Ein PEARL-Testsystem zum Einsatz in verteilten Systemen*)

C. Andres, A. Fleischmann, P. Holleczek, W. Mühlbauer

0. Einleitung

Die Akzeptanz von höheren Realzeitsprachen in der Prozeßsteuerung hängt wesentlich von den Testmöglichkeiten ab. Dieser Tatsache wurde auch bei PEARL bereits vielfach Rechnung getragen /GMH079/. Der Einsatz von PEARL in verteilten Systemen und der Programmtest auf Mikroprozessoren ohne eigene Bedienperipherie bedarf neuer Ansätze. Im vorliegenden Artikel soll ein PEARL-Testsystem vorgestellt werden, das den Test von "physikalisch" verteilten Programmen erlaubt. Es berücksichtigt darüber hinaus Botschaftsmechanismen, wie sie zur Programmierung verteilter Systeme vorgeschlagen wurden /FLH082/.

1. Eigenschaften des Testsystems

Bei Crosscompilern, wie sie oft für Mikroprozessoren verwendet werden, scheidet eine Fehlerlokalisierung durch Programmänderungen aufgrund der langen Programmerstellungszeiten praktisch aus. Deshalb wurde ein sprachorientiertes Testsystem entwickelt, das keine Neuübersetzung von Programmen verlangt,so daß wertvolle Zeit eingespart wird.
Man bediente sich dabei der dynamischen Instrumentierung /GMH079/, bei der ggf. (d.h. bei Traces und Haltepunkten) im bereits geladenen Programm der ursprüngliche Code durch Test-Code ersetzt wird. Das bedingt, daß zur Laufzeit noch verschiedene vom Übersetzer erzeugte Listen vorhanden sein müssen.

Das Testsystem besitzt einen modularen Aufbau und ist sowohl für Ein- wie Mehrprozessor-Konfigurationen einsetzbar.

2. Leistungsumfang des Testsystems

Neben den herkömmlichen Auskünften über Prozesse und Variable liefert das Testsystem Informationen über Betriebssystem-Warteschlangen sowie über den Inhalt der Botschaftspuffer.
An Manipulationen sind die üblichen Operationen auf PEARL-Datentypen sowie das Erzeugen von Botschaften erlaubt.

Das Testsystem liefert folgende Informationen:

(1) Auskunft über den Status aller Prozesse.
Ausgabe: -Taskpriorität
-Taskzustand

(2) Auskunft über Variable und Arrays.
Ausgabe: -Speicherinhalt

(3) Auskunft über Semaphore.
Ausgabe: -aktueller Wert
-Semaphore-Warteschlange
(ausgegeben werden die an dem Semaphore blockierten Tasks)

(4) Auskunft über den Inhalt der Botschaftspuffer.
Ausgabe: -Inhalt des Warte- und des Blockiertbereichs der Task
-Anzahl der Botschaften im Wartebereich der Task

(5) Auskunft über Geräte

*) gefördert von der DFG im Rahmen des Vorhabens Ho824

Ausgabe: -belegende Task
-Geräte-Warteschlange (ausgegeben werden die auf die Zuteilung wartenden Tasks)

(6) Auskunft über Interrupts
Ausgabe: -Interrupt-Warteschlange (ausgegeben werden diejenigen Tasks, die auf das Eintreffen des Interrupts warten)

(7) Auskunft über die Prozessor-Warteschlangen
Ausgabe: -Prozessor-Warteschlange (ausgegeben werden die Tasks, die auf die Zuteilung des Prozessors warten)

Dabei ist zu berücksichtigen, daß die Auskünfte überholt sein können, wenn die Prozesse nicht angehalten werden.

Das Testsystem läßt folgende Manipulationen zu:

(1) Zustandsänderungen von einzelnen Prozessen
- ACTIVATE
- CONTINUE
- PREVENT
- SUSPEND
- TERMINATE

(2) Anhalten und Fortsetzen einzelner Prozesse
- Haltepunkt setzen
- Haltepunkt löschen
- Fortsetzen des angehaltenen Prozesses

(3) Anhalten und Fortsetzen des Gesamtsystems

(4) Erzeugen von Botschaften

(5) Ändern von Semaphore-Werten

(6) Ändern der Werte von Variablen und Arrays

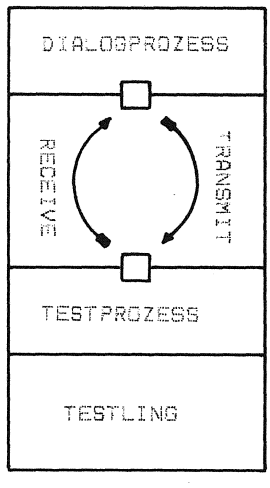
(7) Ein- und Ausschalten von Trace
- Setzen und Löschen von Traceberei-
chen

(8) Trigger-Interrupt

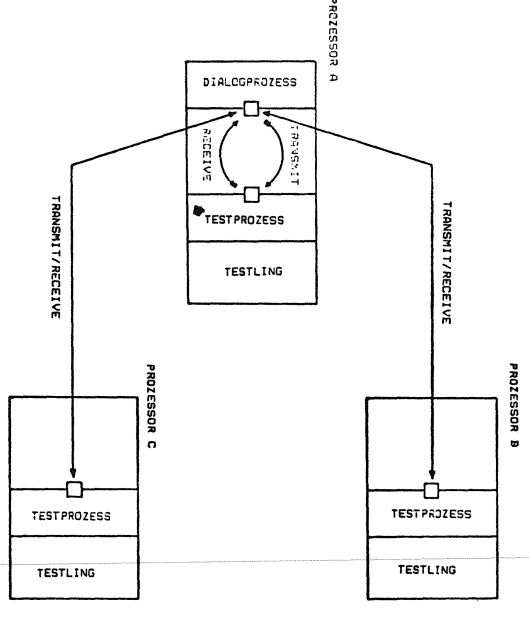
Dabei ist bei manchen Manipulationen zu beachten, daß sie zwar gefährliche Seiteneffekte haben können, aber durch Überwinden von Stillständen wertvolle Testzeit sparen helfen.

3. Aufbau des Testsystems

Die Funktionsweise des Testsystems wurde so spezifiziert, daß es sowohl auf Ein- wie Mehr-Rechnerkonfigurationen läuft (B i l d 1, B i l d 2).



B i l d 1: Testsystem in Monoprozessorkonfiguration



B i l d 2: Testsystem in Mehrprozessorkonfiguration

Da es ein Testsystem für Anwender-Programme darstellt, kann es die Funktionsfähigkeit des Betriebssystems voraussetzen und Funktionen des Betriebssystems mitbenutzen (soweit Seiteneffekte ausgeschlossen sind).
Das Testsystem wird immer in Form von zwei miteinander kommunizierenden Prozessen, einem Dialog-Prozeß und einem Test-Prozeß realisiert.
Der Dialog-Prozeß läuft immer auf dem zentralen Prozessor (Mittelpunkt eines logischen Sterns) des verteilten Systems, der mit einem Bedienterminal ausgerüstet ist.
Der Test-Prozeß läuft stets auf dem Prozessor, der das zu testende Programm beherbergt.
Diese Konfiguration unterstützt Mikroprozessorsysteme, deren Frontend-Prozessoren üblicherweise über keine Bedienperipherie verfügen.
Der Dialog-Prozeß, bestehend aus Kommandointerpreter und Protokollaufbereitung, interpretiert die vom Anwender abgegebenen Testaufträge (Kommandointerpreter) und komprimiert sie in Form von Botschaften, die dem Test-Prozeß übergeben werden. Der Test-Prozeß führt die Testaufträge durch und gibt die Ergebnisse oder Bestätigungen in Form von Botschaften an den Dialog-Prozeß zurück. Der Dialog-Prozeß entkomprimiert die Ergebnisse (Protokollaufbereitung) und protokolliert sie auf dem Bedien-Terminal.

4. Implementation

Wie schon in Kapitel 2 erwähnt, wird der Programmcode des Testlings nicht expandiert (statische Instrumentierung), sondern zur Laufzeit dynamisch ersetzt (dynamische Instrumentierung).

Vorteile der dynamischen Instrumentierung:

(1) Zum Testen ist kein eigener Übersetzungslauf notwendig, was vor allem bei Cross-Übersetzungssystemen viel Zeit spart und sehr zur Benutzerfreundlichkeit des Testsystems beiträgt.

(2) Das Echtzeitverhalten des Testlings zwischen Haltepunkten wird nicht beeinflußt, weil in diesen Bereichen der ursprüngliche Benutzercode - durchlaufen wird.

(3) Die vom Testsystem benötigten Informationen (in Form von Listen) können auf Externspeicher ausgelagert, und bei Bedarf als Teillisten wieder eingelagert werden.
Dies ist besonders wertvoll bei Mikroprozessorsystemen mit eingeschränktem Arbeitsspeicherausbau. Statische Expansion würde an dieser Stelle ein Overlay-Binden erforderlich machen.

Einen Nachteil hat diese Methode:

Derjenige Teil des Testsystems, der den Codeaustausch an den Testpunkten durchführt, ist aufgrund der unterschiedlichen Befehlsformate stark rechnerabhängig. Deshalb muß dieser Teil bei einer Übertragung auf andere Rechner neu codiert werden (allerdings nicht neu spezifiziert).

Im folgenden sollen die vom Testsystem zur Laufzeit benötigten Listen beschrieben werden.

(1) Modulliste
Zur Realisierung von Haltepunkten und Traces muß es möglich sein
- bei gegebenem Modulnamen und gegebener Zeilennummer die zugehörige Adresse und
- bei gegebener Adresse die zugehörige Zeilennummer und den Modulnamen zu ermitteln.

(2) Taskliste
Mit Hilfe der Taskliste ist es möglich
- bei gegebenem Tasknamen die zugehörige Kontrollblockadresse und den Prozessor, auf dem sich diese Task befindet
- bei gegebener Kontrollblockadresse den zugehörigen Tasknamen zu ermitteln.

(3) Semaphoreliste

(4) Geräteliste
Die Semaphore- und Geräteliste sind analog aufgebaut, so daß man bei gegebenem Semaphore- bzw. Gerätenamen die zugehörige Kontrollblockadresse ermitteln kann.

(5) Variablenliste
Die Variablenliste enthält neben den Variablennamen und den zugehörigen Adressen noch Angaben über den Datentyp und die Datentyplänge, um eine typgerechte Ein- bzw. Ausgabe zu ermöglichen.

(6) Arrayliste
Die Arrayliste besitzt denselben Aufbau wie die Variablenliste, erlaubt aber zusätzlich noch eine Überprüfung von Indexgrenzen.

Der Dialogteil des Testsystems wurde in PEARL geschrieben, um einen gewissen Grad an Portabilität zu erreichen.
Die Implementation der Testfunktionen erfolgte weitestgehend im gleichen abstrakten Assembler ("SPASS") wie die des PEARL-Betriebssystems, so daß mit Installation des Betriebssystems auch die Testfunktionen auf einem anderen Zielrechner vorhanden sind.

Einige wenige Testfunktionen und die Listenverwaltung (für die oben beschriebenen Listen) wurden aus Effizienzgründen in Assembler programmiert.

5. Zusammenfassung

Entsprechend den geringen Erfahrungen im Umgang mit verteilten Systemen bilden die entwickelten Testfunktionen lediglich eine Basis für eine Weiterentwicklung des Testsystems.
Der modulare Aufbau des Testsystems ermöglicht es, nachträglich Funktionen hinzuzufügen, die sich im praktischen Einsatz als notwendig erweisen, bzw. überflüssige oder "gefährliche" Funktionen wegzulassen.

Literaturhinweise

/GMH079/ L. Gmeiner, G. Hommel (Hrsg.): Testen und Verifizieren von Prozeßrechner-Software, KfK-PDV 179, Dez.1979, KfK, Karlsruhe

/FLH082/ A. Fleischmann, P. Holleczek, G. Klebes, R. Kummer: Ein Mechanismus zur Kommunikation für verteilte Systeme in PEARL, in diesem Heft