## OPDRACHT 3. CRITICAL SECTIONS

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

### 1. 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

### 2. Taken

### 2.1 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?
- 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)
- Wat zie je als je een hexdump (zie `od`) van die file maakt?

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?
- hoe kan je heen en weer blijven communiceren tussen de processen, bijvoorbeeld de teksten die je in een terminal intypt?
  (dit is een primitief chat-programma)

### 2.2 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.

### 2.3 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?)
- kan *livelock* optreden? (waarom wel/niet?)
- is deze implementatie *fair*?

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

***process 0:***

```
while (true)
{
     vlag[0] = true;
     erin[1] = false;
     if (vlag[1] == true)
     {
          erin[1] = true;
          vlag[0] = false;
     }
     while (erin[1] || vlag[1])
     {
          vlag[0] = false;
          vlag[0] = true;
     }

     CriticalSection();

     vlag[0] = false;
     erin[0] = false;
}
```

***process 1:***

```
while (true)
{
     vlag[1] = true;
     erin[0] = false;
     if (vlag[0] == true)
     {
          erin[0] = true;
          vlag[1] = false;
     }
     while (erin[0] || vlag[0])
     {
          vlag[1] = false;
          vlag[1] = true;
     }

     CriticalSection();

     vlag[1] = false;
     erin[1] = false;
}
```

## 3. 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.