ES3/IPC - Opdracht 5. 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.

# 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/>

# Taken

## 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)

Zie console1.txt console2.txt console3.txt en console4.txt bij de volgende uitleggen

* Beschrijf een scenario dat je uitgevoerd hebt, en wat je dan waarneemt.
  + Er kan worden gezien dat wanneer je een WAIT doet op 2 processen en daarna in een 3e proces POST, dat er 1 van de 2 processen verder gaat, daarna na nog een POST gaat het ander process verder.
* Waar kan je de semaphore in het filesysteem terugvinden?
  + Alle Semaphore objecten bevinden zich in /dev/shm/ met als begin “ *sem.\** ”
* 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).
  + Als er 0 wachtende zijn en je blijft POST doen veranderd er niks in de file. Er wordt wel een variabel bijgehouden hoeveel wachtende er zijn. De size van een semaphore is 64 bytes. Alles is 0 aan het begin en er veranderd steeds een integer (4 bytes) naar het aantal wachtende. De eerste byte komt zo als te zien overeen met de initiële waarde.

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.

printf("Enter initial value (default 1): ");

fgets(line, sizeof(line), stdin);

sscanf(line, "%i", &initial\_value);

printf ("Calling sem\_open('%s', O\_CREAT | O\_EXCL, 0600, %i)\n",

sem\_name, initial\_value);

* + semdes = sem\_open (sem\_name, O\_CREAT | O\_EXCL, 0600, initial\_value);
* 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*!

printf("Enter permission [octal](default 0600): ");

fgets(line, sizeof(line), stdin);

sscanf(line, "%i", &permissions);

printf ("Calling sem\_open('%s', O\_CREAT | O\_EXCL, %i, %i)\n",

sem\_name, permissions, initial\_value);

* + semdes = sem\_open (sem\_name, O\_CREAT | O\_EXCL, octal\_decimal(permissions), initial\_value);

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.

Zie console5.txt en console6.txt door de permissies in te stellen kunnen bijvoorbeeld andere gebruikers toegang gegeven worden tot de semaphoren, dit is bijvoorbeeld handig wanneer je een meerdere gebruikers maar toegang tot 1 resource wilt geven op 1 omgeving

Ls –a list de juiste waarden –rwxr—r-x (0765) voor het programma, en de beginwaarde is 5 zoals opgegeven, wat te zien is in de hexdump (od –t x1):

05 00 00 00…….

Zie IPC32\_ipc\_1.zip (sem.c)

## 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)

Zie console7.txt en IPC32\_ipc\_optellen\_12345678\_ABCD.zip (sem\_mult.c)

## 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.

Zie console8.txt en sem\_rv.c in IPC32\_ipc\_sem\_rv-rendevous.zip

# 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.