Oefeningen Computersystemen (4)

1. Producer-Consumer

De situatie waarbij 1 proces gegevens genereert die door een ander proces verwerkt worden komt veelvuldig voor. Een voorbeeld is de printerdriver uit hoofdstuk 6. Om tijdelijke snelheidsverschillen tussen de producent en de consument te absorberen en om de efficientie van de communicatie te verhogen wordt er tussen beide een buffer voorzien. De producent schrijft sequentieel in de buffer, terwijl de consument er sequentieel uit leest. De buffer kan "oneindig" zijn, i.e. de producent moet nooit wachten op de consument, of "bounded",. i.e. eindig : als de buffer vol is doordat de productiesnelheid (tijdelijk) hoger was dan de consumptiesnelheid moet de producer wachten tot de consumer een plaats vrijmaakt in de buffer. Voor deze oefening beschouwen we een "bounded" buffer.

Het schema voor beide processen is als volgt :

Producer

```
…
WHILE TRUE
        …
        schrijf resultaat in buffer
```

Consumer

```
…
WHILE TRUE
        lees een gegeven uit de buffer
        …
```

waarbij de buffer met bijhorende lees- en schrijfpointer als globale variabelen, i.e. toegankelijk voor beide processen, gedeclareerd wordt. We veronderstellen een preemptive scheduler : elk process kan op elk moment onderbroken worden.

1.a Beschouw de printerdriver uit hoofdstuk 6 (tabel 6.4 en 6.5) :

Producer

```
…
WHILE TRUE
        …
        Bufnum := Bufnum + 1;
        WHILE (Bufnum > Bufsize)
                        DO (*nothing*) END;
        Buffer[in] := ch;
        in := (in +1) MOD Bufsize;
```

Consumer

```
        …
WHILE TRUE
        Bufnum := Bufnum -1;
        WHILE (Bufnum<0)
                DO (*nothing*) END;
        Data := Buffer[out];
        out := (out+1) MOD Bufsize;
        …
```

Welk van de volgende variabelen is kritisch (i.e. exclusieve toegang moet hiervoor verzekerd worden) :

1.a.1	Buffer
1.a.2	Bufnum
1.a.3	in
1.a.4	out
1.a.5	Bufsize
1.a.6	Data
1.a.7	ch

1.2 Herschrijf beide stukken code waarbij exclusieve toegang tot de kritische variabele(n) m.b.v. Dekker's algorithme verzekerd wordt.

1.3 Herschrijf beide stukken code met (een) semafo(o)r(en) en bijhorende "wait" en "signal" operatie(s).

Opgaven 1.4 en 1.5 zijn toepassingen van semaforen en deadlock avoidance. Alhoewel het sterk aangeraden wordt om over de opgaven na te denken zijn deze **facultatief**.

1.4 *Multiple Readers* Probleem

Dit is een complexere variant van het "Producer-Consumer" probleem die veelvuldig voorkomt in databanken en aanverwante systemen.

Gegeven een bestand dat gewijzigd kan worden door *1 enkel* "writer" proces, maar dat gelezen kan worden door *meerdere* "reader" processen. Aangezien een reader de gegevens niet wijzigt kunnen op elke moment meerdere readers *tegelijkertijd* het bestand raadplegen. Als de "writer" het bestand echter wil wijzigen moeten hij er zeker van zijn dat er geen ongewenste interactie met een of meerdere readers kan gebeuren, i.e. de writer moet *exclusieve* toegang tot het bestand verzekerd worden.
1.4.a Schrijf een oplossing voor dit probleem op basis van 1 of meerdere semaforen.

1.4.b Wat gebeurt er als er een kontinue stroom van leesopdrachten is : kan Uw "writer" dan aan de beurt komen ?

1.4.c Wat gebeurt er als er een continue stroom van schrijfopdrachten is : kunnen Uw readers dan aan de beurt komen ?

De toestand waarbij een (groep) proces(sen) een resource (het bestand in dit geval) monopolizeert noemt men "*starvation*".

1.5 *Dining Philosophers*

Beschouw 5 filosofen die hun leven doorbrengen met eten en nadenken. Bij het nadenken is er geen interactie met de andere filosofen … Na enige tijd wordt een filosoof hongerig. Na het eten begint elke filosoof terug te denken.

Het eten gebeurt in een eetkamer aan een ronde tafel waar elke filosoof een stoel heeft (cf. figuur). Centraal staat er een grote pot spaghetti (deze wordt constant bijgevuld en alle filosofen kunnen zonder probleem tegelijkertijd spaghetti op hun bord scheppen). Elke filosoof heeft eveneens een vork. Om te kunnen eten heeft de filosoof echter *twee* vorken nodig, i.e. zij heeft ook de vork van haar buurman nodig … Vooraleer te eten neemt elke filosoof dus twee vorken op : de zijne en die van zijn buurman. Vervolgens eet hij (voor een onbepaalde, doch eindige tijd) om tenslotte beide vorken terug neer te leggen. Hierna gaat zij opnieuw nadenken (opnieuw voor een onbepaalde, doch eindige tijd), waarna ze terug hongerig wordt …

1.5.a Verklaar waarom er enkele jaren geleden een groep filosofen dood zittend rond de tafel gevonden is ?

1.5.b Beschouw dat de kritische resources beschermd worden door semaforen. Ontwerp een oplossing die gegarandeerd deadlock vrij is en voeg deze toe aan onderstaand schema.

```
Procedure Filosoof
BEGIN
        Denk_Na();
        Wait_For_Left_Fork();
        Wait_For_Right_Fork();
        Eet();
        Signal_Left_Fork();
        Signal_Right_Fork();
END Filosoof
```