OPDRACHT 4. SEMAFOREN

In deze opdracht onderzoek je hoe je de acties van processen (of threads) kunt synchroniseren en mutual exclusion kunt realiseren door middel van semaforen.

1. Voorbereidingen

Bestudeer Stallings, met name hoofdstuk 5.3.
"Alles" over semaforen kun je vinden in:

- The Little Book of Semaphores
  http://www.greenteapress.com/semaphores/

2. Taken

2.1 POSIX Semaphores

Voor informatie over POSIX semaphores, zie de `sem_overview` man pages.
http://www.kernel.org/doc/man-pages/online/pages/man7/sem_overview.7.html

In het programma `sem.c` zie je hoe semaphores gebruikt kunnen worden.
Geef bij het compileren de optie `-lrt` mee (zodat de POSIX rt library meegenomen wordt bij het linken).

Start het meerdere keren op in verschillende shells en speel een scenario af (bijvoorbeeld: ene proces creëert een semaphore, de ander opent het; ene proces wacht op een semaphore, de ander laat passeren)

- Beschrijf een scenario dat je uitgevoerd hebt, en wat je dan waarneemt.
- Waar kan je de semaphore in het filesysteem terugvinden?
- Wat zie je als je een hexdump (zie `od`) van die file maakt? Check dit als je enkele keren 'post' hebt gedaan, en daarna nogmaals als je enkele keren 'wait' hebt gedaan (zodat alle processen geblokkeerd zijn).

Maak de volgende uitbreidingen:

- Als je in `sem.c` een nieuwe semaphore creëert, dan wordt de initiële waarde op 1 gezet. Pas dit aan dat de gebruiker bij optie 'n' ook kan intypen wat de initiële waarde is.
- Een gecreëerde semaphore in `sem.c` krijgt read+write permissies voor de user. Wijzig dit zodat de gebruiker bij optie 'n' ook kan intypen wat de permissions voor de `group` en de `other` worden. Let op: permissies worden vaak *octaal* geschreven, en niet *decimaal*!

Met `ls -l` (in de juiste directory!!!) kan je controleren of de read+write permissies van de semaphore goed staan. Waarschijnlijk zal je merken dat het niet direct goed gaat. In de manual pages van `sem_open` (bijv. http://linux.die.net/man/3/sem_open) zie je wat de oorzaak is. Ga dan kijken wat `umask` doet, en hoe de permissies wel toegekend kunnen worden. Om de parameter van `umask` te begrijpen moet je hard nadenken, maar op deze site is het duidelijk beschreven: http://nl.tech-faq.com/umask.shtml.
Geef een uitleg wat je hebt gedaan rondom de permissies.

2.2 Synchronisatie

Gevraagd worden 4 processen A, B, C, D (dus geen *threads*!) die het volgende herhaaldelijk uitvoeren:

> Ze drukken de getallen 1 t/m 8 af op een gemeenschappelijke terminal; Hierbij drukt proces A de getallen 1 en 5 af; proces B de getallen 2 en 6; proces C de getallen 3 en 7; proces D de getallen 4 en 8.

Eisen:

- De getallen worden in de "goede volgorde" afgedrukt
- Er mogen alleen semaforen gebruikt worden om te synchroniseren (dus geen busy-wait lussen en geen shared variabelen).
- Het mag geen verschil uitmaken welk proces als eerste wordt opgestart
  (maar het is wel toegestaan om de semaforen van te voren al te creëren, bijv. met een ander proces)

2.3 Rendez-vouz

We hebben vier *processen* die allen twee statements (`statement_1` en `statement_2`) uit willen voeren.
Schrijf één programma dat je vier keer tegelijkertijd opstart zodat er vier processen draaien.
Elk proces mag zijn `statement_2` pas uitvoeren *nadat* alle vier de processen hun eerste `statement_1` hebben uitgevoerd. Alle `statement_1`'s moeten door elkaar uitgevoerd kunnen worden, en dat geldt ook voor alle `statement_2`'s.
De code ziet er zoiets uit als:

```
initialisatie
statement_1
sync_code
statement_2
```

De synchronisatie (`sync_code`) *moet* met semaphores en shared memory plaatsvinden; niet met busy-wait lussen (het is overigens ook mogelijk om uitsluitend semaphores te gebruiken (zonder shared memory)).

Tips:

- Met behulp van een command line parameter kan je aangeven dat het eerste opgestarte proces de semaforen en shared memory moet *creëren*, en dat de andere processen ze slechts hoeven te *openen.*
- Bedenk eerst heel goed het algoritme voordat je gaat coderen. Hoe weet een proces in `sync_code` dat hij moet wachten op de anderen, en hoe weet hij dat hij door mag gaan?
- `statement_1` en `statement_2` kunnen een `printf` zijn waarin bijvoorbeeld een timestamp en het process ID van het proces staat (en wellicht nog semaphore informatie en shared memory informatie).
- Als je merkt dat grote stukken van je code heel erg op elkaar lijken: maak er een functie van met parameters.

3. Opleveren

Je dient een document op te leveren waarin je de bovenstaande experimenten uitwerkt. Voeg, waar nodig, screendumps toe om een en andere duidelijk te maken. Lever ook de bijbehorende source code in van de programma's die je gemaakt of aangepast hebt.