KERNFORSCHUNGSZENTRUM

KARLSRUHE

November 1975 KFK 2229

Institut für Datenverarbeitung in der Technik

Ein Hilfsmittel zur Simulation des dynamischen Verhaltens von Betriebssoftware

M. Rupp, J. Nehmer

GESELLSCHAFT
FÜR
KERNFORSCHUNG M.B.H.

KARLSRUHE

GESELLSCHAFT FÜR KERNFORSCHUNG M. B. H.

KARLSRUHE

KERNFORSCHUNGSZENTRUM KARLSRUHE

KFK 2229

Institut für Datenverarbeitung in der Technik

Ein Hilfsmittel zur Simulation des dynamischen Verhaltens von Betriebssoftware

M. Rupp
J. Nehmer

Gesellschaft für Kernforschung mbH, Karlsruhe

Kurzfassung

Der Aufsatz beschreibt die wesentlichen Merkmale eines Simulationssystems, das für die detaillierte Untersuchung des dynamischen Verhaltens von Betriebssoftware entwickelt wurde. Der Umgang mit dem System, das sich vom Standpunkt des Anwenders als eine virtuelle PL/I-Maschine darstellt, wird anhand eines Beispiels demonstriert.

Abstract

A tool for the simulation of the dynamic behavior of operating system software

The paper describes the characteristics of a simulation system to study the dynamic behavior of operating system software in detail. The use of the system which appears to the application programmer like a virtual PL/I-machine will be illustrated by an example.

1. Einleitung

Bei der Entwicklung von zeitkritischer Software steht der Systemingenieur häufig vor dem Problem, sich zwischen Alternativlösungen entscheiden zu müssen, deren dynamische Verhaltensweisen im Betrieb nicht vorhersagbar sind. Gerade bei der Konzipierung von Betriebssystemen bzw. Betriebssystemkomponenten hat er darauf zu achten, daß sich eine zur Erreichung eines vorgegebenen Betriebszieles, wie Durchsatzoptimierung, Reaktionszeitminimierung, Sicherheit usw. ausgewählte Strategie auch effizient realisieren läßt, d.h. keinen übermäßigen Systemverwaltungsaufwand produziert.

In den meisten Fällen erweist es sich als unmöglich, die Entwurfsalternativen im Betrieb zu erproben, so daß als Hilfsmittel zur Entscheidungsfindung gegenwärtig vorrangig die Simulation in Frage kommt.

Für eine relativ grobe Simulation von Betriebssystem-Software eignen sich die bekannten Simulationssprachen, wie GPSS, SIMULA-67, SIMSCRIPT usw. zwar recht gut, erweisen sich jedoch zur detaillierten Simulation des dynamischen Programmverhaltens als recht unkomfortabel und schlecht handhabbar.

Das hier vorgestellte Simulationssystem entstand aus der Forderung nach einem effizienten Werkzeug zur Simulation von Betriebssystem-Software. Es wurde im Institut für Datenverarbeitung in der Technik (IDT) im Kernforschungszentrum Karlsruhe entwickelt und trägt den Namen PLIM (PL/I-Maschine).

PLIM wird im IDT vornehmlich als Hilfsmittel

- zum Vergleich von Scheduling-Verfahren,
- zum Austesten von Software,
- zur näherungsweisen Bestimmung von Ausführungszeiten sowie zur Ermittlung anderer Meßgrößen verschiedener Entwurfsstrategien und
- zur Analyse dynamischer Abhängigkeiten innerhalb eines Systems

eingesetzt.

PLIM basiert auf einer frei wählbaren Anzahl von Rechnerkernen und verfügt über einen umfangreichen Instruktionsvorrat, der eine relativ komfortable Beschreibung und Simulation auch von komplexen Hardware- und Softwaresystemen ermöglicht. Mit Hilfe der Rechnerkerne lassen sich neben Zentraleinheiten (CPU's) auch alle anderen unabhängigen "Aktivitäten" eines DV-Systems, wie Kanäle, Laufwerke, Geräte, usw. simulieren.

Der Instruktionsvorrat jedes Rechnerkerns wird unterschieden in:

- Maschineninstruktionen, mit denen Rechnerkerne manipuliert und Programmabläufe gesteuert werden, und in
- Pseudoinstruktionen, die zur effizienten Handhabung des Simulationsinstrumentes dienen.

Beispielsweise stehen Pseudoinstruktionen zur Verfügung, die einen Wechsel zwischen unterschiedlichen Modellierungstiefen (mikroskopisch - makroskopisch) gestatten und damit zu einer wesentlichen Einsparung an Simulationsrechenzeit beitragen.

Die Rechnerkerne der PLIM werden bezüglich eines Simulationsgrundtaktes, dessen Maßeinheit modellspezifisch festzulegen ist, synchronisiert und verarbeiten simultan Instruktionen in Form von PL/I-Anweisungen.

Zu jeder Instruktion muß eine Ausführungszeit mitangegeben werden, die ein beliebiges Vielfaches des Simulations-Grundtaktes ist. Aufgrund der Ausführungszeiten wird die Simulationszeit fortgeschrieben, wobei die exakte Einhaltung zeitlich paralleler Abläufe garantiert wird.

Da die PLIM-Instruktionen aus PL/I-Statements (modifiziert durch Zeitbewertungen) bestehen und demnach das Modell durch ein PL/I Programmsystem nachgebildet wird, ergeben sich folgende Vorteile:

- Betriebssystemsoftware ist auf einem hohen sprachlichen Niveau formulierbar,
- Programme sind leicht lesbar und gut dokumentiert,
- Portabilität des Systems ist leicht möglich,
- Vielfalt der PL/I-Sprachelemente erlaubt Modellverfeinerungen und Vergröberungen und

- ausgereifte Compiler sind verfügbar, die Fehlererkennung erleichtern und laufzeitarmen Code produzieren.

2. Struktur der PLIM

2.1. Architektur der Rechnerkerne

Die Aufgabe der Rechnerkerne besteht darin, Modellprogramme, die aus PLIM-Instruktionen bestehen, auszuführen.

Die PLIM-Instruktionen werden - nach vorangehender Transformation der Zeitbewertungen in PL/I-Statements - direkt durchlaufen, so daß der Arbeitsspeicherbereich der PLIM gleich dem Speicherbedarf der PL/I-Anweisungen ist. Spezielle Zugriffsbeschränkungen bei simultanem Zugriff mehrerer Rechnerkerne zum Arbeitsspeicher sind nicht Bestandteil der PLIM.

Jeder Rechnerkern der PLIM besitzt eine Unterbrechungseingabe, die ihn bei Eintreffen eines Unterbrechungssignals (Interrupt) zwingt, nach Abarbeitung der aktuellen PLIM-Instruktion das aktive Modellprogramm zu verlassen und mit der Abhandlung des zugeordneten Unterbrechungsprogramms zu beginnen. Dieser Wechsel findet jedoch nur dann statt, wenn sich der Rechnerkern in unterbrechbarem Modus befindet.

Beim Programmwechsel bedient sich der Rechnerkern der drei folgenden, gleich strukturierten Registersätze, in denen der Programmstatus:

- des aktiven Programms (im Arbeitsregistersatz),
- des zuletzt unterbrochenen Programms (im Ablage-Registersatz) und
- des Unterbrechungs-Programms (im Unterbrechungs-Registersatz)

festgehalten ist.

Die Registersätze werden durch eintreffende Unterbrechungen auf folgende Weise manipuliert: Ein Rechnerkern führt grundsätzlich das Modellprogramm aus, dessen Programmstatus im Arbeitsregistersatz steht. Wenn ein Unterbrechungssignal zur Unterbrechung des Programmablaufes führt, wird automatisch

der Ablage-Registersatz durch den Arbeits-Registersatz und dieser dann durch den Unterbrechungs-Registersatz überschrieben. Damit wird das Unterbrechungsprogramm aktiv (Bild 1).

Daneben kann ein Statuswechsel nicht nur aufgrund eines Unterbrechungssignals, sondern auch mittels spezieller Maschineninstruktionen ausgelöst werden, wie das nachfolgende Beispiel zeigt.

Generell führt jeder Statuswechsel zu einer Veränderung des Arbeitsregistersatzes, womit der Rechnerkern automatisch die Programmausführung wechselt. Der Rechnerkern entnimmt die gesamte Steuerinformation dem Arbeitsregistersatz, wo alle Angaben über das auszuführende Modellprogramm und den resultierenden Ablaufmodus festgehalten sind.

Aus diesen beiden Teilen setzt sich jeder Registersatz zusammen, wobei im ersten Teil der Programmzustand durch

- die Identifikationsnummer des Modellprogramms,
- den aktuellen PLIM-Befehlszähler,
- den Verweis auf einen, dem Modellprogramm zugeordneten Datenbereich,
- die aufgelaufene CPU-Zeit und
- die Priorität des Modellprogramms

spezifiziert ist.

Der Teil des Registersatzes, der den Ablaufmodus des Rechnerkerns bei Abarbeitung des Modellprogramms beschreibt, gibt an

- ob ein Unterbrechungssignal empfangen oder abgewiesen wird,
- ob ein empfangenes Signal das aktive Modellprogramm momentan unterbrechen kann oder nicht und
- in welcher Betriebsart der Rechnerkern gerade arbeitet.

PLIM kennt vier Betriebsarten, in denen sich ein Rechnerkern während des Simulationslaufes befinden kann. Ein Rechnerkern ist entweder:

- aktiv, wenn er gerade eine PLIM-Instruktion bearbeitet, oder
- verzögert, wenn er auf ein Synchronisierungssignal wartet, oder
- pseudo-aktiv, wenn die Ausführung von Programmen makroskopisch durch ein vorgegebenes Zeitintervall simuliert wird, oder schließlich
- untätig, wenn er die Programmabarbeitung bis zum Eintreffen der nächsten Unterbrechung aussetzt.

Der Ablaufmodus wird bei Programminitialisierung festgelegt und läßt sich durch spezielle Maschineninstruktionen, wie $$MASK, $$ENABLE, $$DISABLE, $$IDLE, $$LOCK, $$UNLOCK u.a., verändern.

2.2. Unterbrechungs-Konzept

Das der PLIM zugrundeliegende Konzept zur Unterbrechungsabarbeitung sieht vor, daß jedem Rechnerkern bei Modellinitialisierung ein Modellprogramm als Unterbrechungsprogramm fest zugeordnet wird, das nach jeder angenommenen Unterbrechung über einen Statuswechsel neu aktiviert wird.

Jeder Rechnerkern verfügt über einen Puffer, in den die an ihn gerichteten Unterbrechungssignale - sofern er unmaskiert ist - prioritätsgeordnet abgelegt werden.

Unterbrechungen werden entweder durch bestimmte PLIM-Instruktionen oder bei Konsolstart (Rechnerkerninitialisierung) und Modellprogrammende erzeugt.

Als unterbrechungsauslösende PLIM-Instruktionen stehen die beiden Maschineninstruktionen

- $$SVC : Unterbrechung an den eigenen Rechnerkern (Supervisor-Call)
- $$SIGNAL: Unterbrechung an einen fremden Rechnerkern

und die Pseudoinstruktion

- $$INT : Unterbrechung an den eigenen Rechnerkern nach Ablauf einer angegebenen Zeit

zur Verfügung.

Neben einer Priorität können den Unterbrechungsinstruktionen als weitere Parameter ein Motiv zur groben Unterscheidung der Unterbrechungsart und eine Parameterablageadresse zur Adressierung einer unterbrechungsspezifischen Parameterliste mitgegeben werden.

Wird das Unterbrechungsprogramm aktiv, so kann es mit Hilfe von Maschineninstruktionen auf die zum auslösenden Interrupt gehörigen Parameter zugreifen und eine gezielte Interrupt-Bearbeitung vornehmen.

Auf der Basis dieser Verarbeitung bietet die PLIM dem Anwender die Möglichkeit

- die unterschiedlichsten Rechner- und Geräteunterbrechungswerke nachzubilden,
- Interprozessorkommunikation zu simulieren und
- zeitlich unbestimmte, asynchrone Unterbrechungen zu generieren.

2.3. Aufbau eines Modellprogramms

Ein Modellprogramm wird statisch unterteilt in den

- Deklarationsteil und den
- Ausführungsteil.

Die Forderung nach klarer Trennung von Daten- und Instruktionsteil ermöglicht den Aufbau ablaufinvarianter Modellprogramme. Dadurch wird gewährleistet, daß ein und dasselbe Modellprogramm auch von verschiedenen Rechnerkernen mit unterschiedlichen Datenbereichen ausgeführt werden kann.

Dieses Konzept erfordert vom Anwender bei der Erstellung des Deklarationsteiles die Beachtung einiger Vorschriften (z.B. Verwendung von Based-Pointern).

Der Ausführungsteil eines Modellprogramms besteht aus einer Folge von PLIM-Instruktionen. Eine PLIM-Instruktion setzt sich zusammen aus einer

- vorangestellten Zeitbewertung, die eine Konstante oder Variable sein kann und durch ein führendes '$'-Zeichen gekennzeichnet sein muß, und aus
- einem oder mehreren Ausführungsanweisungen.

Als Ausführungsanweisungen gelten die üblichen PL/I-Anweisungen oder die von der PLIM zur Verfügung gestellten Maschinen- und Pseudoinstruktionen, z.B.:

```
          :
$5        A = A+1;
          GOTO M1;
          MO:
$1        $$IDLE;
          :
```

Eine PLIM-Instruktion ist ununterbrechbar, sie beginnt mit einer Zeitbewertung und endet vor Überlaufen der nächsten Zeitbewertung im dynamischen Programmablauf.

Eine genaue Beschreibung des Modellprogrammaufbaus und der vorhandenen Maschinen- und Pseudoinstruktionen wird in /1/ gegeben. Einige typische Instruktionen werden im folgenden Demonstrationsbeispiel verwendet und dabei ausführlicher erläutert.

2.4. Ablauf eines Modellprogramms

Die Simulation zeitlich paralleler Programmabläufe wird dadurch ermöglicht, daß PLIM-Instruktionen, die simultan von verschiedenen Rechnerkernen ausgeführt werden, zeitlich unabhängig zum Ablauf gebracht werden. Für den zeitgerechten Aufruf der PLIM-Instruktionen wird ein in die PLIM integrierter Ereignismechanismus /2/ verwendet. Er besitzt eine Zeitachse, auf der für jeden Rechnerkern vermerkt ist, wann die nächste PLIM-Instruktion auszuführen ist.

Um die schritthaltende Abarbeitung der Modellprogramme zu ermöglichen, wird jedes Modellprogramm von einem systemzugehörigen Vorübersetzer modifiziert und aufgeweitet. Der Vorübersetzer fügt überall dort, wo die Zeitbewertungen stehen, Einsprung-

(entry) und Aussprunganweisungen (return) ein und wandelt alle PLIM-spezifischen Sprachelemente in gültige PL/I-Anweisungen um. Als Produkt des Vorübersetzers entsteht ein syntaktisch korrektes PL/I-Programm, das nach Übersetzung in das vorhandene PLIM-Simulationssystem eingebunden wird.

Insgesamt besteht das Simulationssystem aus einem hierarchisch gegliederten System von PL/I-Prozeduren, die - wie auch die einzelnen PLIM-Instruktionen - über Unterprogrammaufrufe (call) aktiviert werden.

Der Anwender der PLIM hat neben den Modellprogrammen, mit denen er sein Modell realisiert, noch das Hauptprogramm (procedure options (main)) zu erstellen, das zumindest den Aufruf für den Start der Simulation enthalten muß.

In der Regel wird der Anwender im Hauptprogramm vor dem Startaufruf die Initialisierung und nach Abbruch der Simulation die Auswertung der Meßergebnisse vornehmen.

3. Beispiel

Das Demonstrationsbeispiel beinhaltet den Vergleich zweier von Denning /3/ angegebener Strategien des Trommelzugriffs in einem Ein-Prozessorsystem.

Es soll einen Einblick vermitteln, wie Modellierung und Simulation eines in sich abgeschlossenen Problems aus dem Gebiet der Systemsoftware mit Hilfe von PLIM gelöst wird.

3.1. Problemanalyse

Denning zeigt, wie man durch gezielte Verwaltung der E/A-Aufträge Zugriffszeiten zur Trommel verbessern kann.

Bild 2 zeigt schematisch die Systemkonfiguration der Problemstellung, wobei davon auszugehen ist, daß alle E/A-Aufträge mittels eines SVC (Supervisor-Call) eingeleitet werden. Das SVC-Programm hat die Aufgabe, die E/A-Aufträge zu verwalten und den Kanal zur Datenübertragung anzustoßen. Das Ende der Übertragung wird durch ein Unterbrechungssignal quittiert.

Im folgenden wird davon ausgegangen, daß die Spuren der Trommel in gleich große Sektoren eingeteilt sind, und daß für jeden Sektortransfer ein E/A-Auftrag abzugeben ist.

Das aufgestellte Modell beschränkt sich auf die Betrachtung einer Spur der Trommel und wird ohne Unterscheidung der Transferrichtung (Ein- oder Ausgabe) durchgeführt.

Die beiden Strategien, die verglichen werden, heißen

- FCFS (First Come - First Serve) und
- SATF (Shortest Access Time First).

Bei der Strategie FCFS werden alle E/A-Aufträge in eine Warteschlange eingeordnet und entsprechend der resultierenden Reihenfolge abgearbeitet. Es wird gezeigt, daß die mittlere Zugriffszeit, die bei Verwendung von FCFS bei einer halben Trommelumdrehungszeit liegt, bei Anwendung von SATF erheblich vermindert werden kann.

Bei der Strategie SATF werden die E/A-Aufträge in sektorspezifischen Warteschlangen gesammelt, die dann unter Berücksichtigung der Trommelumdrehungsgeschwindigkeit entsprechend der periodischen Sektorsequenz abgearbeitet werden.

3.2. Abbildung auf die PLIM

Bei der Abbildung der realen Konfiguration von Prozessor, Programmen, Kanal und Trommel auf die PLIM ergibt sich folgender Modellaufbau (Bild 3).

Der Prozessor wird durch einen Rechnerkern simuliert, dessen Unterbrechungsprogramm entsprechend dem SVC-Programm die E/A-Aufträge verwaltet und an die Trommel weitergibt.

Neben dem Unterbrechungsprogramm führt der Rechnerkern nur noch ein Modellprogramm aus, das die Programme simuliert, die im System E/A-Aufträge an die Trommel absenden. Das Modellprogramm produziert in einer Schleife nach Ablauf von zufallsverteilten Zeiten E/A-Aufträge, wobei die Sektornummern durch einen Zufallszahlengenerator erzeugt werden.

Hierzu stellt die PLIM einen Satz von Zufallszahlengeneratoren mit verschiedenen Verteilungsfunktionen zur Verfügung, die wie Pseudoinstruktionen aufrufbar sind.

Der Kanal wird im Modell nicht nachgebildet, da bei der Problemstellung der Datentransport unberücksichtigt bleibt. Die Übermittlung der Aufträge an die Trommel und die Rücksendung der Quittung wird durch die Maschineninstruktion $$SIGNAL bewerkstelligt.

Die Trommel kann aufgefaßt werden als unabhängiger Spezialprozessor mit zugeordnetem Speicher und festverdrahtetem Programm, so daß sie durch einen eigenen Rechnerkern mit einem zugehörigen Modellprogramm simuliert werden kann. Das Modellprogramm hat die Aufgabe, die Position des Lese- Schreibkopfes während der Simulation zu verfolgen und ein E/A-Ende-Signal nach Durchlauf des referierten Sektors an den Prozessor zu melden.

Zwecks Einsparung von Rechenzeit sieht die vorliegende Realisierung die Aktivierung des Modellprogramms ausschließlich bei Auftragsannahme und -ende vor. Diese Strategie erfordert die Berechnung der aktuellen Position des Lese- Schreibkopfes und verlangt die Kenntnis der letzten Position und der seither verstrichenen Zeit.

3.3. Modellablauf

Ein Simulationssystem muß ein Instrument zur Verfolgung des dynamischen Modellablaufes zur Verfügung stellen. Diese Aufgabe kann in der PLIM durch einen vom eigentlichen Untersuchungsobjekt unabhängigen Rechnerkern durchgeführt werden, der die Rolle eines Monitors übernimmt. Dem Rechnerkern wird ein Modellprogramm zugeordnet, das mittels der bereitgestellten Pseudoinstruktionen eine rückwirkungsfreie Verfolgung der dynamischen Vorgänge im Modell zu jedem Zeitpunkt ermöglicht.

Unter Einbeziehung des Monitorrechners in das Modell ergibt sich die im Bild 4 dargestellte Modell-Konfiguration.

Im Anhang B sind die Modellprogramme zur Implementierung des Modells bzgl. der Strategie FCFS aufgeführt. Aus der Modellkonfiguration sind die folgenden Namenszuordnungen ersichtlich:

- Rechnerkern 1 simuliert den Prozessor und führt das SVC-Programm MP_1 und das auftragsproduzierende Programm mit dem Namen MP_2 aus.
- Rechnerkern 2 simuliert die Trommel mit Hilfe des Unterbrechungs-Programms MP_3.
- Rechnerkern 3 führt als Monitor das Überwachungsprogramm MP_4 aus.

Der Ablauf der Simulation beginnt mit der Aktivierung des Hauptprogramms (s. Anhang B).

Im Hauptprogramm MAIN werden die Modellbeschreibungsgrößen (Anzahl der Sektoren, Durchlaufzeit eines Sektors, Auftragsverteilungen) eingelesen, die Meßgrößen (mittl. Verweilzeit, mittl. Zugriffszeit) initialisiert und die Simulation gestartet. Nach Simulationsende erfolgt die Auswertung und Ausgabe der Meßwerte. Anhang A zeigt das Protokoll des Simulationslaufes (Strategie FCFS) und verdeutlicht den PLIM-gesteuerten Konsol-Start der Rechnerkerne. Das folgende Tracing wird über einen Dialog mit dem Benutzer vom Modellprogramm MP_4 bewirkt. Das Protokoll vermittelt einen Einblick in den Simulationsablauf, indem es in der Reihenfolge der Abarbeitung für jeden Rechnerkern angibt:

- welches Programm er bearbeitet,
- welche PLIM-Instruktion er gerade ausführt und
- wieviel CPU-Zeit er bisher für das Modellprogramm aufgewendet hat.

Mit dem Startaufruf des Hauptprogramms erfolgt der Konsolstart, wodurch die Unterbrechungs-Programme der drei Rechnerkerne mit Übergabe des Konsolstart-Motivs 2 aktiviert werden. Im Rechnerkern 1 beginnt das Unterbrechungs-Programm MP_1 aufgrund des von $$IT_MOT erhaltenen Unterbrechungs-Motives mit der Verzweigung zur Modellinitialisierung. Dort wird der Registersatz für das auftragsproduzierende Modellprogramm MP_2 erstellt, der

danach durch die Maschineninstruktion $$RESUME aktiviert wird.

$$RESUME bewirkt den Transfer eines vom Benutzer verwalteten Registersatzes auf den Arbeits-Registersatz und kann zusammen mit der Komplementärinstruktion $$SAVE, die den Ablage-Registersatz dem Anwender übergibt, als Instrument zum Parallelbetrieb von Prozessen verwendet werden.

Im Modell gibt nun der Rechnerkern die Abarbeitung von MP_1 auf und beginnt mit der Durchführung von MP_2.

MP_2 veranlaßt über $$RUN, daß der Rechnerkern für eine - nach einer Binominialverteilung ($$BINOM) ermittelten - Zeit in die Pseudoaktiv-Phase eintritt. Mit Hilfe der Pseudoinstruktion $$RUN lassen sich Programmsequenzen simulieren, die - abgesehen von ihrem Zeitbedarf - keinen Einfluß auf die vorliegende Modellogik ausüben. $$RUN ermöglicht somit die Einbettung von makroskopischen Modellabläufen an allen Stellen des Simulationsmodells, an denen eine feinere Auflösung unerwünscht ist. Nach Ablauf der Pseudoaktiv-Phase produziert MP_2 einen E/A-Auftrag, dessen Sektornummer sich aus einer Zufallszahl ergibt, die im angegebenen Bereich einer Gleichverteilung ($$RADIS) unterliegt. Danach übermittelt MP_2 den Auftrag über $$SVC an das Unterbrechungs-Programm MP_1.

Die Maschineninstruktion $$SVC bewirkt ein Überschreiben des Ablage-Registersatzes durch den Arbeitsregistersatz und des Arbeitsregistersatzes durch den Unterbrechungs-Registersatz. Damit beginnt der Rechnerkern erneut mit der Abarbeitung von MP_1.

Aus dem Unterbrechungs-Motiv 3 erkennt MP_1 den SVC und erzeugt ein Warteschlangenelement. MP_1 entnimmt über $$IT_ADR die mitgegebene Sektornummer und ordnet das Element in die der Strategie entsprechenden Warteschlange ein. Für den Fall, daß die Trommel gerade aktiv ist, sorgt MP_1 über die Maschineninstruktion $$RETURN für die Fortsetzung des unterbrochenen Modellprogramms MP_2.

$$RETURN transferiert den Ablage-Registersatz in den Arbeits-Registersatz und veranlaßt somit die Reaktivierung des zuletzt unterbrochenen Programms.

Ist die Trommel inaktiv, so sendet MP_1 mit Hilfe der Maschinen-instruktion $$SIGNAL ein mit der Sektornummer als Parameter versehenes Unterbrechungssignal an den die Trommel simulieren-den Rechnerkern 2. Unter Bezugnahme auf die oben beschriebene Strategie weckt das Unterbrechungssignal von MP_1 den Rechner-kern 2. Im Rechnerkern 2 wird das Unterbrechungsprogramm MP_3 aktiv und berechnet unter Verwendung der Pseudoinstruktion $$CLOCK, die die aktuelle Simulationszeit bereitstellt, die gegenwärtige Position des Lese- Schreibkopfes.

Entsprechend der Zeitdifferenz bis zum Ende des gewünschten Transfers arbeitet der Rechnerkern 2 das Programm MP_3 ab und meldet schließlich über $$SIGNAL das Transferende an den Rech-nerkern 1.

Danach setzt MP_3 den Rechnerkern 2 mittels der Maschinenin-struktion $$IDLE in den Ruhestand, aus dem der Rechnerkern nur über ein Unterbrechungssignal geweckt werden kann.

Die Trommelunterbrechung unterbricht entweder den Rechnerkern 1 in der Abarbeitung von MP_2, oder wird in dessen Unterbrechungs-Puffer eingeordnet, wenn der Rechnerkern - aufgrund eines $$SVC - gerade das Unterbrechungsprogramm abarbeitet.

Aus dem von MP_3 gesetzten Unterbrechungs-Motiv erkennt MP_1 das Übertragungsende und entfernt den abgeschlossenen Auftrag aus seiner Buchführung. Danach ermittelt MP_1 - entsprechend der Strategie - den nächsten Auftrag, übermittelt ihn durch $$SIGNAL an die Trommel und sorgt durch $$RETURN für die Fortsetzung von MP_2.

Unabhängig von der in den Rechnerkernen 1 und 2 ablaufenden Trommelzugriffs-Simulation läuft im Rechnerkern 3 das Überwa-chungsprogramm ab. Das Überwachungsprogramm wurde als Unter-brechungsprogramm MP_4 des Rechnerkerns 3 durch Konsol-Start aktiviert und führt einen Dialog mit dem Anwender. MP_4 sorgt nach den Angaben des Anwenders

- für den Simulationsabbruch durch die Pseudoinstruktion $$STOP und
- für ein Tracing des Modellablaufs durch die Pseudoinstruktion $$TEST in den gewünschten Zeitintervallen.

MP_4 weckt den eigenen Rechnerkern nach Ablauf der vom Anwender angegebenen Zeiten durch die Pseudoinstruktion $$INT, durch die ein zeitverzögertes Unterbrechungssignal an den eigenen Rechnerkern abgesetzt wird. Der Anwender hat dann die Möglichkeit, das im Anhang A gezeigte Tracing des Modellablaufes zu starten ($$TEST('B')) oder abzubrechen ($$TEST('E')).

In diesem Zusammenhang soll noch auf die Pseudoinstruktion $$RKDUMP hingewiesen werden, die für die Auflistung der Registerinhalte aller Rechnerkerne sorgt. $$RKDUMP und $$TEST ermöglichen ein rasches Auffinden von Fehlern in der Ablauflogik des Modells und haben sich in der Praxis als äußerst nützliche Simulationswerkzeuge bewährt.

4. Erfahrungen

Für das gesamte Simulationssystem, einschließlich Vorübersetzer, wurden ca. 5000 PL/I-Statements codiert. Zusätzlich wurden für die interaktive Bedienung des Systems unter TSO (Time Sharing Option) auf der IBM 370/168 ein Satz von Kommandoprozeduren erstellt, mit deren Hilfe ein Anwender - ohne über nähere Systemkenntnisse zu verfügen - leicht

- Modellprogramme übersetzen,
- Objektmodule in das Simulationssystem einbinden, ergänzen oder austauschen

und

- den Simulationslauf starten kann.

Mit einer Rechenzeit von ca. 150 CPU-Stunden wurden bisher unterschiedliche Varianten eines Dispatchers in einem Mehrprozessorsystem mit bis zu 16 Prozessoren simuliert und das Verhalten bei Prozessor-Thrashing untersucht /4/. Gegenwärtig befindet sich ein Modell für ein symmetrisches 3-Rechnernetz im Aufbau, an dem

die Software für die Koordinierung der drei Rechner im Falle eines redundanten 2 von 3-Betriebes entwickelt und erprobt werden soll. Der Quotient: aufgewendete reale Rechenzeit/simulierte Zeit, der ein Maß für die Effizienz des Simulationssystems darstellt, hat trotz der maschinennahen Darstellung der untersuchten Betriebssoftware den Wert 70 je Rechnerkern nicht überschritten.

5. Literaturverzeichnis

/1/ M. Rupp, J. Nehmer, G. Fleck, D. Hilse, R. Friehmelt
PLIM - Eine virtuelle PL/I-Maschine
KFK 1712, Kernforschungszentrum Karlsruhe, Juli 1973

/2/ J. Nehmer, D. Hilse, M. Rupp
Ein PL/I Unterprogrammpaket für die diskrete Eventsimulation
KFK 1711, Kernforschungszentrum Karlsruhe, Januar 1973

/3/ P.J. Denning
Effects of scheduling on file memory operations
Spring Joint Computer Conference, 1967

/4/ J. Nehmer
Dispatcher-Elementarfunktionen für symmetrische Mehrprozessor-DV-Systeme
Dissertation an der Fakultät für Informatik der Universität Karlsruhe, Juli 1973

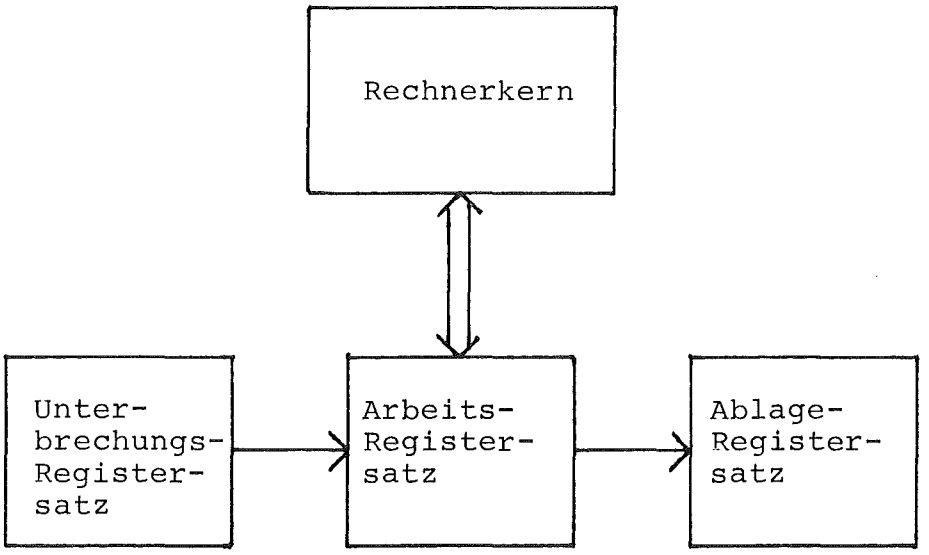

Bild 1. Unterbrechungs-Bearbeitung

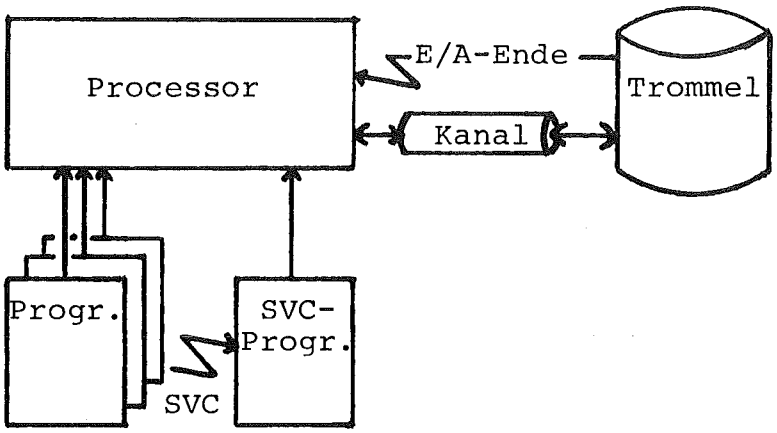

Bild 2. Demonstrationsmodell

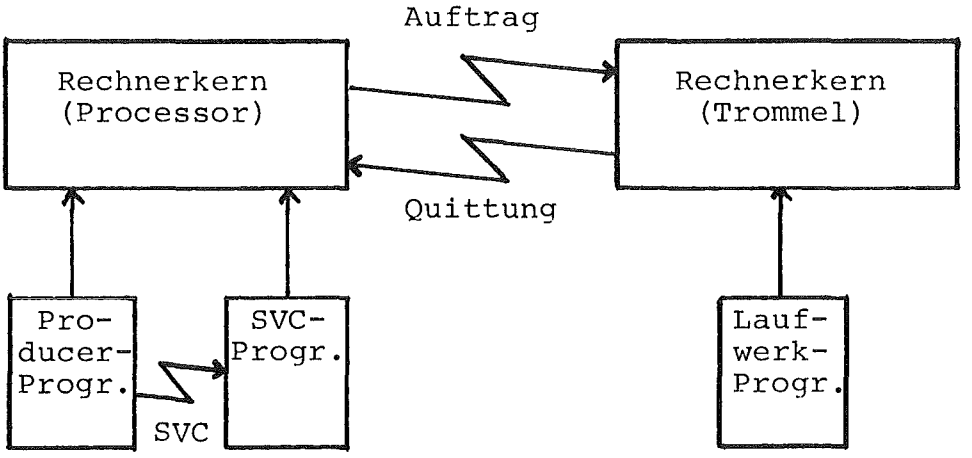

Bild 3. Modellaufbau

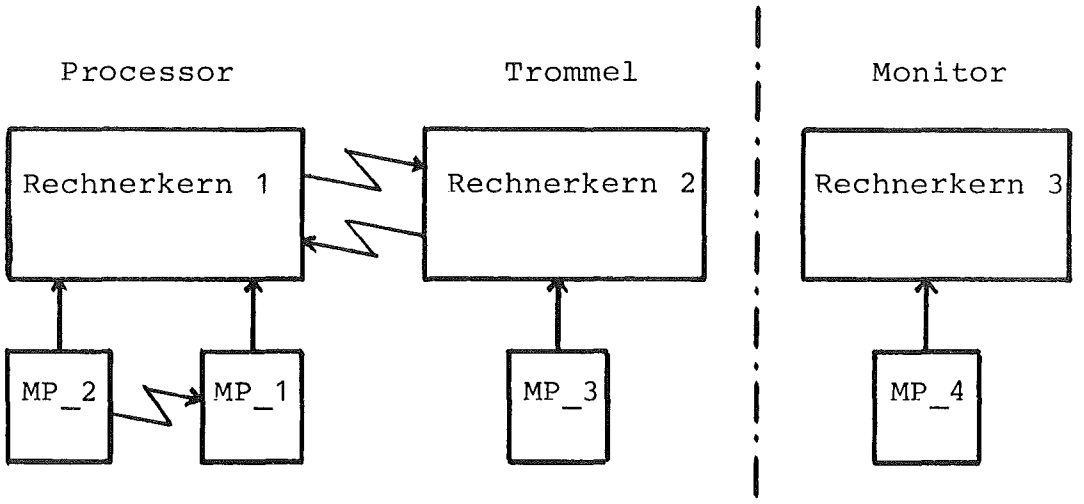

Bild 4. Modellkonfiguration

Anhang A:

Ablaufprotokoll

(Eingabe ist unterstrichen)

```
GIB ANZAHL DER SEKTOREN PRO SPUR: 64
GIB DURCHLAUFZEIT EINES SEKTORS:  260
GIB ZEITANTEIL FUER DEN DATENTEIL: 250
GIB SEKTORANSTEUERUNGS-DIFFERENZ (BEI SATF): 0
GIB MAX. ANKUNFTSINTERVALL DER ANFORDERUNGEN: 10000

GIB ANZAHL RECHNERKERNE: 3
LADE RECHNERKERN NR  1
GIB MP_NR DES INTERRUPT-PROGRAMMS: 1
GIB PRIORITAET: 3
LADE RECHNERKERN NR  2
GIB MP_NR DES INTERRUPT-PROGRAMMS: 3
GIB PRIORITAET: 2
LADE RECHNERKERN NR  3
GIB MP_NR DES INTERRUPT-PROGRAMMS: 4
GIB PRIORITAET: 1

START DES SIMULATIONSLAUFES

RECHNERKERN NR  1  'START' OR 'IDLE'?: 'START'
RECHNERKERN NR  2  'START' OR 'IDLE'?: 'START'
RECHNERKERN NR  3  'START' OR 'IDLE'?: 'START'

GIB SIMULATIONSDAUER: 500000
TRACING? -'JA'/'NEIN' ?: 'NEIN'
WIELANGE NICHT?: 15000
TRACING? -'JA'/'NEIN' ?: 'JA'
WIELANGE?: 331

        CLOCK    RK_NR     PROG_N      INDEX     STMT     CPUTME

        15080        2          3          2        4      10052
        15092        1          2          1        2      15025
        15102        1          2          1        3      15035
        15102        1          1          4        0          0
        15102        1          1          4        1          0
        15112        1          1          4        3         10
        15115        1          1          4        4         13
        15130        1          1          4        6         28
        15131        1          2          1        1      15036
        15330        2          3          2        5      10302
        15330        1          1          5        0          0
        15330        1          1          5        1          0
        15331        3          4          3        0          0
        15331        3          4          3        3          0

TRACING? -'JA'/'NEIN' ?: 'NEIN'
WIELANGE NICHT?: 500000

SIMULATIONSABBRUCH
        CLOCK    RK_NR     PROG_N      INDEX     STMT     CPUTME
       500000        3          4          4        1          0
ENDE DES SIMULATIONSLAUFES

AUSWERTUNG:
ANZAHL DER E/A-AUFTRAEGE:                98
ANZAHL DER BEARBEITUNGEN:                53
MITTL. VERWEILZEIT:                 127183.4
MITTL. SYSTEMBEARB.-ZEIT:              65.5
MITTL. ZUGRIFFSZEIT:                 8905.1
```

Anhang B:

Simulationsmodell

```
MAIN: PROC OPTIONS(MAIN);          /* HAUPTPROGRAMM TROMMEL-SIMULATION */

DCL PLIMCTL ENTRY;
DCL (PSKTW,PRUNW) POINTER EXT,
    1 TROMMEL EXT,                 /* TROMMELBESCHREIBUNG                */
      2 SKT_DLZ BIN FIXED(31),      /* DURCHLAUFZEIT DES SEKTORS          */
      2 SKT_DTT BIN FIXED(31),      /*    '' DES DATENTEILES              */
      2 SKT_ANZ BIN FIXED(15);      /* ANZAHL DER SEKTOREN (JE SPUR)      */
DCL 1 MESSGR EXT,                  /* MESSGROESSEN                       */
      2 ANZ_AUFTR BIN FIXED(31),    /* ANZAHL DER E/A-AUFTRAEGE           */
      2 ANZ_BEARB BIN FIXED(31),    /* ANZAHL DER BEARBEITUNGEN           */
      2 S_VERWZ   BIN FLOAT,        /* SUMME DER AUFTR.-VERWEILZEITEN     */
      2 S_SYSTZV  BIN FLOAT,        /* 1.SUMME DER SYSTEMBEARB.-ZEITEN    */
      2 S_SYSTZN  BIN FLOAT,        /* 2.SUMME DER SYSTEMBEARB.-ZEITEN    */
      2 S_ZGRFZ   BIN FLOAT;        /* SUMME DER ZUGRIFFSZEITEN           */
DCL N BIN FIXED(31),
    P BIN FLOAT;
DCL ANSTDF BIN FIXED EXT;          /* DIFFERENZ VOM AKTUELLEN SEKTOR ZUM
                        NAECHSTEN ANZUWAHLENDEN SEKTOR - NUR BEI SATF */

PUT SKIP EDIT('GIB ANZAHL DER SEKTOREN PRO SPUR:')(A);
GET LIST(SKT_ANZ);
PUT EDIT('GIB DURCHLAUFZEIT EINES SEKTORS:')(A);
GET LIST(SKT_DLZ);
PUT EDIT('GIB ZEITANTEIL FUER DEN DATENTEIL:')(A);
GET LIST(SKT_DTT);
PUT EDIT('GIB SEKTORANSTEUERUNGS-DIFFERENZ (BEI SATF):')(A);
GET LIST(ANSTDF);
PUT EDIT('GIB MAX. ANKUNFTSINTERVALL DER ANFORDERUNGEN:')(A);
GET LIST(N);
P=.5;
PRUNW=$$R_CRT(N,P);                /* INITIALISIERUNG BINOMIALVERTEILUNG */
N=SKT_ANZ-1;
PSKTW=$$R_CRT(N);                  /* INITIALISIERUNG GLEICHVERTEILUNG   */
ANZ_AUFTR,ANZ_BEARB=0;
S_VERWZ,S_SYSTZV,S_SYSTZN,S_ZGRFZ=0;

CALL PLIMCTL;    /* AUFRUF DER PLIM ZUR DURCHFUEHRUNG DER SIMULATION */

PUT SKIP(2) EDIT('AUSWERTUNG:')(A);
PUT SKIP EDIT('ANZAHL DER E/A-AUFTRAEGE:',ANZ_AUFTR)(A(30),F(10));
PUT SKIP EDIT('ANZAHL DER BEARBEITUNGEN:',ANZ_BEARB)(A(30),F(10));
S_VERWZ=S_VERWZ/ANZ_BEARB;
S_SYSTZN=S_SYSTZN/ANZ_BEARB;
S_SYSTZV=S_SYSTZV/ANZ_AUFTR;
S_ZGRFZ=S_ZGRFZ/ANZ_BEARB;
PUT SKIP EDIT('MITTL. VERWEILZEIT:',S_VERWZ)(A(30),F(10,1));
PUT SKIP EDIT('MITTL. SYSTEMBEARB.-ZEIT:',S_SYSTZV+S_SYSTZN)
            (A(30),F(10,1));
PUT SKIP EDIT('MITTL. ZUGRIFFSZEIT:',S_ZGRFZ)(A(30),F(10,1));
END MAIN;
```

```
MP_1: PROC;                    /* INTERRUPT-PROGRAMM : FCFS-STRATEGIE    */

DCL 1 MESSGR EXT,                  /* MESSGROESSEN                        */
      2 ANZ_AUFTR BIN FIXED(31),   /* ANZAHL DER E/A-AUFTRAEGE            */
      2 ANZ_BEARB BIN FIXED(31),   /* ANZAHL DER BEARBEITUNGEN            */
      2 S_VERWZ   BIN FLOAT,       /* SUMME DER AUFTR.-VERWEILZEITEN      */
      2 S_SYSTZV  BIN FLOAT,       /* 1.SUMME DER SYSTEMBEARB.-ZEITEN     */
      2 S_SYSTZN  BIN FLOAT,       /* 2.SUMME DER SYSTEMBEARB.-ZEITEN     */
      2 S_ZGRFZ   BIN FLOAT;       /* SUMME DER ZUGRIFFSZEITEN            */
DCL 1 REG_SATZ BASED(P_REG_SATZ), /* REGISTERSATZ FUER MP_2              */
      2 PROG_NAME BIN FIXED(31),
      2 PROG_BER  POINTER,
      2 PROG_PRIO BIN FIXED(15),
      2 MODUS,
        3 AKTION    BIN FIXED(15),
        3 IT_SPERRE BIN FIXED(15),
        3 IT_MASKE  BIN FIXED(15);
DCL 1 WS_HEAD STATIC,              /* WARTESCHL.(WS)-KOPF                 */
      2 FIRST POINTER,             /* ERSTES ELEMENT                      */
      2 LAST  POINTER;             /* LETZTES ELEMENT                     */
DCL 1 WS_EL BASED(P_WS_EL),        /* WARTESCHL.(WS)-ELEMENT              */
      2 NEXT POINTER,              /* NAECHSTES WS_EL                     */
      2 INF  POINTER;              /* VERWEIS AUF INHALT                  */
DCL 1 AUFTRAG BASED(P_AUFTRAG), /* E/A-AUFTRAG                            */
      2 SKTNR BIN FIXED(15),       /* NUMMER DES ZU TRANSF. SEKTORS       */
      2 ANM   BIN FIXED(31),       /* ZEITPUNKT DER AUFTRAGSANMELDUNG     */
      2 AUSFB BIN FIXED(31),       /* AUFTRAGSANKUNFT TROMMEL             */
      2 ZUGRB BIN FIXED(31);       /* BEGINN DES ZUGRIFFS AUF TROMMEL     */
DCL M(4) LABEL,
    (P_REG_SATZ,P_WS_EL,P_AUFTRAG) POINTER STATIC;

$0      GOTO M($$IT_MOT);          /* VERTEILER ENTSPR. INTERRUPT         */

        M(1):                      /* PROGRAMMENDE MP_2                   */
        $$STOP;

        M(2):                      /* INITIALISIERUNG                     */
$0      ALLOCATE REG_SATZ;         /* ALLOKATION VON MP_2                 */
        PROG_NAME=2;
        PROG_BER=NULL;
        PROG_PRIO=10;
        AKTION=1;
        IT_SPERRE=1;
        IT_MASKE=0;

        FIRST,LAST=NULL;
        $$RESUME(P_REG_SATZ);      /* AKTIVIERUNG VON MP_2                */

        M(3):                      /* WS-AUFBAU NACH SVC VON MP_2         */
$10     P_AUFTRAG=$$IT_ADR;        /* PARAMETERUEBERGABE DES SVC          */

    ANZ_AUFTR=ANZ_AUFTR+1;
    ANM=$$CLOCK;

        ALLOCATE WS_EL;            /* AUFBAU DES WS_EL                    */
        INF=P_AUFTRAG;
        NEXT=NULL;
```

```
$3      IF FIRST=NULL
                THEN DO;
            $6      FIRST=P_WS_EL;
                    LAST=P_WS_EL;
                    GOTO SEND;          /* SOFORT AUFTRAG AUSFUEHREN        */
                    END;
$15     LAST->NEXT=P_WS_EL;
        LAST=P_WS_EL;

    S_SYSTZV=S_SYSTZV+$$CPUTME;

        $$RETURN;                       /* WEITER MP_2                      */

        M(4):                           /* INTERRUPT VON TROMMEL: E/A-ENDE */
$2      P_AUFTRAG=$$IT_ADR;

    ANZ_BEARB=ANZ_BEARB+1;
    S_VERWZ=S_VERWZ+$$CLOCK-ANM;
    S_ZGRFZ=S_ZGRFZ+ZUGRB-AUSFB;

$8      FREE AUFTRAG;                   /* ELEMENT AUS WS AUSTRAGEN         */
        P_WS_EL=FIRST;
        FIRST=NEXT;
        FREE WS_EL;

$3      IF FIRST=NULL THEN GOTO RET; /* KEIN AUFTRAG LIEGT VOR              */
$4      P_WS_EL=FIRST;               /* NAECHSTER AUFTRAG AUS WS            */
        P_AUFTRAG=INF;

        SEND:                           /* ABGABE DES AUFTRAGS AN TROMMEL  */
$10     $$SIGNAL(2,3,1,P_AUFTRAG);

        RET:
$10
    S_SYSTZN=S_SYSTZN+$$CPUTME;

        $$RETURN;                       /* WEITER MP_2                      */

$0      END MP_1;
```

```
MP2: PROC;                          /* AUFTRAG ABGEBENDES PROGRAMM        */

DCL 1 AUFTRAG BASED(P_AUFTRAG),     /* E/A-AUFTRAG                        */
      2 SKTNR BIN FIXED(15),        /* NUMMER DES ZU TRANSF. SEKTORS      */
      2 ANM   BIN FIXED(31),        /* ZEITPUNKT DER ANMELDUNG            */
      2 AUSFB BIN FIXED(31),        /* AUFTRAGSANKUNFT TROMMEL            */
      2 ZUGRB BIN FIXED(31);        /* BEGINN DES ZUGRIFFS AUF SEKTOR     */
DCL P_AUFTRAG POINTER STATIC,
    (PSKTW,PRUNW) POINTER EXT;

        M:
$1      $$RUN($$BINOM(PRUNW));      /* SIMULIERTE PROGRAMM-AKTIV-PHASE    */
$5      ALLOCATE AUFTRAG;
        SKTNR=$$RADIS(PSKTW);       /* FESTLEGUNG DES E/A-SEKTORS         */
$10     $$SVC(P_AUFTRAG);       /* AUFTRAGSABGABE AN INTERRUPT-PROGRAMM */
        GOTO M;

$0      END MP_2;
```

```
MP_3: PROC; /* TROMMELSIMULATION                                          */
/* ES WIRD NUR DIE ZEITSPANNE SIMULIERT, IN DER EIN E/A-AUFTRAG          */
/* ABGEARBEITET WIRD.                                                    */
/* UEBER DIE BEIDEN GROESSEN: AKTSNR ( - POSITION DES LESE/SCHREIB-      */
/* KOPFES ) UND TNEXTSK ( - ZUGEH. ZEITPUNKT ) IST DER ZULETZT           */
/* SIMULIERTE TROMMELZUSTAND FESTGEHALTEN. DER AKTUELLE ZUSTAND          */
/* BERECHNET SICH SOMIT AUS DER SEITHER VERSTRICHENEN ZEIT UND           */
/* DER SEKTOR-DURCHLAUFZEIT.                                             */

DCL 1 TROMMEL EXT,                    /* TROMMELBESCHR.                  */
      2 SKT_DLZ BIN FIXED(31),         /* DURCHLAUFZEIT DES SEKTORS       */
      2 SKT_DTT BIN FIXED(31),         /*    ''  DES DATENTEILS EINES SKT */
      2 SKT_ANZ BIN FIXED(15);         /* ANZAHL DER SEKTOREN (JE SPUR)   */
DCL 1 TR_VERW STATIC,                 /* VERWALTUNG DER TROMMEL          */
      2 AKTSNR  BIN FIXED(15),         /* AKTUELLE SEKTORNUMMER           */
      2 DIFFNR  BIN FIXED(15),
      2 DIFFT   BIN FIXED(31),
      2 TNEXTSK BIN FIXED(31);         /* ZEITPUNKT NAECHSTER SEKTOR      */
DCL 1 AUFTRAG BASED(P_AUFTRAG),       /* E/A-AUFTRAG                     */
      2 SKTNR BIN FIXED(15),           /* NR DES ZU TRANSF. SEKTORS       */
      2 ANM   BIN FIXED(31),           /* ZEITPUNKT DER AUFTR.-ANMELDUNG  */
      2 AUSFB BIN FIXED(31),           /* AUFTRAGSANKUNFT TROMMEL         */
      2 ZUGRB BIN FIXED(31);           /* BEGINN DES ZUGRIFFS AUF SEKTOR  */
DCL M(2:3) LABEL,
    (PTR,P_AUFTRAG) POINTER STATIC;

$0      GOTO M($$IT_MOT);             /* VERTEILER ENTSPR. INTERRUPT     */

        M(2):                         /* INITIALISIERUNG                 */
        TNEXTSK=0;
        AKTSNR=0;
        $$IDLE;

        M(3):                         /* E/A TRANSFER                    */
$0      /* BERECHNUNG VON TNEXTSK UND AKTSNR                             */
        P_AUFTRAG=$$IT_ADR;           /* UEBERNAHME DES AUFTRAGES        */

    AUSFB=$$CLOCK;

        /* UPDATING VON TNEXTSK UND AKTSNR                               */
        DIFFT=$$CLOCK-TNEXTSK;
        DIFFNR=FLOOR(DIFFT/SKT_DLZ)+1;
        TNEXTSK=TNEXTSK+DIFFNR*SKT_DLZ;
        AKTSNR=AKTSNR+DIFFNR;
        IF AKTSNR>SKT_ANZ THEN AKTSNR=AKTSNR-SKT_ANZ; /* MOD.-R.     */

        /* ERMITTL. DES ZEITINTERVALLS BIS TNEXTSK                       */
        DIFFT=TNEXTSK-$$CLOCK;
        IF DIFFT>0 THEN $$RUN(DIFFT);

        SYNCHRON: /* ERMITTL. DES ZEITINTERVALLS BIS SKTNR ERREICHT      */
$0      DIFFNR=SKTNR-AKTSNR;
        IF DIFFNR<0
                  THEN DO;
                  DIFFNR=SKT_ANZ+DIFFNR;
                  GOTO W1;
                  END;
```

```
        IF DIFFNR>0
                  THEN DO;
                  W1: /* AKTUALISIEREN AUF SKTNR                          */
                  AKTSNR=SKTNR;
                  TNEXTSK=TNEXTSK+DIFFNR*SKT_DLZ;
                  $$RUN(DIFFNR*SKT_DLZ);
                  END;

        TRANSF: /* TRANSFER DES SEKTORS                                    */
$0
    ZUGRB=$$CLOCK;

$SKT_DTT /* ZEIT FUER EIN/AUS-LESEN DES DATENTEILS EINES SEKTORS         */
        TNEXTSK=TNEXTSK+SKT_DLZ;    /* TROMELZUSTAND NACH UEBERTRAGUNG*/
        AKTSNR=AKTSNR+1;
        $$SIGNAL(1,4,5,P_AUFTRAG); /* TRANSFERENDE                        */
        $$IDLE; /* SIMULATIONS-STILLSTAND BIS ZUM NAECHSTEN TRANSF.   */

$0      END MP_3;
```

```
MP_4: PROC;                              /* SIMULATIONS-MONITOR          */
DCL DAUER BIN FIXED(31),
    M(3)  LABEL,
    ANTW  CHAR(2);

        ON CONVERSION GOTO M(3);

        GOTO M($$IT_MOT);                /* VERTEILER ENTSPR. INTERRUPT */

        M(1):                            /* SIMULATIONS-ENDE             */
$0      $$STOP;

        M(2):                            /* SIMULATIONS-BEGINN           */
$0      PUT SKIP(3) EDIT('GIB SIMULATIONSDAUER:')(A);
        GET LIST(DAUER);
        $$INT(DAUER,1,1,NULL);

        M(3):                            /* TRACING INTERVALL ERMITTELN */
$0      $$TEST('E');
        PUT SKIP EDIT('TRACING? -''JA''/''NEIN'' ?:')(A);
        GET LIST(ANTW);
        IF ANTW='JA' | ANTW='J'
                  THEN DO;
                  PUT EDIT('WIELANGE?:')(A);
                  GET LIST(DAUER);
                  $$INT(DAUER,3,2,NULL);
                  $$TEST('B');
                  END;
                  ELSE DO;
                  PUT EDIT('WIELANGE NICHT?:')(A);
                  GET LIST(DAUER);
                  IF DAUER<0 THEN GOTO M(1);
                  $$INT(DAUER,3,2,NULL);
                  END;
        $$IDLE;

$0      END MP_4;
```