ES3/IPC - Opdracht 4. Critical Sections

In deze opdracht onderzoek je de problematiek van “critical sections” die optreedt wanneer processen (of threads) data delen..

# Voorbereidingen

Bestudeer Stallings, met name de hoofdstukken 3.1, 3.2, 3.4 (tot "Mode Switching”), 4.1, 4.6, 5.1, 5.2, 6.7 en Appendix A.1.

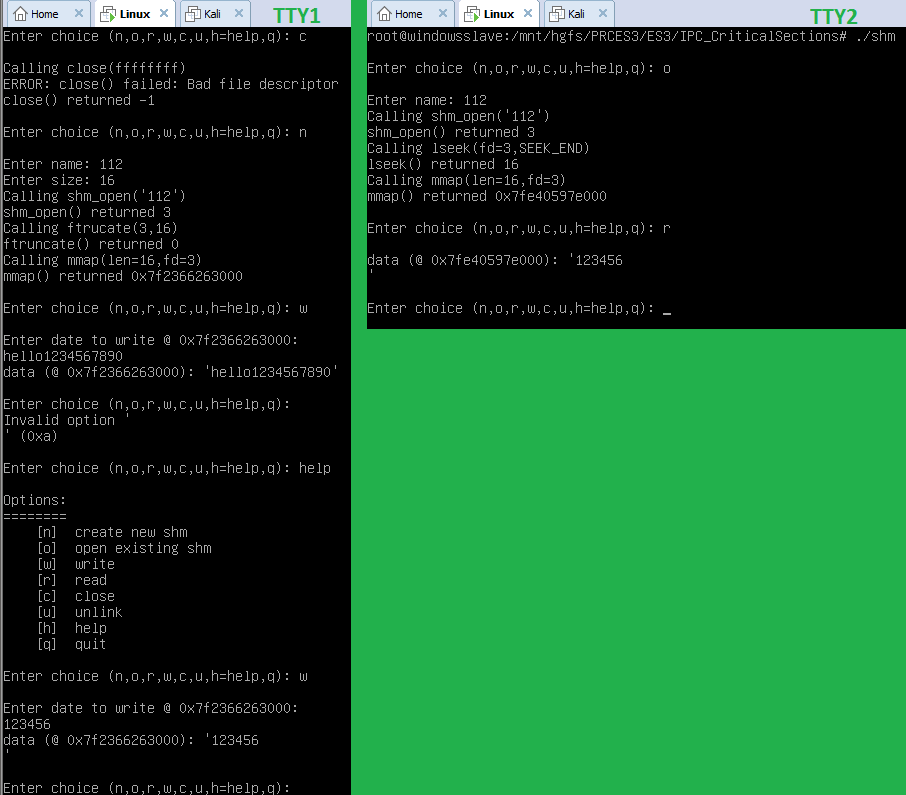
Bestudeer de praktische informatie op de volgende webpagina:

* shm\_overview - Overview of POSIX shared memory  
  <http://www.kernel.org/doc/man-pages/online/pages/man7/shm_overview.7.html>

# Taken

## POSIX Shared Memory

In het programma shm.c zie je hoe shared memory gebruikt kan worden. Start dit programma meerdere keren tegelijkertijd op en ga na of je inderdaad informatie via shared memory kan uitwisselen.



Beantwoord de volgende vragen:

* Als je in een tweede proces een stuk shared memory opent dat al gecreëerd is, hoe wordt dan de lengte ervan achterhaald?
  + Door lseek() te gebruiken, elke memory mapping heeft een begin en einde, je kunt naar het einde seeken en begin, en deze van elkaar aftrekken (de adressen). Dat is de size in bytes.
* Waar kan je het shared memory in het filesysteem terugvinden?  
  Tips:   
  a. de normale File Manager Dolphin en de Webbrowser Konqueror hebben een Find File button, (maar die is niet zo geweldig)   
  c. in een shell kan je ook het commando find / | grep <name> opgeven (bij voorkeur met root-privileges)
  + Het commando ‘find / -name ‘\*naam van shared memory\*’ geeft terug:
    - /dev/shm/naame van shared memory
* Wat zie je als je een hexdump (zie od) van die file maakt?
  + (oct) 0000000 031061 032063 033065 000012 032464 033466 034470 000060

0000020 [od /dev/shm/112]

(hex) 0000000 3231 3433 3635 000a 0000 0000 0000 0000

0000020 [od –x /dev/shm/112]

Dit is 40 bytes, terwijl de aangevraagde shm size 16 bytes is, er worden dus extra bytes gebruikt voor additionele data. De string ‘123456\n’ heeft de bytes   
(oct) 61 62 63 64 65 66 15

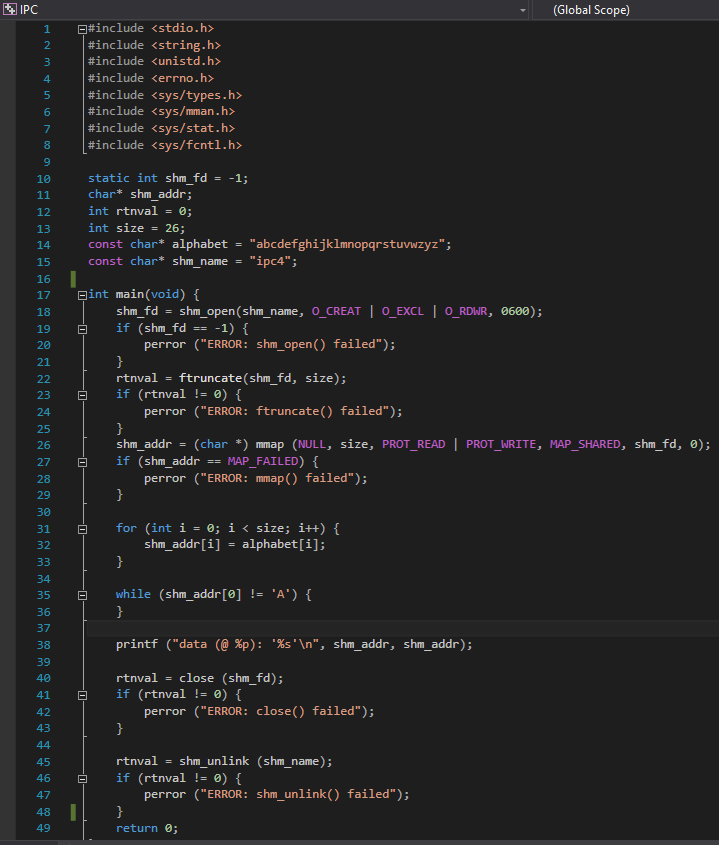
(hex) 31 32 33 34 35 36 0D

De data is hierin dus te herkennen, met nog een extra 0x20 helemaal aan het einde, niet bekend waarvoor

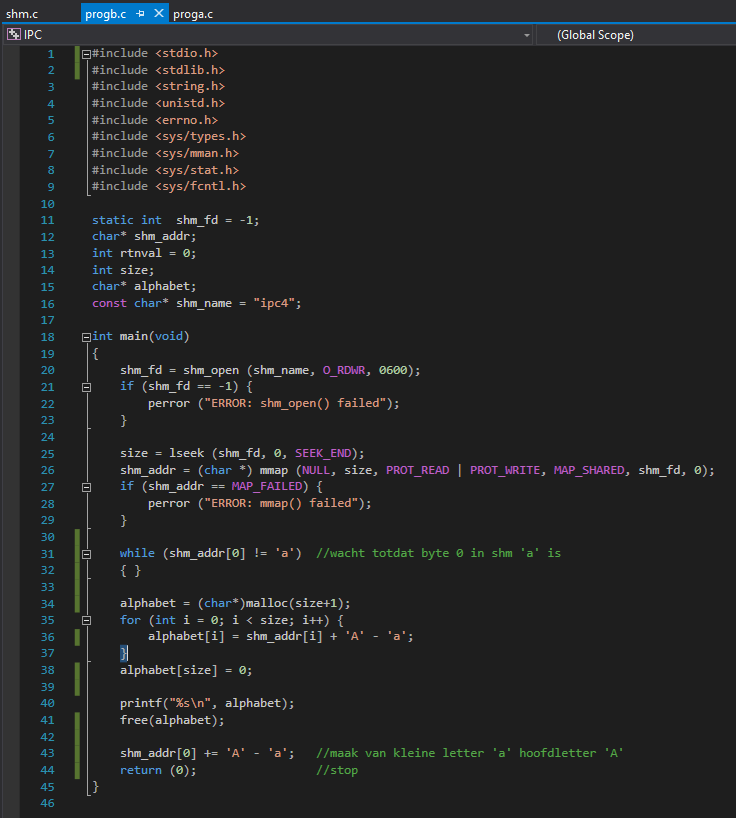


Maak nu zelf 2 programma's die *tegelijkertijd* draaien en met behulp van shared memory met elkaar communiceren. Bijvoorbeeld:

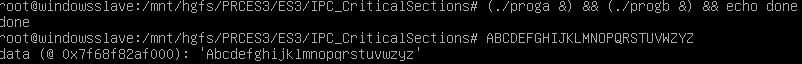
* programma A creëert een stuk shared memory, schrijft dit vol met het alfabet (in kleine letters), *wacht* tot er een hoofdletter A aan het begin staat, verwijdert het shared memory en stopt.



* programma B opent hetzelfde shared memory, *wacht* tot er een kleine letter a vooraan staat, zo ja: leest de rest van het shared memory, maakt van de kleine letter a een hoofdletter A en stopt.

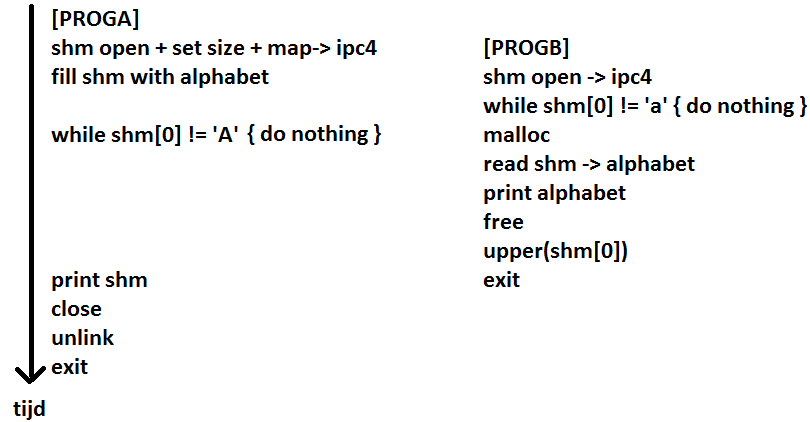






Extra uitbreidingen:

* maak een tijdsplaatje wat de processen na elkaar doen



* wat gebeurt er als proces A het shared memory verwijdert terwijl proces B er nog uit wil lezen?
  + Na ‘unlink’ (return 0 in server) kan je alsnog gewoon de memory uitlezen, ook nadat het server process weg is.
  + Na unlink van server, unlink in client geeft -1 met error no such file, read werkt nog steeds
* hoe kan je heen en weer blijven communiceren tussen de processen, bijvoorbeeld de teksten die je in een terminal intypt?
  + Shared memory maken
  + Voor beide programma’s elk 2 counters bijhouden (ieder voor zich) van:
    - hoeveel characters die in zijn eigen buffer heeft
    - en hoeveel characters die van de ander heeft gelezen
  + Check of hoeveel gelezen < characters in buffer van ander
    - Zo ja, lees meer en laat zien   
      (of lees totdat een bepaald character zoals \n aangetroffen wordt)
    - Zo niet, wait/pas eigen buffer aan
  + (dit is een primitief chat-programma)

## Dekker

Implementeer de volgende onderstaande programma's:

P0:

void main(void)

{

printf ("1\n");

printf ("3\n");

printf ("5\n");

}

P1:

void main(void)

{

printf ("2\n");

printf ("4\n");

printf ("6\n");

}

Als we deze twee processen uitvoeren zal de uitvoer 1 t/m 6 in een of andere volgorde zijn. We willen nu echter dat de uitvoer 1 2 3 4 5 6 is. Daarvoor hebben we extra synchronisatiecode nodig die tussen de printf() regels van de programma's P0 en P1 geplaatst wordt.

Dat is code die ervoor zorgt dat er op het juiste moment van process gewisseld wordt. Door die synchronisatie code zal de timeslice van een proces verstrijken, en dan geeft het Operating System de beurt aan het andere proces.

Die synchronisatiecode kan zoiets zijn als 'while(flag == 0){;}' en het andere proces moet dan op het juiste moment de flag op 1 zetten. Deze flag zit dan in shared memory.

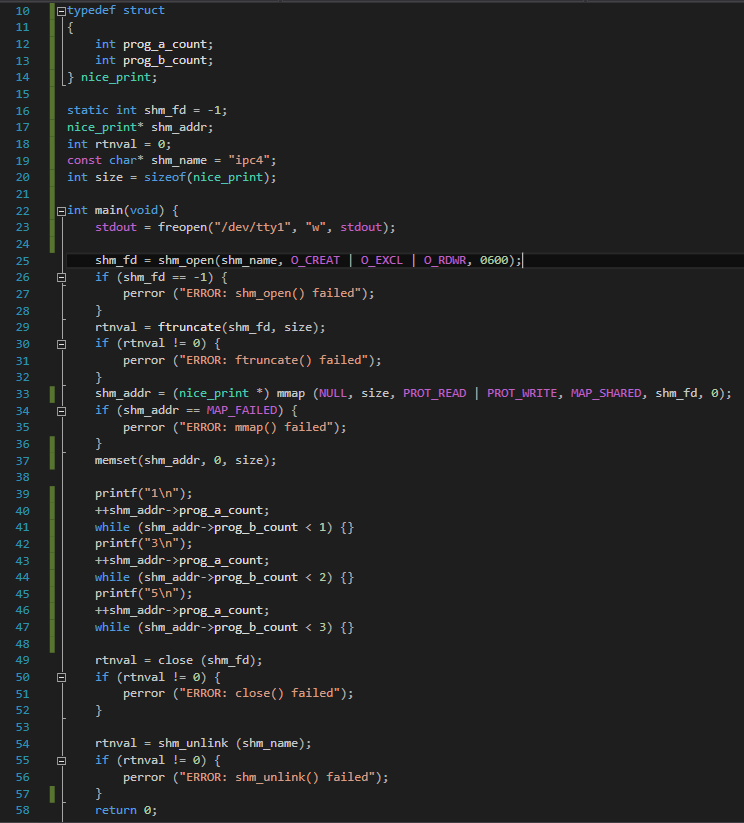
Beide processen moeten hun uitvoer naar 1 terminal sturen. Twee processen die in twee verschillende shells opgestart zijn, kunnen eenvoudig hun uitvoer naar één shell sturen. In de shell waarin je de uitvoer zou willen geef je het commando tty, dit zal als antwoord iets geven als /dev/pts/3. Start nu beide processen op en redirect hun stdout naar (bijvoorbeeld) /dev/pts/3.

Schrijf de programma's met synchronisatiecode zodanig dat de uitvoer inderdaad 1 2 3 4 5 6 wordt.

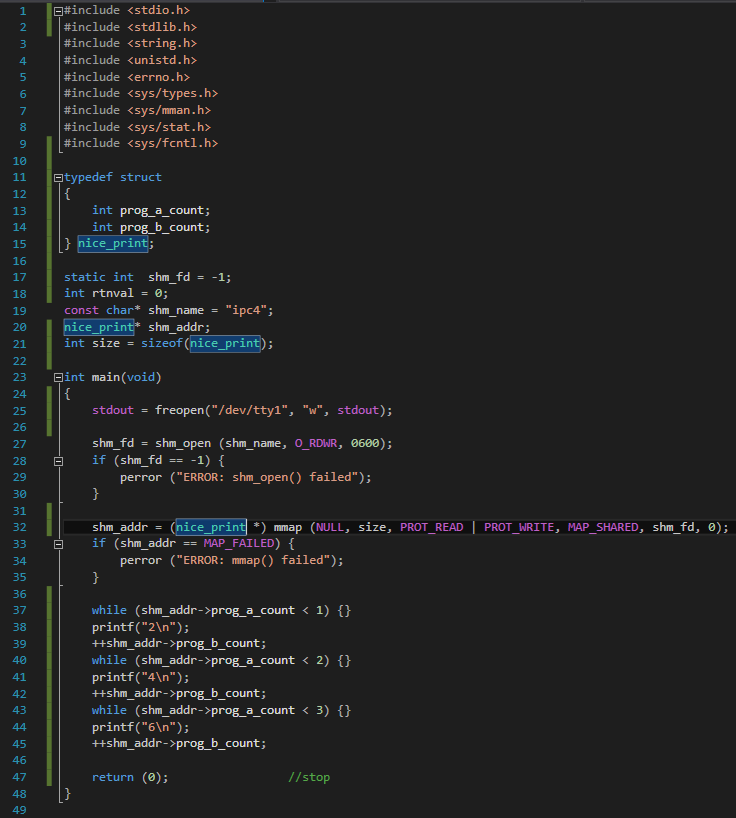
Output:



Code prog a:



Code prog b:



## Critical Section

Er draaien twee processen, met de code die hieronder staat.

*Mutual exclusion* is niet gegarandeerd. Toon dit aan door een statement volgorde te geven van beide processen, zodanig dat ze uiteindelijk allebei in de Critical Section terecht komen.

Beantwoord ook de vragen:

* kan *deadlock* optreden? (waarom wel/niet?)
  + ja, als beide processen precies tegelijk lopen dan blijft vlag bij beide true dus blijven ze altijd in de nested while (2e while)
* kan *livelock* optreden? (waarom wel/niet?)
  + nee want er is geen deadlock detectie mechanisme, livelock gebeurd (meestal) wanneer deadlock dubbel gedetecteerd wordt, waardoot allebei de processen voor elkaar uit de weg blijven gaan, en daarbij nooit verder komen.
* is deze implementatie *fair*?
  + Nee, het is mogelijk voor een thread om steeds te blijven runnen, als de 2e geen (of te weinig) cpu tijd krijgt en in de 2e while loop blijft bv wanneer erin[1] true blijft voordat de verandering door 1e thread gedetecteerd kan worden)

bool vlag[2] = { false, false };

bool erin[2] = { false, false };

***process 0: process 1:***

while (true) while (true)

{ {

vlag[0] = true; vlag[1] = true;

erin[1] = false; erin[0] = false;

if (vlag[1] == true) if (vlag[0] == true)

{ {

erin[1] = true; erin[0] = true;

vlag[0] = false; vlag[1] = false;

} }

while (erin[1] || vlag[1]) while (erin[0] || vlag[0])

{ {

vlag[0] = false; vlag[1] = false;

vlag[0] = true; vlag[1] = true;

} }

CriticalSection(); CriticalSection();

vlag[0] = false; vlag[1] = false;

erin[0] = false; erin[1] = false;

} }

# Opleveren

Je dient een document op te leveren waarin je de bovenstaande experimenten uitwerkt. Voeg 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.