ASIM
München '87

Simulation eines fehlertoleranten Multi-Mikroprozessorsystems unter UNIX™

R.Brause, Universität Frankfurt

Abstrakt:

Es wird die Architektur und Simulation des fehlertoleranten Multi-Mikroprozessorsystems ATTEMPTO unter UNIX™ beschrieben. Insbesondere wird auf die Abbildung der Soft- und Hardware-Charakteristika des Multi-Mikroprozessorsystems auf die unter UNIX gegebenen System-Möglichkeiten eingegangen sowie die Maßnahmen zum Testen, Validieren und Monitoring einer derart komplexen Software beschrieben.

1) Das ATTEMPTO System

ATTEMPTO ist ein experimenteller, fehlertoleranter Arbeitsplatzrechner, der in Tübingen entwickelt wurde /FTCS/.
Ziel der Bemühungen ist es, trotz eventuell auftretender Ausfälle und Störungen eine korrekte Ergebnisse des Computersystems zu erreichen. Diese Fehlertoleranz läßt sich durch Zeitredundanz (Rollback etc.) oder Hardwareredundanz (Vervielfachung von Hardware und Maskierung bzw. Voting der Ergebnisse) verwirklichen. Wir entschieden uns für Hardware-Redundanz auf Ebene von Single-Board-Computern (SBC), da damit sowohl fehlertolerante Ausführung eines Programms mit vielen Prozessoren als auch schnelle, parallele und nicht-fehlertolerante Ausführung mehrerer Programme möglich ist.
Der Benutzer sieht den Arbeitsplatzrechner als ein Single-User, Multi-Tasking System; die Realisierung als Multi-Computersystem ist ihm verborgen. Das System bietet dem Benutzer die Möglichkeit, die Fehlertoleranzeigenschaften individuell für jede Anwendung zu wählen.
Es ist das Auftreten von vorwiegend transienten, aber auch permanenten Defekten auf den SBC vorgesehen. Als kleinste ersetzbare Einheit ist damit ein SBC gewählt. Wird der Input parallel an alle SBC herangeführt, so kann der Ausfall eines SBC nicht zur Verfälschung des Inputs für andere führen; falsche Ergebnisse eines SBC sind nur auf eigene Fehlfunktionen zurückzuführen.
Parallel zu einer Fehlermaskierung wird auch die Diagnose auftretender Fehler vorgenommen. Um eine Fehlermaskierung zur fehlertoleranten Abarbeitung eines Benutzerjobs zu erreichen, wird der gleiche Job von mehreren SBC abgarbeitet. Kommt es zur Ausgabe von Ergebnissen an den Benutzer, so werden die Ergebnisse vorher zwischen den SBC verglichen und das von der Diagnose als korrekt diagnostizierte Ergebnis wird von einem als intakt diagnostizierten SBC ausgegeben.
Aus der Entscheidung, Fehlererkennung und Diagnose durch lose gekoppelte Multiprozessorsysteme zu realisieren, folgt die Notwendigkeit einer Kommunikation zur Koordinierung der Multiprozessorsysteme. Da keine zentrale, fehlersensible Systemtafel gewünscht wird, müssen die lokalen Systemtafeln der SBC für das Management globaler Resourcen (Zuteilung des Terminal etc.) durch Kommunikation widerspruchsfrei gehalten werden. Eine Möglichkeit dazu besteht darin, die Bearbeitungsreihenfolge der Nachrichten, die sich auf die Systemtafeln auswirken, auf allen SBC gleich zu halten. Diese Forderung muß dann sowohl vom Inter-Prozessor-Kommunikationssystem als auch vom Betriebssystem auf den SBC erfüllt werden /BRA/.
Die Hardwarestruktur des Systems besteht aus einer Zahl von handelsüblichen Single-Board-Computern mit dual-port memory, die mit

einem multi-master-bus zur Inter-Prozessor-Kommunikation miteinander verbunden sind (Abb.1). Das Lesen der Nachrichten in den Briefkästen des dual-port memory wird durch dedizierte Interrupts (einer pro SBC) ausgelöst. Da auf jedem SBC für jeden SBC je ein Briefkasten (port) vorhanden ist, besteht zwischen je zwei SBC ein separater Kommunikationskanal.

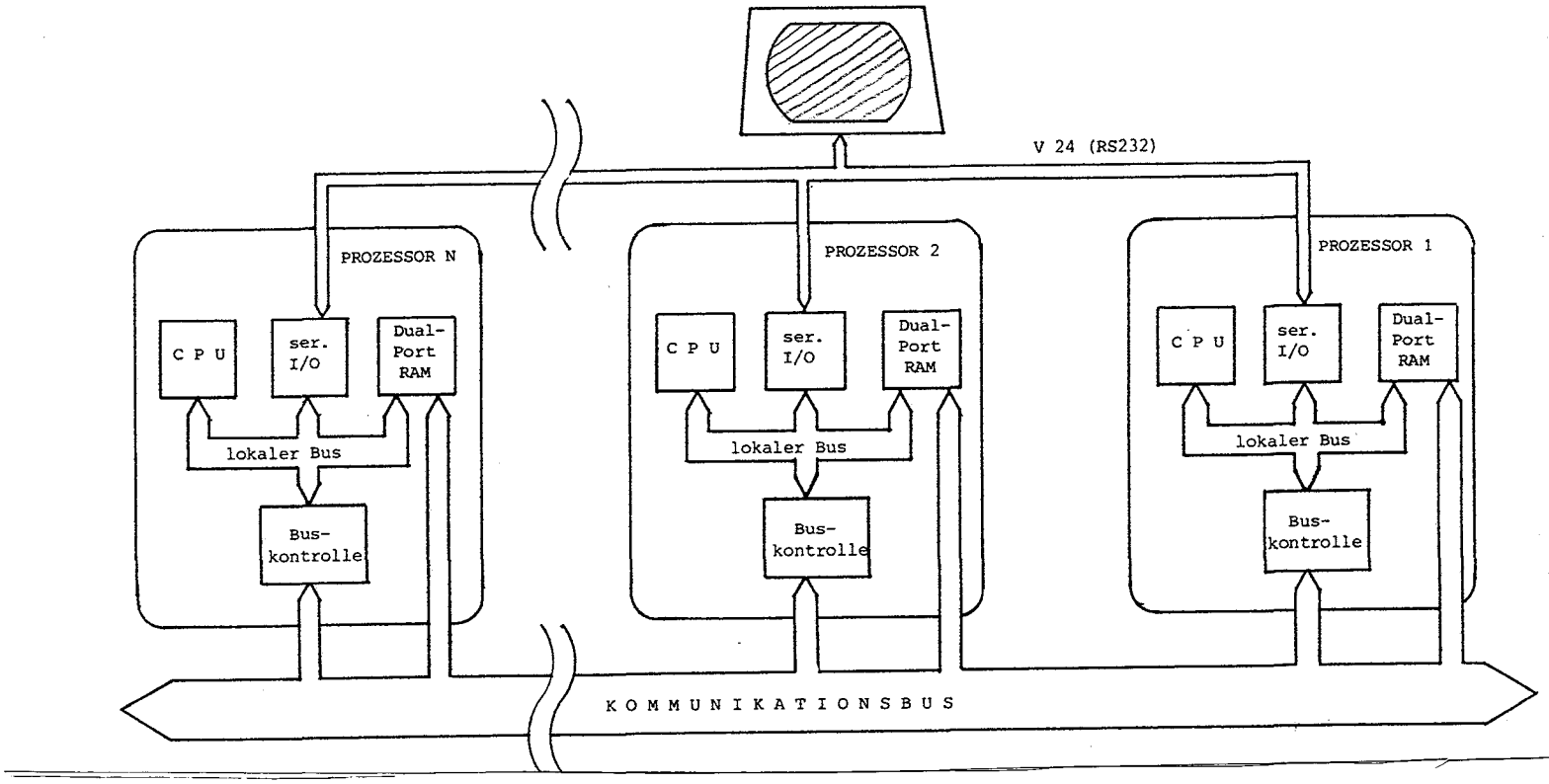


Abb.1 Hardwarekonfiguration von ATTEMPTO

Die Systemsoftware ist dezentralisiert; jeder SBC besitzt ein eigenes, lokales Betriebssystem ATOS /BRA/. Den Kern von ATOS bildet ein UNIX-ähnliches Ein-Prozessor multi-tasking Betriebssystem. Darauf baut eine Schicht von Systemfunktionen (FTL) auf, welche die Fehlertoleranzeigenschaften bereitstellt. Diese Schicht wird über Nachrichten aktiviert. Betriebssystemabhängiger Nachrichtenempfänger und -sender der FTL ist die Kommunikationsinstanz CI, die auch die Kommunikation zum User-Job erledigt. Davon abgesetzt ist der portable Kern der FTL, die Fehlertoleranzinstanz FTI. Um den Benutzerjob zu Fehlertoleranzzwecken beobachten zu können, werden bestimmte UNIX-Systemaufrufe (SysCalls) des Benutzerjobs nicht direkt ausgeführt, sondern erst der FTL zugestellt, die dann das Nötige veranlasst(Abb.2). Insbesondere sind dies folgende SysCalls:

- open() einer globalen Resource (z.B. Terminal)
- read() und write() für globale Resourcen
- fork() um die Prozeßnummer der Kinder für read/write zu gewinnen
- exit() um Kenntnis von der Terminierung des Benutzerjobs und seiner Kinder zu erlangen

Die FTI selbst ist eine Menge von Modula-2 Prozessen, die durch Nachrichten aktiviert werden(Abb.3a). Die Aufteilung der Aufgaben unter den Modula2-Prozessen ist nach dem Objekt-Konzept gestaltet worden: jeder Prozeß verwaltet exklusiv eine Datenstruktur. Damit ist Lesen und Schreiben in den Daten nur durch Nachrichtenaustausch möglich, der

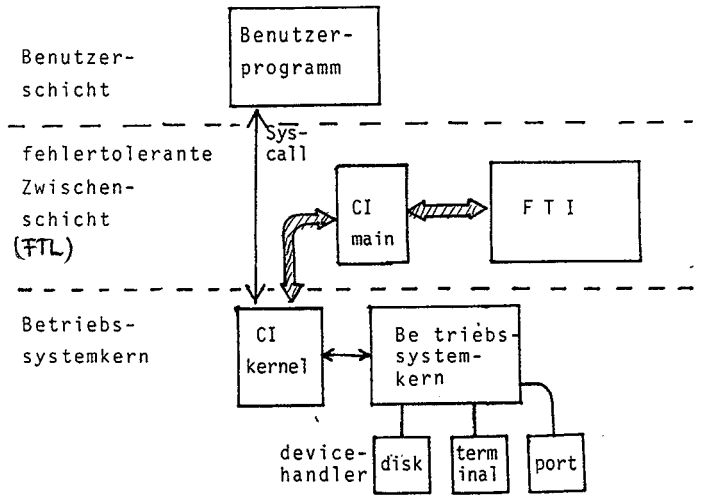


Abb.2 Schichtenmodell

leicht kontrolliert werden kann (s.Abschnitt 3).

Die Nachrichten an die FTL stammen vom

- Terminal (Benutzer-Eingabe)
- Devices (Magnetplatte etc.)
- über den Kommunikations-port von anderen FTL anderer SBC
- Betriebssystemkern, ausgelöst durch eine system call des Benutzerjobs

Eine Möglichkeit, die Nachrichten zu puffern sowie ihre zeitliche Reihenfolge zu erhalten, bietet das pipe-Konstrukt in UNIX /BELL/. Das Schreiben in eine pipe ist atomar, d.h. es kann nicht von einem Prozeß-Wechsel unterbrochen werden. Da der aufrufende Prozeß bei einer leeren pipe automatisch die Prozessorkontrolle weitergibt, wird für jedes Peripheriegerät, auf deren Daten gewartet werden muß, ein Stellvertreterprozeß erzeugt. Dieser hat die Aufgabe, von der Peripherie eintreffende Daten in ein einheitliches Nachrichtenformat zu transferieren und sie in die pipe zur FTL zu schreiben. Damit kann die FTL die Daten der verschiedenen Nachrichtenquellen separieren und verarbeiten. In Abbildung 3b ist die UNIX-Prozeßstruktur gezeigt. Die UNIX-Prozesse und die pipes zur Kommunikation werden bis auf die shell SH, die den Benutzerjob UJ hervorbringt, alle beim Starten des Systems (booten) erzeugt. Die shell bzw. Benutzerjob werden immer dann von der FTL erzeugt, wenn der Benutzer einen neuen Auftrag gegeben hat und durch Kommunikation mit den anderen SBC abgestimmt wurde, daß gerade diese SBC den Job übernimmt (Prinzip der Auftragsanziehung). Näheres zur Synchronisation und Kommunikation ist in /BRA/ beschrieben.

2) Die Simulation des ATTEMPTO Systems

Die Simulation hat die Aufgabe, die vorgegebe Hard- und Softwarestruktur zum Testen und Validieren der logischen Grundfunktionen (Interprozessor-Kommunikation und Koordination) sowie der Fehlertoleranzfunktionen nachzubilden.
Dies bedeutet u.a.

- Die Prozeßstruktur des simulierten Systems sollte Untermenge der Simulation sein

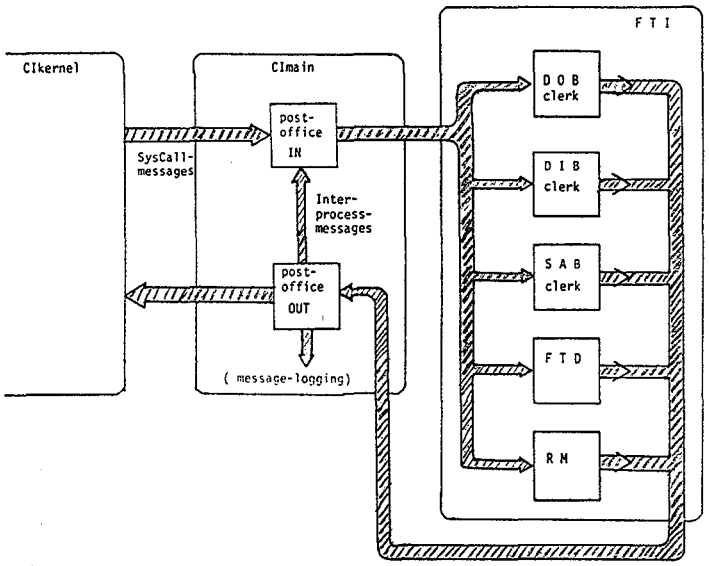
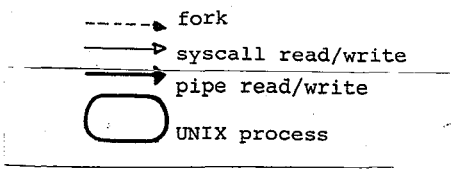


Abb. 3a Modula-2 Prozesse



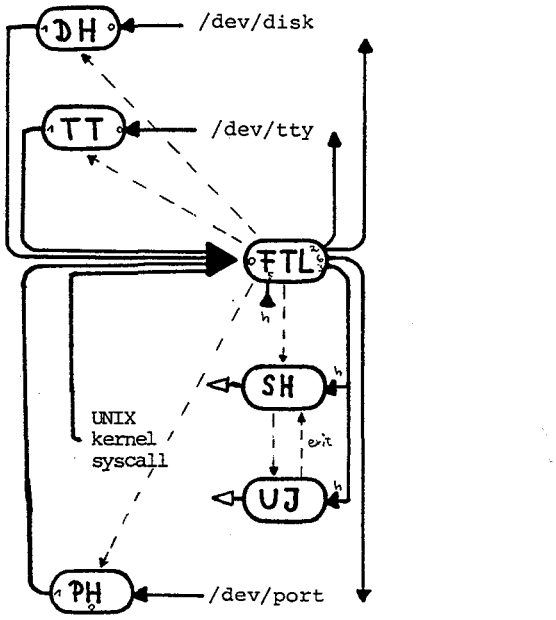


Abb.3b UNIX-Prozeßkonfiguration

- Als Grundmechanismen der Kommunikation sollten die Hardware-Mechanismen des Zielrechners logisch emuliert werden.

Betrachten wir nun die Simulation für drei SBC in Abbildung 4. Zum Aufbau des Simulationsrahmens wird ein Initialisierungsprozeß (gestrichelte Linien) gestartet. Dieser erzeugt alle FTL-Prozesse sowie einen Supervisor-Prozeß (SV). Der SV verfügt über ein System überlappender Windows, die es gestatten, sowohl die Ausgabe auf das simulierte Terminal als auch Kontrollnachrichten einelner FTL übersichtlich getrennt darzustellen. Außerdem wird damit ermöglicht, Benutzereingaben (TTY) an alle SBC oder für Kontroll- und Monitorzwecke an einzelne, selektierte SBC zu geben. Die Fehlertoleranzschicht FTL sowie die Prozesse TT, PH, SH und UJ sind pro SBC einmal vorhanden und mit den gleichen Kommunikations-pipes versehen wie in ATTEMPTO (Abbildung 3b).
Für die Simulation wurden die Anforderungen an den Benutzerjob zunächst geringfügig modifiziert: Der Job darf kein eigenes Terminal öffnen und keine Kinder erzeugen. Mit dieser Einschränkung ist es nicht mehr nötig, die SysCalls des Benutzerjobs UJ zu überwachen. Stattdessen werden der shell (und damit auch dem UJ) pipes für Standardinput und Standardoutput zur Verfügung gestellt. Beim exit()-SysCall des UJ wird automatisch der shell die Kontrolle übergeben, die mit einer speziellen Ausgabe vor ihrem exit() dem SC Prozeß das Ende des Benutzerjobs anzeigt. Dies wird von SC in eine entsprechende Nachricht an die FTL umgesetzt.
Für die Interprozessor-Kommunikation werden die Kommunikationskanäle durch pipes nachgebildet und die Sendeseite der Porthandler durch einen eigenständigen UNIX-Prozeß PH. Die Interrupts, die auf den SBC das Auslesen der ports bewirken, lassen sich durch Signale des UNIX-Systems ersetzen. Dabei wird von der Möglichkeit des UNIX-Systems Gebrauch gemacht, ein Signal an alle Prozesse eines Terminals zu senden (Broadcast). Um die nicht beteiligten Prozesse damit nicht abzubrechen, werden alle Prozesse (bis auf die PH) bei der Erzeugung unempfindlich gegenüber diesen Signalen gesetzt.

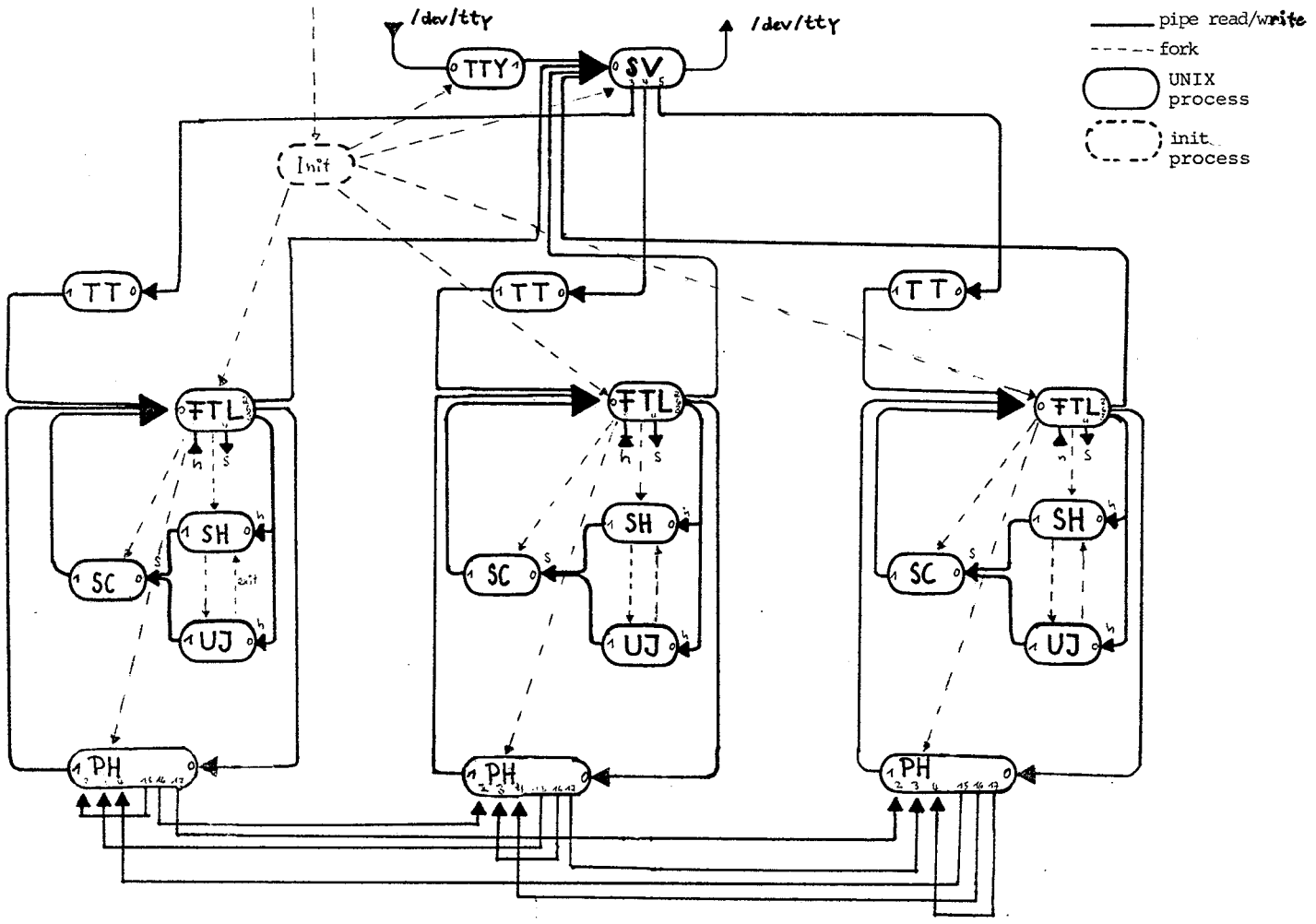


Abb.4 Simulation der UNIX-Prozeßkonfiguration

3) Testhilfen und Debugging der Module

Wie läßt sich ein solches komplexes Software-Paket effektiv testen?
Da es sich bei dieser Simulation um ein begrenzt zeitkritisches System (nur time-out der Kommunikation) handelt, lassen sich die Systemteile relativ gut als Einzelmodule testen. Die zu testende Software-Struktur ist sowohl durch die verwendete Sprache (Modula-2 /WIR/) als auch durch die gewählte Systemstruktur (UNIX und Modula-2 Prozesse) stark modularisiert.
Zuerst wurde das Programm zum Erzeugen des Simulationsrahmens erstellt und es wurde ausgetestet, ob auch alle Prozesse und Pipes mit den richtigen, vorher vereinbarten file-Deskriptoren erzeugt wurden. Dann wurde der Code der UNIX-Prozesse, die danach den Prozessen im Simulationsrahmen überlagert werden, separat getestet. Das gewünschte Input-Output Verhalten der Prozesse TTY, TT, SC, SV, SH/UJ kann leicht durch Testnachrichten überprüft werden. Dieser Testinput wird vorher auf files geschrieben; bei der Test-Aktivierung des Prozesses wird von der Möglichkeit der UNIX shell Gebrauch gemacht, den Testinput auf den Standard Inputkanal 0 und den Testoutput auf einen log-file umzulenken.
Da die Fehlertoleranzschicht FTL als einzelner UNIX-Prozeß in mehrere Modula-2 Prozesse unterteilt ist, die über Nachrichten und Briefkästen miteinander kommunizieren, kann durch kontrollierte Eingabe von

Testnachrichten (Testfiles) sowie durch Protokollieren aller durch das post-office laufender Nachrichten (s.Abb.3a) genau festgestellt werden, welcher Modula-2 Prozeß falsches Input-Output im Zusammenspiel mit den anderen liefert.
Das Prozeßmanagement, welches das von Modula-2 zur Verfügung gestellte Koroutinen-Konzept benutzt, behandelt die auf unserem Rechner (PDP11/24) nötigen Overlays transparent für die Prozesse und konnte somit getrennt ausgetestet werden.
Um die Überprüfung der Datenbasis der FTL und des Laufzeitverhaltens der Modula-2 Prozesse auf Source-Code-Level zu erreichen, wurde der Post-Mortem Debugger von Modula-2 /MAI/ für UNIX leicht modifiziert. Anstatt den zu überwachenden UNIX-Prozess direkt zu starten, wird zuerst der Debugger aufgesetzt, der wiederum als Kind-Prozeß den fraglichen Prozeß erzeugt. Empfängt der mit dem UNIX-SysCall "ptrace()" initialisierte Kind-Prozeß ein Signal (z.B. vom Debugger), so wird er als Prozeß "eingefroren" und die Kontrolle geht an den Debugger über. Zum eigentlichen Debuggen wird der Speicherinhalt anstelle aus dem Post-Mortem-Dump file mittels des UNIX SysCall "ptrace()" direkt von dem angehaltenen, zu überwachenden Kind-Prozeß gelesen und in Source-Code Form dargestellt. Die einzige Änderung, die den Post-Mortem-Debugger in einen Online-Debugger umwandelt, liegt in dem Ersatz der Funktion "CoreWord()", die ein Datenwort vom Post-Mortem-Dump file liest, durch den UNIX-SysCall "ptrace()".

4) Zusammenfassung und Ausblick

In dem Beitrag wurde dargelegt, wie aus den Modellierungsentscheidungen für ein fehlertolerantes Computersystem eine das Modell realisierende Multiprozessorkonfiguration resultiert.
Außerdem wurde gezeigt, wie mit Hilfe einer modularisierten Grundstruktur der Software-Konfiguration eines Computers sowie UNIX-spezifischen Systemmöglichkeiten auf relativ einfache Weise die komplexe Simulation einer Multi-Computer Konfiguration durchgeführt werden kann.
Diese Simulation ermöglicht darüber hinaus auch die Tolerierung transienter Fehler während der Simulation. Im Unterschied zum ATTEMPTO System wird dazu keine Hardware-Redundanz (multiple SBC) genutzt, sondern die Zeitredundanz der simulierten SBC-Prozesse.
Mit der Simulation ist es auch möglich, andere modellierte Diagnoseverfahren zu implementieren, sie auf Konsistenz zu testen sowie ihren Kommunikationsaufwand zu messen. Auch eine Erweiterung des ATTEMPTO-Systems zur verteilten Zusammenarbeit der SBC (pipelining etc.) ist damit leicht testbar.

Literatur

/BELL/ The UNIX Time-sharing System
The Bell System Technical Journal Vol 57/6 Part2 1978

/BRA/ R.Brause, E.Ammann, M.DalCin, E.Dilger, J.Lutz, T.Risse
Softwarekonzepte des fehlertoleranten Arbeitsplatzrechners ATTEMPTO
Proc. of ACM Microcomputing, München 1983

/FTCS/ E.Ammann, R.Brause, M.DalCin, E.Dilger, J.Lutz, T.Risse
ATTEMPTO: A fault-tolerant Multiprocessor Working Station;
Design and Concepts
IEEE Proc. of FTCS-13, Milano 1983

/MAI/ G.Maier
Modula-2 Debugger for structured Data
Institut für Automatik und Industrielle Elektronik
ETH Zürich, Switzerland

/WIR/ N. Wirth
Programming in Modula-2
Springer Verlag Berlin Heidelberg New York 1982

Anschrift des Autors:
Dr.R.Brause
Universität Frankfurt
Dantestr.9
Postfach 111932
D- 6000 Frankfurt