Het controlerende communicatieapparaat

Het volgende is een studie over het bespelen van een controlerend communicatieapparaat dat bediend wordt via een beperkt startmagazijn. Als dit startmagazijn bv. 4 startopdrachten kan bevatten, dan hebben X8 en communicatieapparaat toegang tot "array startopdracht,slotwoord [0:3]"; in de praktijk zullen deze twee array's in het geheugen dooreen gevlochten staan, maar dat doet nu niet ter zake.

Het autonoom transport leest de startopdracht en vult het slotwoord; omdat het lezen-gehoorzamen- van de startopdracht als primair ervaren wordt, noemen wij de wijzer, onder controle waarvan dit gebeurd, de "integer leegwijzer".

Twee seinpalen, AFT en IFT regelen de synchronisatie: AFT is het aantal startopdrachten in voorraad, IFT het aantal "terugmeldingen", waarvan de X8 nog geen nota genomen heeft.

In dit geval kan de handeling van het autonoom transport in eerste aanleg beschreven worden door zoiets als:

```
"L: P(AFT);
    leegwijzer:= REM(leegwijzer +1,4);
    gehoorzaam(startopdracht [leegwijzer]);
    meld terug in (slotwoord[leegwijzer]);
    V(IFT);
    goto L"
```

Met "zoiets als" wordt bedoeld
a) dat het hier irrelevant is, of leegwijzer voor of na gebruik wordt opgehoogd; in deze beschrijving hebben we "voor" gekozen.
b) dat hier over de non-OK-toestand nog niet gesproken is; we beperken ons voorlopig tot het geval, dat het allemaal goed is gegaan.

In het volgende wil ik de consequenties nagaan van het feit, dat, voorzover de X8 betreft, de "V(IFT)" een dubbele betekenis kan hebben. Dit immers meldt een gebeuren, dat in twee richtingen betekenis heeft. Enerzijds is het gericht op het verleden, en wordt hier dus gemeld, hoe iets dat vroeger gestart is, is afgelopen, anderzijds heeft het betekenis gericht op de toekomst, omdat er nu, desgewenst, weer plaats in het startmagazijn vrijkomt. Tengevolge hiervan kan het zijn, dat er nu twee processen voortgang kunnen vinden, nl. het proces, voor welks voortzetting de voltooiing der autonome handeling essentieel is en het proces, dat graag van het apparaat gebruik zou willen maken.

We voeren hiertoe een derde, geprogrammeerde seinpaal in nl. SRT = startruimtetelling. SRT geeft aan het aantal startopdrachten, waarvoor in het startmagazijn ruimte is.

Het startmagazijn wordt nu bespeeld door twee onderling asynchrone processen, viz. Starter en Controleur. De zojuist ingevoerde seinpaal dient voor de onderlinge synchronisatie van deze twee; deze twee processen hebben, zoals wij straks zullen zien, nog wel meer verknopingen.

De Controleur raadpleegt het slotwoord onder controle van een "integer controlewijzer" in ruwste vorm:

```
"Controleur: P(IFT);
   controlewijzer:= REM(controlewijzer +1,4);
   if slotwoord[controlewijzer] ≠ correct then actie fout else actie goed;
   V(SRT); goto Controleur"
```

Over "actie goed" en "actie fout" straks meer.

De Starter neemt met het startmagazijn contact op onder controle van een vulwijzer, altijd via de sequens:

```
".....; P(SRT);
        vulwijzer:= REM(vulwijzer +1,4);
        startopdracht[vulwijzer]:= volgende startopdracht;
        V(AFT);....."
```

Ook over "volgende startopdracht" straks meer; deze moet er nl. zijn.

De gepregrammeerde seinpaal SRT is ingevoerd om controle op de voltooide operatie en aanbieden van de volgende, waarvan nu immers ruimte in het starmagazijn is, in de tijd te kunnen splitsen.

Nu moeten wij in de organisatie verdisconteren:

a) dat het gegeven van een startmagazijn van 4 startopdrachten een physisch gegeven is, maar dat deze omvang niets met de logische eenheden te maken heeft, "logische eenheden" ten aanzien van controle (en overdoen) of van interesse in voltooiing.

b) dat activiteiten van Starter en Controleur op elkaar afgestemd dienen te worden: de ene zal dingen in gang zetten, de ander zal van de voltooiing kennis nemen. De laatste dient dan te weten van de voltooiing waarvan hij kennis neemt, waar dit goed voor was en welke consequenties hieraan verbonden dienen te worden.

Wij kunnen deze beide dingen bereiken, door een (eventueel groter) magazijn in te voeren, waartoe zowel Starter als Controleur toegang hebben. De elementen van dit magazijn -laten we het even actiemagazijn noemen- bevatten:

a) ten bate van de Starter startopdrachten

b) ten bate van de Controleur nadere specificaties van de handelingen "actie goed" resp. "actie fout" na voltooiing van de in dit element gegeven startopdracht.

Beide processen zullen dit actiemagazijn aftasten via een cyclisch werkende leegwijzer. Voeren wij de twee gebruikelijke seinpalen voor dit actiemagazijn in, viz. "actiemagazijn vol" en "actiemagazijn leeg", dan wordt in dit algemene schema de Starter

```
"Starter: P(SRT, actiemagazijn vol);
    starterleegwijzer:= REM(starterleegwijzer +1,N);
    vulwijzer:= REM(vulwijzer +1,4):
    startopdracht[vulwijzer]:= actiemagazijn[starterleegwijzer];
    V(AFT); goto Starter"
```

opm. Met "actiemagazijn[ ]" wordt hier het startopdrachtgedeelte van een element bedoeld; N = omvang van actiemagazijn.

De Controleur begint met

```
"Controleur: P(IFT);
    controleurleegwijzer:= REM(controleurleegwijzer +1,N);
    controlewijzer;= etc...."
```

"actie goed" wordt nu iets, dat het controleursgedeelte van "actiemagazijn[controleurleegwijzer]" verwerkt.
Hieronder kan vallen:
a) een aantal malen V(actiemagazijn leeg)
b) de V-operatie op een in het actiemagazijn opgegeven seinpaal.

Het gecontroleerd transport kan nl. een reeks afsluiten; hierdoor komt een aantal plaatsen in het actiemagazijn vrij; verder kan bij een bepaald programma hierop hebben staan wachten.

Een programma, dat een serie elementen van het actiemagazijn wil opgeven, moet de mogelijkheid hebben, deze serie "engespatieerd" door ander aanbod te kunnen opgeven. Dit kan met een binaire seinpaal AV (=aanbod vrij):

```
"P(AV);
L: P(actiemagazijn leeg);
   vulwijzer:= REM(vulwijzer +1,N);
   actiemagazijn[vulwijzer]:= volgend element;
   V(actiemagazijn vol);
   if B then goto L;
   V(AV)"
```

Opm. De elementen bevatten, hoeveel plaatsen in het actiemagazijn vrijgegeven zullen worden na controle. Het is wel zaak niet meer dan N niet vrijgevende elementen in successie aan te bieden. De veilige manier om dit te garanderen is om bij laatst aangeboden element alle elementen, die tussen P(AV) en V(AV) zijn aangeboden, vrij te doen geven. Als je niet oppast, zou het actiemagazijn "combinatorisch" dicht kunnen groeien!

Als het aantal elementen van het actiemagazijn, dat "per keer" gevuld wordt een redelijke bovengrens heeft, dan lenen deze programma's zich tot een vereenvoudiging, die nog wel illustratief is. Je kunt dan zeggen: kom, wees niet kinderachtig, stel in het actiemagazijn ruimte ter

beschikking in wat grotere porties; we noemen deze porties nu "transportschema's": een transportschema neemt de rol van een"aantal bij elkaar behorende elementen uit het actiemagazijn" over.

Het actiemagazijn wordt nu vervangen door een"transportschemabuffer"; de seinpalen "actiemagazijn leeg, resp. -vol" vervallen en worden vervangen door "transportschemabuffer leeg, resp. vol".

Het laatste programma wordt nu:

```
"P(AV, transportschemabuffer leeg);
  vulwijzer:= REM(vulwijzer +1,N1);
  transportschema[vulwijzer]:= .....;
  V(AV,transportschemabuffer vol)"
```

De Controleur bevat dan als onderdeel van "actie goed" de operatie "V(transportschemabuffer leeg)", de Starter krijgt de volgende constructie:

```
Starter: P(transportschemabuffer vol);
   starterleegwijzer:= REM(starterleegwijzer +1,N1);
L: P(SRT);
   vulwijzer:= REM(vulwijzer +1,4);
   startopdracht[vulwijzer]:= .....;
   V(AFT): if transportschema nog niet leeg then goto L;
   goto Starter"
```

Hiermee zijn enige "hoogfrequente" seinpalen geruild tegen evenveel "laagfrequente": ten koste van wat bufferruimte hebben wat werk uitgespaard. Het aantal onderling asynchrone processen is niet geslonken!

De trommel

Voor de trommel hadden we een iets afwijkend schema bedacht, dat tot op heden -12 februari 1964- nog niet verworpen is. Hierbij komt nl. nog de complicatie, dat een programma zelf geen "transportschema" opstelt: het kan slechts zijn behoefte uiten. Welke transport of transporten dit impliceert, hangt af van de momentane geheugenbezetting. M.a.w.: de "vertaling" van behoefte naar transportschema is een handeling, en het moment, waarop deze handeling plaats zal vinden moet gekozen worden. Wij hebben ons op het standpunt gesteld dat het gewenst is, deze handeling zo laat mogelijk te laten plaats vinden: het verlaten van behoefte in transportschema kan dan nl. van de meest recente gegevens uitgaan. Dit wordt "tegengewerkt" door het streven, de trommeltransport, als dit de bottleneck wordt, zo min mogelijk ledigheid te gunnen. D.w.z. de bufferingsmogelijkheden zijn hier niet bedoeld om grote asynchroniteit op te kunnen vangen, integendeel: tussen het uiten van behoefte en bevrediging daarvan zal het programma in kwestie als regel helemaal niet door kunnen werken! We hebben dus meer te maken met wat inmiddels een "morele haastsituatie" is gedoopt: buffering van startopdrachten is slechts een middel, om de morele haastsituatie te mitigeren.

Op deze gronden kwamen we tot de volgende constructie.
We hebben een transportschemabuffer, dat cyclisch wordt afgewerkt. Over het aantal transportschema's, dat in deze buffer kan, zullen we het later hebben.

Elk programma communiceert met de trommeltransport via een zg. "behoeftetoonbank"; hier is gedacht aan een toonbank met de capaciteit van 1 behoefte.

Het aanbod van een behoefte vindt plaats via

```
".....P(behoeftetoonbank leeg);
      aanbod behoefte;
      V(behoeftetoonbank vol);
      P(zekere seinpaal)....."
```

Over de"zekere seinpaal" zullen we het straks nog wat uitgebreider hebben.

Het vertalen van behoefte vindt plaats gesynchroniseerd ten opzichte van het vullen van het physisch startmagazijn van de trommel, nl:

```
VERZORGER: P(toonbank vol, transportschemabuffer leeg);
           vertaal behoefte in transportschema;
        L: P(SRT);
           zet volgende startopdracht trommel;
           V(AFT);
           if heersend transportschema nog niet leeg then goto L;
           goto VERZORGER..."
```

De controleur bevat als onderdeel van "actie goed" na controle van een heel transportschema

"V(transportschemabuffer leeg, zekere seinpaal)".

Voor de "zekere seinpaal" kan men kiezen:
a) elk programma zijn eigen seinpaal; deze moet dan (als onderdeel van de behoefte) in het transportschema worden opgenomen.
b) anderzijds weet men, dat van deze seinpalen er maar een paar tegelijkerijd actueel kunnen zijn (ik geloof 1 meer dan er transportschema's in de transportschemabuffer kunnen). Men kan dit N-tal cyclisch afwerken bij het toonbank vullen en bij het vrijgeven door de controleur. Dit is een klein detail, dat niet belangrijk is.

Uit de tekst van de VERZORGER blijkt, dat als elk transportschema minstens 1 transport vergt, het zinloos is, om meer dan vijf transportschema's te kunnen bufferen. Twee is een absoluut minimum: als je maar 1 transportschema hebt, is bij overgang van het ene transportschema naar het volgende de trommeltransport bezig even ledig. Gezien de tijd van transporten (tientallen millisecunden) dachten we niet, dat een transportschemabuffer van meer dan twee erg verdedigbaar is.