Over stapeladministratie. II

0. Inleiding.

Ik had aanvankelijk gehoopt, dat we een min of meer volledig overzicht zouden krijgen over de in aanmerking komende mogelijkheden en dat we tevens de criteria zouden krijgen met behulp waarvan we tussen deze mogelijkheden zouden kunnen kiezen. Dit vlot echter niet zo erg, mijn geduld raakt wat op en ik wil wel eens een voorstel zien, dat in elk geval werkt en niet te gek is. Ik hoop dus nu de verdieping van ons inzicht op een andere manier te bereiken, nl. door critische beschouwing van een concreet voorstel. De rest van deze notitie is aan zulk een voorstel gewijd.

Mijn eerste zorg zijn drie fundamentele problemen, te weten:
a) abrupte blokverlating
b) stapelverschuiving
c) paginavrijgave voor zg. grote arrays.

Daarbij komt dan nog een practisch probleem aan de orde, nl. optimalisering van de red - en herstelprocedure in het geval de zg. "simpele impliciete subroutine".

Over de stapelverschuiving het volgende: behalve de lijstjes van de referentiepunten (de displays) hoeven we in de stapel geen absolute adressen op te nemen, die verwijzen naar punten in de stapel; deze laatste kunnen we altijd opbergen relatief t.o.v. het begin van de stapelbodem. (Of we het altijd zullen doen is een tweede: aangenomen, dat [illegible] stapelverschuivingen dingen zijn, die tot de zeldzaamheden behoren, krijgen we wellicht een sneller systeem, als we waar mogelijk met physische adressen werken.)

Uit het bovenstaande volgt, dat het belangrijk is, dat in de stapel de referentielijstjes gevonden kunnen worden; het is hier, dat we onderscheid maken tussen een procedure (dwz. nieuw lijstje) en ingewikkelde impliciete subroutine (dwz. terugvallen op een oud lijstje). De simpele impliciete subroutine kunnen we hier dacht ik buiten beschouwing laten, omdat deze immers in doofheid uitgevoerd wordt

Ik ga nu stapelbeelden voorstellen om toe te voegen bij activering van een procedure, resp. ingewikkelde parameter.

1. Aanroep van een procedure.

Het is de bedoeling dat het volgende mechanisme gebruikt wordt zowel voor de formele als voor de nonformele procedure, zowel voor de type procedure als de non-type procedure. Het is de bedoeling dit mechanisme blindelings te volgen in het geval van de type procedure call als zelfstandige statement. (De type-procedure zal zijn waarde in het F-register afleveren; bij terugkeer zal SW wijzen naar de stapeltop in de zin van Voorhoeve, dwz. naar de eerste vrije plaats in kerngeheugen. Negeren van dit resultaat is dan al heel eenvoudig!)

Ik veronderstel, dat we voor de formele plaatsen aan twee woorden genoeg hebben en dat ook de invariante representatie van het terugkeeradres aan twee woorden genoeg heeft. (Deze veronderstellingen zijn waarschijnlijk niet juist, maar dat is in dit verband niet essentieel.) Op de volgende pagina is het voorgestelde stapelbeeld gegeven.

Opm.1 Het "centerpunt" wordt dus door een statusvariabele "d" vastgehouden, die in het kader van deze introductie de waarde d = d[i] krijgt; de waarde aan

```
M[ d - 3 - 2 * nf]
M[ d - 2 - 2 * nf]      } formele plaatsen van de 1ste parameter
                        .
                        .
                        .
M[ d - 5]               .
M[ d - 4]               } formele plaatsen van nf-de parameter
M[ d - 3]
M[ d - 2]               } invariant terugkeeradres
M[ d - 1]               = nf
→ M[ d]                 = d[ i - 1]
M[ d + 1]               = L[ i]   (=d + 3)
M[ d + 2]               = bn[ i]
M[ d + 3]               = 0-de referentiepunt
                        .
                        .
                        .
```

call-kant -hier aangeduid met ■ d[i - 1]- wordt hier gered. De index i onderscheidt in deze beschrijving de vigerende activeringen van procedures en ingewikkelde parameters. NB. De index i speelt alleen een rol in deze beschrijving, in het werkende systeem wordt deze niet bijgehouden, we voeren dus geen dynamische diepteteller in. Met "bn[i]" is gemeend de heersende blokhoogte als geintroduceerd in EWD69; L[i] is de alvast in de stapel neergezette heersende waarde van L.

Opm.2 Het stapelen van "nf" is strikt genomen overbodig, maar ik wou het toch maar doen:
a) het maakt een dynamische controle op de juistheid van het aantal meegegeven parameters mogelijk
b) het maakt het -desgewenst- mogelijk om (in de bibliotheek) procedures op te nemen met een variabel aantal parameters.
c) het terugkeermechanisme hoeft van de procedure niet meer het gegeven mee te krijgen, hoeveel formele plaatsen vergeten kunnen worden.

2. Aanroep van een gecompliceerde parameter.

Dit genereert aan de top van de stapel

```
M[ d - 3]
M[ d - 2]               } invariant terugkeeradres
M[ d - 1]               = negatieve  value-address indicatie
→ M[ d]                 = d[ i - 1]
M[ d + 1]               = L[ i]   (= L[ j] met j < i)
```

Dit stapelbeeld beantwoordt aan de aanroep van de impliciete subroutine. De impliciete ■■■■ subroutine is eenvoudiger dan de volslagen procedure in zoverre er geen parameters aan worden meegegeven; hij is ingewikkelder omdat (nl. als de actuele parameter een geindiceerde variabele is) zijn aanroep ook als "left-hand side" van de assignment statement voor kan komen.

Voor deze laatste complicatie zijn ettelijke oplossingen
Het boven gegeven stapelbeeld beantwoordt aan de techniek, waarin het objectprogramma van de body zijn behoefte kenbaar maakt met behulp van "Take formal address" of "Take formal value". Welke van deze twee aanroepen gegeven wordt, moet dan in de stapel opgeslagen worden.

Een alternatief is, dat het objectprogramma slechts zegt "Go formal" en dat de impliciete subroutine behalve zijn primitieve resultaat er bij aflevert de indicatie "address" resp. "value". Het objectprogramma kan dan aan de call kant van de impliciete subroutine inlassen
a) als linkerkant "if value alarm"
b) als rechterkant "if address, take value"

Een derde mogelijkheid is, om altijd "value" en "address" af te leveren en aan call kant het gewenste te selecteren. Hierbij doe je echter altijd te veel. (Dit is bovendien naar als de waarde nog ongedefinieerd is; ik zou niet graan door een parity fout gestraft worden.)

De bovengenoemde beelden zijn gesuggereerd in de veronderstelling dat het teken van M[d - 1] steeds descriptief zou zijn voor de situatie procedure resp. impliciete subroutine. (Dit impliceert, dat een procedure zonder parameters van de call kant nf = + 0 mee moet krijgen; het onderscheid tussen "value" resp. "address" kan dan door -0 en -1 gemaakt worden.)

In beide gevallen zouden we zonder de indicatie in M[d - 1] kunnen; het onderscheid tussen de twee stapelbeelden kunnen we ook maken door de testen of er een nieuwe referentielijst is ingevoerd, dwz. of M[d + 1] = d + 3 is of niet.

3. Simpele impliciete subroutine.

In dit geval hoeft alleen het terugkeeradres in de stapel opgenomen te worden. Het onderscheid tussen value en address bestaat niet, het is nl. altijd value. (De enkele variabele identifier als actuele parameter geeft nl. geen aanleiding tot een impliciete subroutine; het moet een echte expressie zijn. Hier is het feit, dat de indicering in de actuele parameter de impliciete subroutine per definitie ingewikkeld maakt een meevaller!)

4. Stapelverschuiving.

Tijdens de simpele impliciete subroutine kan geen interruptie optreden en kan dus ook geen behoefte aan verschuiving optreden. In alle andere gevallen geeft de heersende waarde van de statusvariabele "d" ons de clou, hoe we verder moeten gaan. De referentielijsten kunnen we feilloos vinden en dus kunnen we de verschuiving bijwerken. De gestapelde L-waarden en de geketende d-waarden kunnen we eveneens feilloos vinden en ook hier kunnen we ons de luxe van physische adressen veroorloven. Dankzij de conventie van de gestapelde nf kunnen we ook alle formele plaatsen eenduidig vinden. Als we hier een goede layout voor kiezen, kunnen we ook hier misschien ons de luxe van physische adressen veroorloven. (Het is questieus of ons dat geen verdere ellende oplevert. We moeten niet de vergissing van de B5000 maken!)

5. Paginavrijgave.

Zoals d inhaakt op de jongste schakel van de dynamische ketting, zo kunnen we een statusgrootheid e invoeren (eveneens per programma), die inhaakt op de jongste storage function van een groot array; storage functions van grote arrays ketenen we op de gebruikelijke manier.

In geval van blokverlating onderzoekt men, of hierdoor SW onder de heersende waarde van e gezakt is: zo ja, dan zakt men de ketting af, daarbij alle gepasseeerde ~~kettin~~ storage functions scannend voor vrij te geven pagina's. Hiermee gaat men door totdat e onder SW gezakt is.