesamte Software ist damit betriebsbereit. Die Interrupt-Behandlungsroutinen warten auf die zugeordneten, äußeren Ereignisse; die verschiedenen Tasks warten auf ihre Ausführung.

Die Ablaufsteuerung von RMOS2 verwaltet die Zustandswechsel aller im System enthaltenen Prozesse.

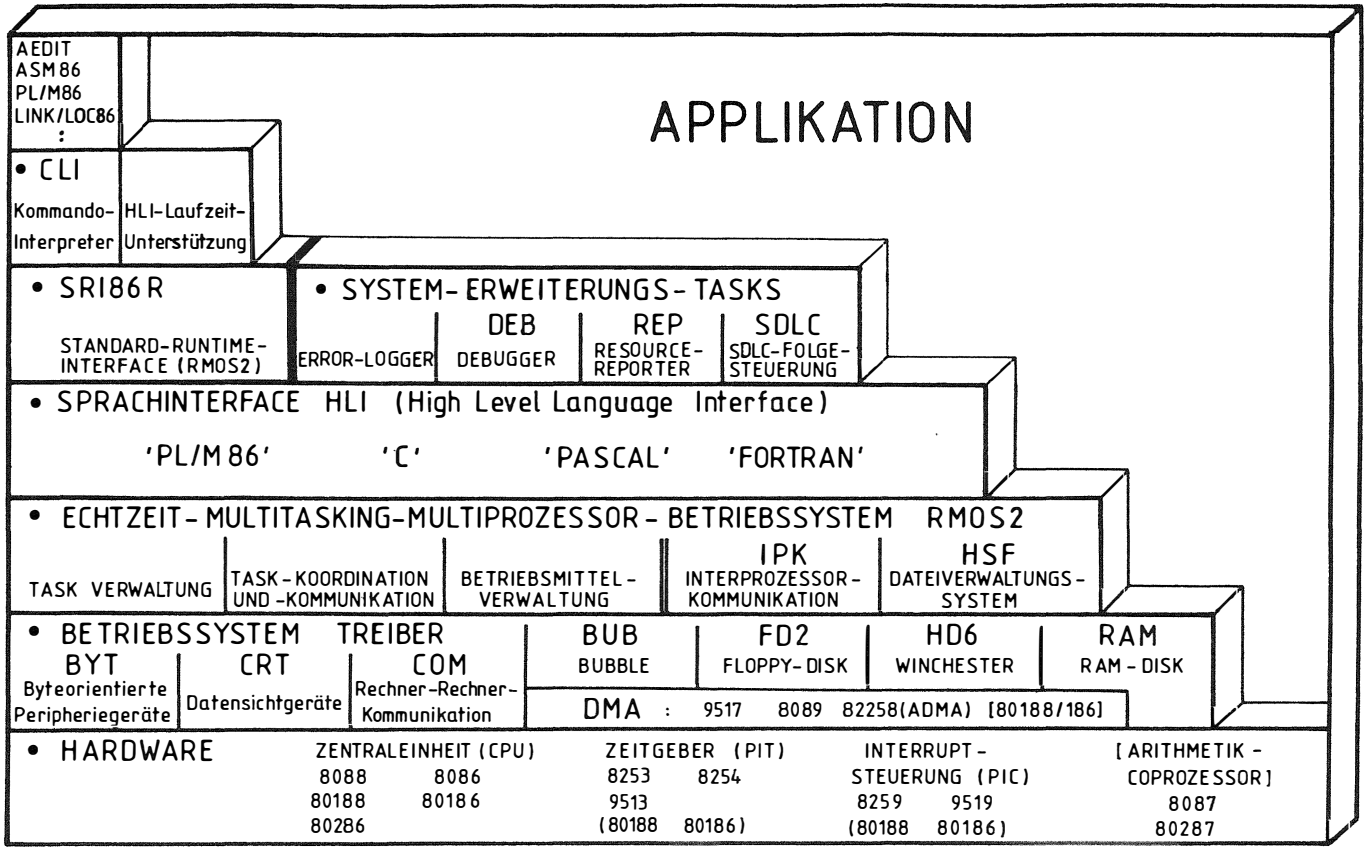


Bild 1. SCHICHTENMODELL DES BETRIEBSSYSTEMS RMOS2

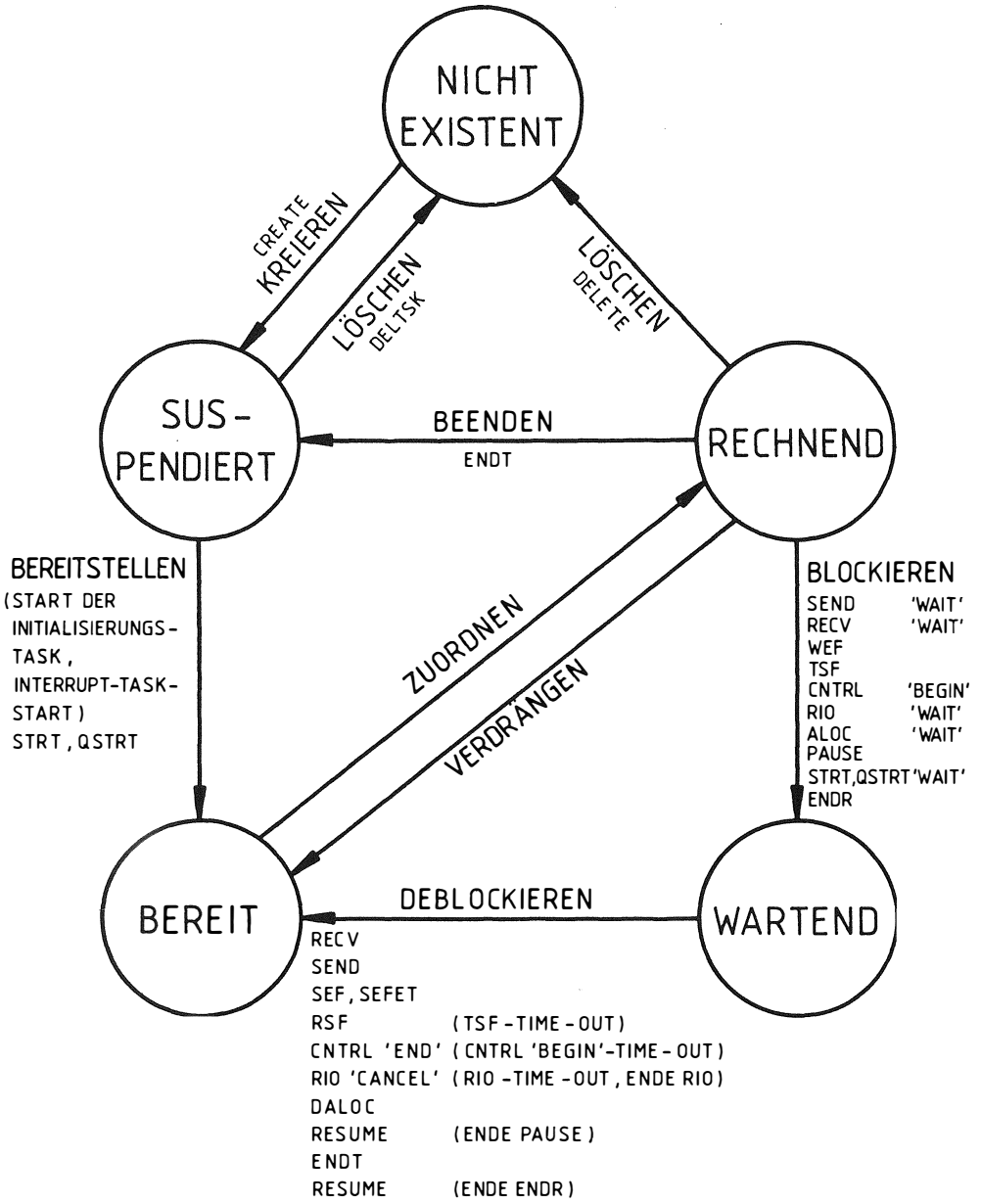


Bild 2. TASK - ZUSTANDS - GRAPH

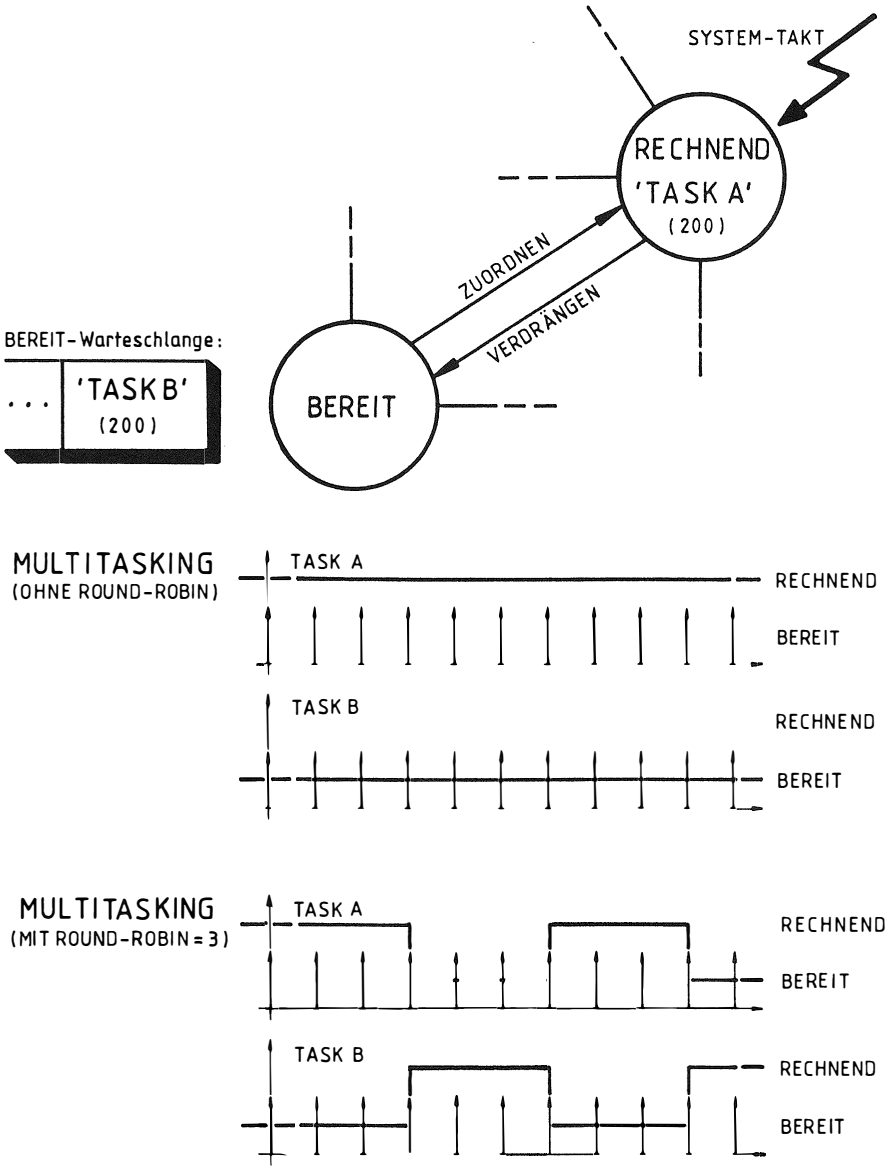
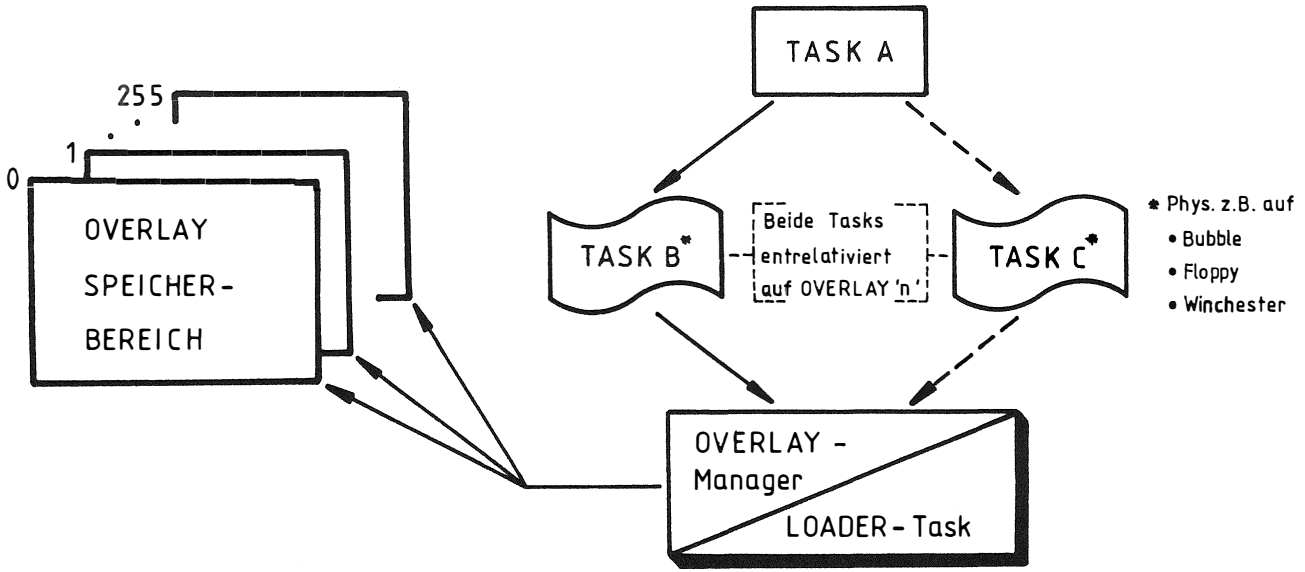


Bild 3. TIME - SHARING - CPU - ZUTEILUNG
DURCH ROUND-ROBIN-FUNKTION



ABLAUF:

- TASK A STARTET TASK B (oder TASK C)
- OVERLAY- Manager stellt fest, daß TASK B(C) nachgeladen werden muß / kann
- LOADER-Task wird gestartet .... diese lädt TASK B(C) in OVERLAY - Bereich
- LOADER-Task meldet Ladeende... TASK B(C) wird von RMOS2 "BEREIT" gesetzt

Bild 4. NACHLADEN ENTRELATIVIERTER TASKS

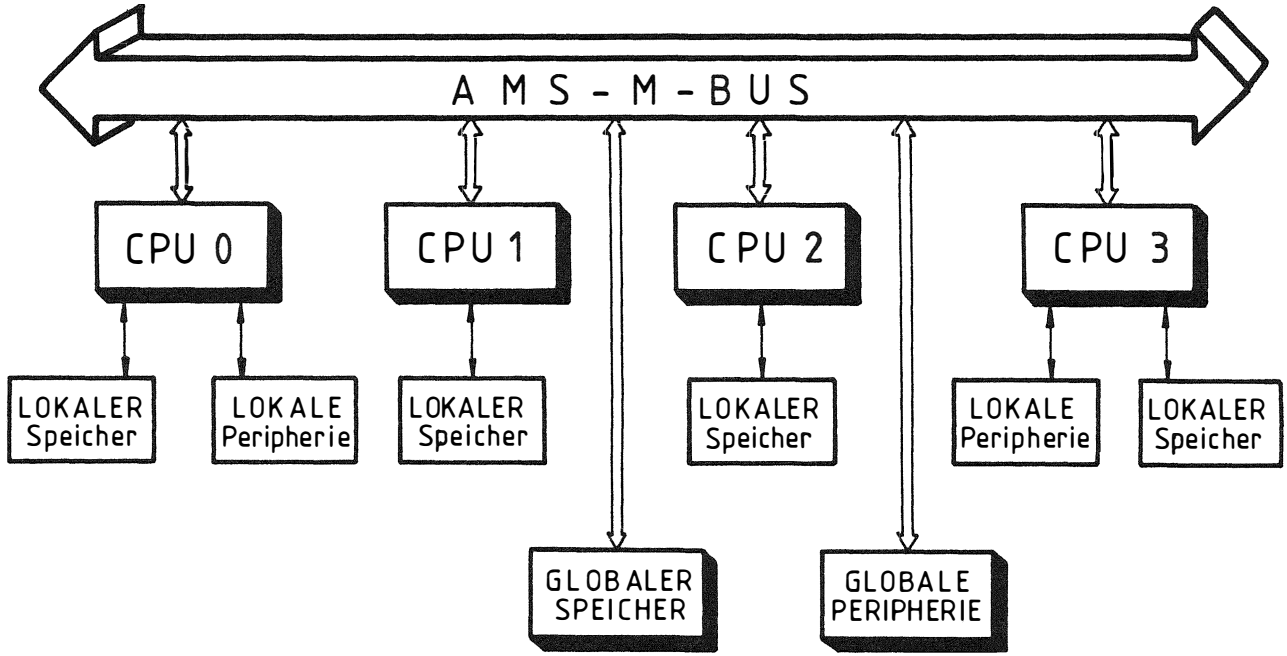


Bild 5. AMS - M - MULTIPROZESSOR - KONFIGURATION

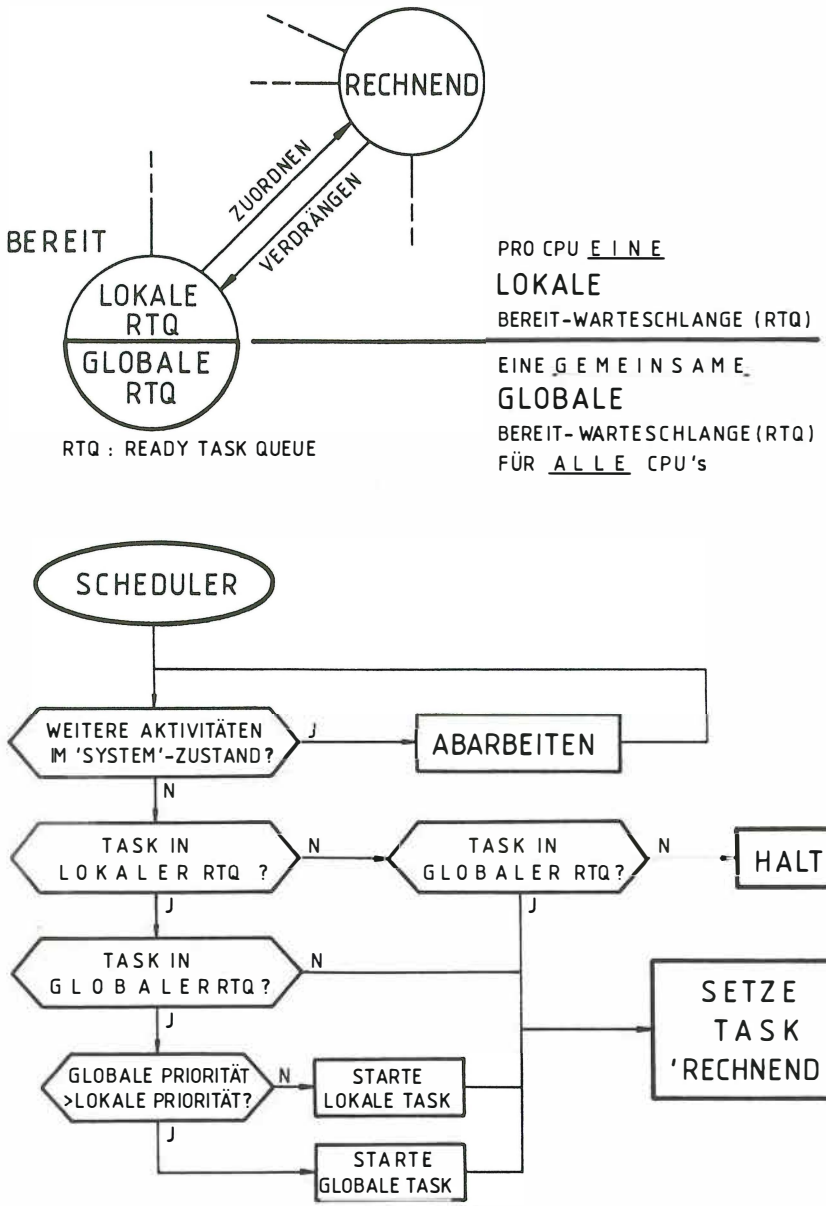
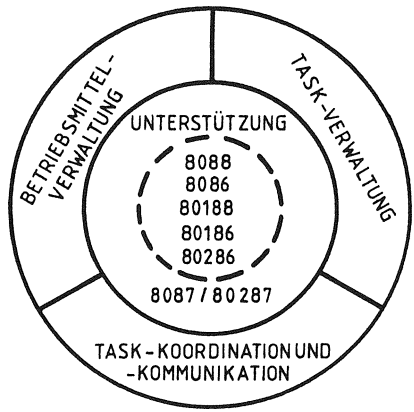


Bild 6. MULTIPROZESSOR - SCHEDULING - MECHANISMUS



TASK-VERWALTUNG		
TASK KREIEREN / LÖSCHEN	TASK AUSLÖSEN / UNTER-BRECHEN / BEENDEN INTERRUPT-TASK-START	TASK-PRIORITÄTS-STEUERUNG TASK-STATUS-ABFRAGE
CREATE DELETE DELTSK	STRT ENDT RESUME (ITS) QSTRT ENDR SETINT	(DYNAMISCHE PRIORITÄTSÄNDERUNG) CPRI GETSTAT

TASK-KOORDINATION UND -KOMMUNIKATION			
LOKALER BOTSCHAFTEN-VERKEHR	EREIGNIS-FLAGS	SEMAPHORE	KRITISCHE ABSCHNITTE
CREATEMBOX SEND DELETEMBOX RECV	CREATEFLAG SEFET SEF DELETEFLAG REF TEF WEF	CREATESEMA TSF DELETESEMA RSF	CNTRL

BETRIEBSMITTEL-VERWALTUNG			
ZEIT-VERWALTUNG	EIN/AUSGABE-VERWALTUNG	SPEICHER-VERWALTUNG	VERWALTUNG LOGISCHER BETRIEBSMITTEL-NAMEN
STIME PAUSE (SEFET) TIME (ENDR) (TIME-OUT's, DYN.-PRIO.-ÄND.)	CREATEDRIV CREATEDSCT DELETEDRIV DELETEDSCT RIO DSCRETE	CREATEPOOL CREATEOVL DELETEPOOL DELETEOVL ALOC DALOC (OVERLAYS)	CATALOG LOOK

Bild 7. SYSTEMDIENSTE DES RMOS2-NUKLEUS

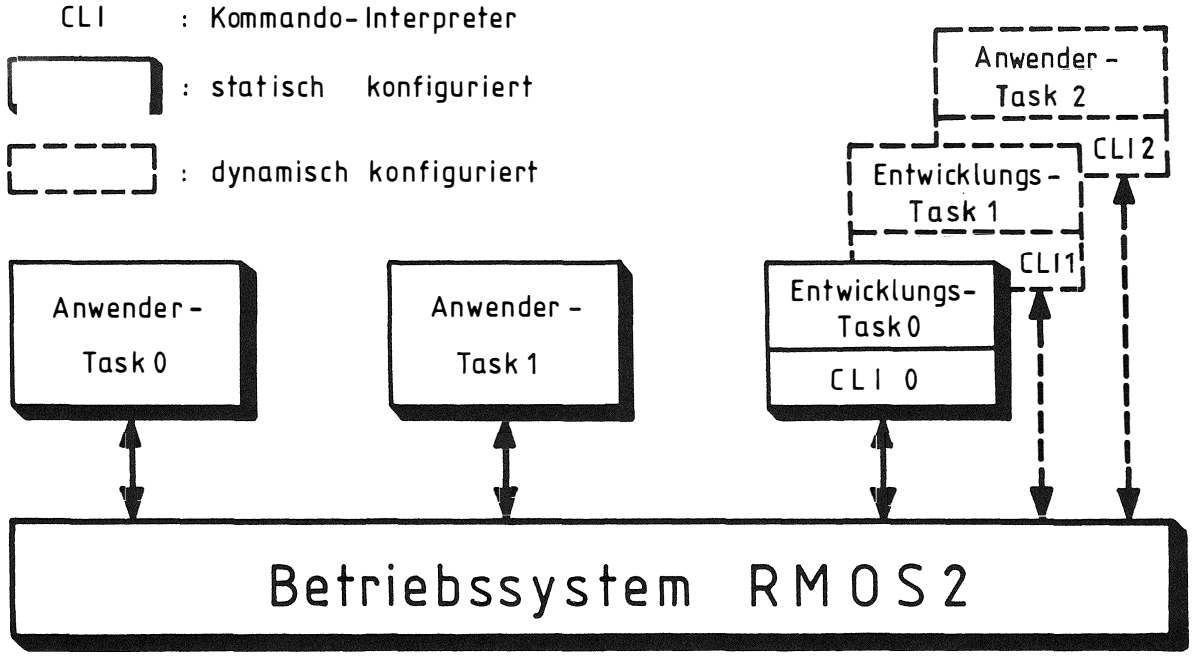


Bild 8. BEISPIEL EINER TASK-KONFIGURATION IN UDI-ENTWICKLUNGS-UMGEBUNG MIT RMOS2

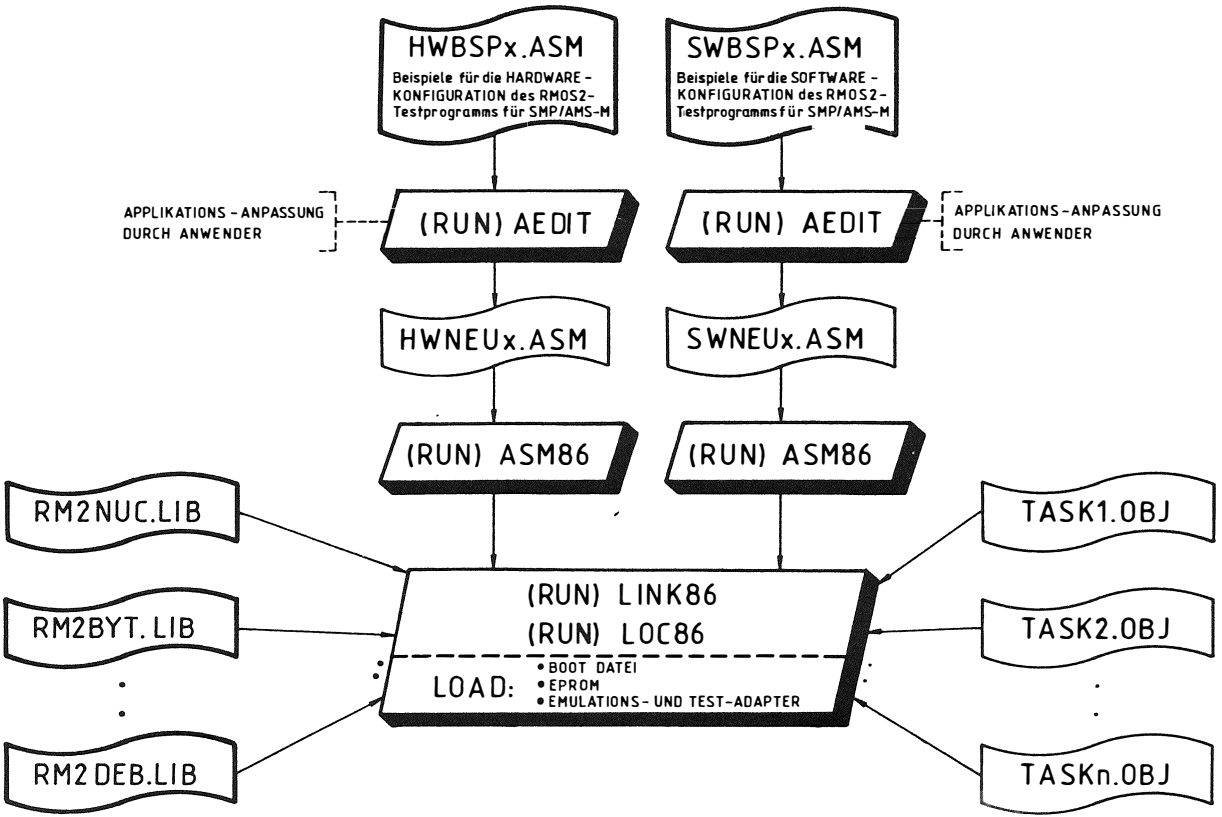


Bild 9. GENERIERUNG EINES RMOS2-SYSTEMS

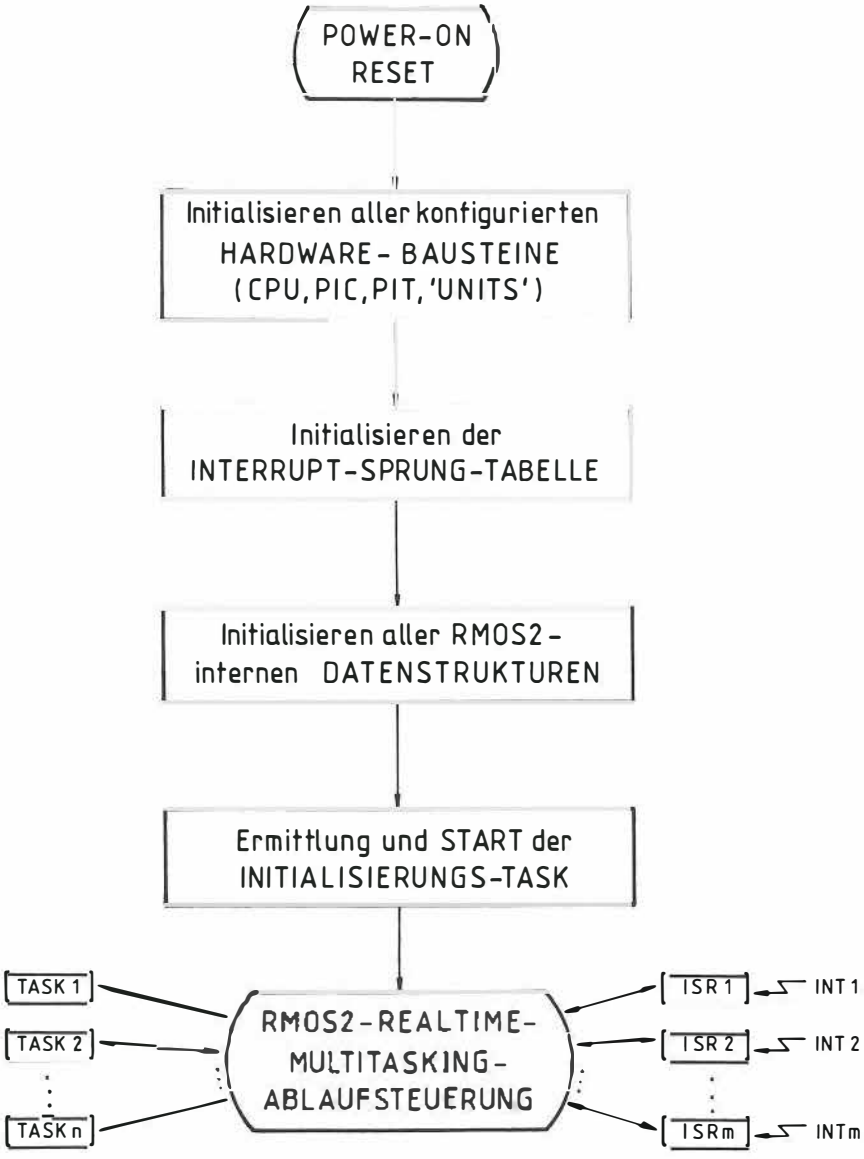


Bild 10. PRINZIP DES RMOS2-BOOT-VORGANGS