Sitzungsnr.: 14

Termin : 30.6.78 9^{00} - 12^{30}

Teilnehme : Andreas, Hans P., Jens, Volker

*Thema : PATH-Expressions und Speichervergabe

Inhalt :

Bezugnehmend auf den Artikel von Habermann (→ LITDOC) stellten wir Überlegungen an, wie man durch zusätzliche Angabe eines PATH's im MONITOR algorithmisch Informationen über

a) die Aufrufreihenfolge der ENTRY-Prozeduren

b) den Speicherbedarf der Modulhülle

und damit zur

c) Deadlockerkennung und ~ Vermeidung

zu gewinnen (bei Modulhüllen) kann.

1) Ein Buffer würde z.B. so aussehen:

```
Buffer = MONITOR
        -- VAR-Teil ...
        (* PATH Write ; Read END *)
        PROCEDURE ENTRY Read;
        !... DELAY... CONTINUE...
        PROCEDURE ENTRY Write;
        !-- DELAY... CONTINUE...
        END;
```

Man kann dem PATH also entnehmen, daß zuerst geschrieben und dann abwechselnd geschrieben und gelesen wird. Das würde bedeuten, der MANAGER muß dem Buffer nur das Maximum des Speicherbedarfes von "Read" und "Write" zuteilen, da sie nacheinander ausgeführt werden.

*Literatur : Habermann "Path Expressions"

2) Ein anderes Beispiel:

... (*PATH Write; (Read + Write)* *) ...

Hier wären die Folgen

Write Read Read Read...

oder Write Write Write... möglich, man kann damit also keine Speicherersparnis gewinnen.

3) Man betrachte eine Platte, deren Benutzung erst angemeldet, dann initialisiert, dann terminiert und abgemeldet werden muß:

.. (*PATH A; B; C; D END *) ..

Da die Prozeduren durchaus von verschiedenen Prozessen aufgerufen werden können, ist es nicht möglich, hier Speicherplatz zu sparen, noch zu kontrollieren, ob ein bestimmter Prozess sich auch an die vorgeschriebene Reihenfolge hält.

Ein Vorschlag wäre, 2 PATH's anzugeben

a) COREPATH A | B | C | D END

nur für den Speicherplatz, und

b) CALLPATH A; B; C; D END

nur für die vorgeschriebene Reihenfolge.

Da dies das Problem verkompliziert, wurde beschlossen, nur den Fall b) vorerst weiterzuverfolgen. Wir gehen also davon aus, daß der PATH uns, unabhängig von der Identifikation des rufenden Prozesses, aber nicht unabhängig von der Art (ob Stellvertreterprozess oder Rechner-interner Prozess), Angaben macht über das zeitliche Verhalten der MONITOR-Prozeduren zueinander.

Betrachten wir also folgenden PATH:

PATH A; B; C; D END

Wenn A, B, C, D nur von Stellvertretern aufgerufen werden, genügt es offensichtlich, als Speicherbedarf das max(|A|, |B|, |C|, |D|) zu wählen, da eine gleichzeitige Bearbeitung wegen der Aufrufreihenfolge verboten ist. Sollte z.B. auch B vor A aufgerufen werden, so muß der rufende Prozess (Stellvertreter) im Manager verzögert werden.

Im Allgemeinen wird aber bei einem Ausdruck

PATH A_1; A_2 ... A_n END

es mindestens ein A_j geben, das sowohl von Stellvertretern als auch von internen Prozessen aufgerufen wird. Als Nomenklatur wollen wir "A_j^D" als eine solche Prozedur verstehen.

Ebenso gibt es i.A. Prozeduren, die nur von internen Prozessen aufgerufen werden. Da diesen vom Compile automatisch Speicherplatz alloziert wird, sind sie für uns nicht von Interesse. Im Folgenden nehmen wir an ein vollständiger PATH wäre um diejenigen Prozeduren verkürzt, die nur von internen Prozessen aufgerufen werden, was durch Untersuchung des Quellcodes möglich ist.

Wir betrachten folgenden Fall:

PATH A^D; B^D; C; D; E^D; F^D; G END

und wollen anhand eines Aufrufbeispiels eine worst-case-Betrachtung machen.

X^n bedeute Aufruf durch normale Prozesse,
X^s Aufruf durch Stellvertreter.

Zeitintervall	Aufruf	zu reservierende Platz
1	A^n B^n	–
2	A^s B^s	$\|A\| + \|B\|$
3	C^s	$\max(\|C\|, \|D\|)$
4	D^s	–
5	E^n F^n	–
6	E^s F^s	$\|E\| + \|F\|$
7	G^s	$\|G\|$
		$\|A\| + \|B\| + \max(\|C\|, \|D\|) + \|E\| + \|F\| + \|G\|$

Man ist also nur wenig vom Bedarf bei statische Allokation entfernt. Für den allgemeinen Fall vermuten wir:

$$S = \max(A_i) + \sum A_j$$

A_i einfach, A_j doppelt

Probleme: Gesucht wird der Algorithmus für die Speicherberechnung im allgemeinen Fall, bei Benutzung aller Operatoren.