Gelijktijdigheid: Wederzijdse Uitsluiting & Synchronisatie
Concurrency: Mutual Exclusion & Synchonization
(5e ed: 5.1-5.2, Appendix A.1)

Processes zijn meestal niet onafhankelijk

Bijvoorbeeld:

- 2 processen willen dezelfde printer gebruiken
- 2 processen willen dezelfde file lezen of schrijven
- 2 processen willen communiceren

Hoe organiseren we dit?

Threads zijn al helemaal niet onafhankelijk,
want shared memory.

Hoe organiseren we dit, zonder problemen te krijgen?

Voorbeeld: Twee Processen die Allebei Printen

P1	P2
`print(A1);`	`print(B1);`
`print(A2);`	`print(B2);`

Wat komt eruit de printer?

Voorbeeld: Twee Threads Veranderen Zelfde Variable

P1	P2
`x = x+1 ;`	`x = x+1 ;`

Als `x` *initieel 3 is, wat is* `x` *na afloop?*

Voorbeeld: Twee Threads Veranderen Zelfde Variable

Instructies niet atomair want CPU voert machinecode uit!

P1	P2
`reg = x ;`	`reg = x ;`
`reg = reg ++ ;`	`reg = reg ++ ;`
`x = reg ;`	`x = reg ;`

Als `x` *initieel 3 is, wat is* `x` *na afloop?*

Is er hier verschil tussen true concurrency op multiprocessor machine en interleaving op uniprocessor machine?

Moore's Puzzel : Twee Threads Veranderen Zelfde Variable

P1

```
while true do
  x = x+x ;
od
```

P2

```
while true do
  x = x+x ;
od
```

Als `x` *initiëel 1 is, welke waarden kan* `x` *dan aannemen?*

Race Conditie

Een situatie waarin meerdere threads of processen een gedeelde variabele lezen en schrijven, en het uiteindelijke resultaat afhangt van de relatieve timing van hun executies.

Een oplossing van de 2 processen die willen printen, bijv

```
while (claimPrinter()!=0) /* do nothing */ ;
print(A1);
print(A2);
releasePrinter();
```

zal met system calls moeten, en die zullen uiteindelijk shared memory moeten gebruiken (bijv een boolean `printerBezet`).

Oplossing: Kritieke Sectie

- identificeer bepaalde stukken code als kritieke sectie
- laat ten hoogste één process z'n kritieke sectie binnen om zo mutual exclusion (wederzijdse uitsluiting) te bereiken

Vraag: hoe realiseren we dit?

Oplossing mbv Kritieke Secties

P1

```
"enter critical section"
reg = x ;
reg = reg ++ ;
x = reg ;
"leave critical section"
```

P2

```
"enter critical section"
reg = x ;
reg = reg ++ ;
x = reg ;
"leave critical section"
```

Voorbeeld

Er dient sprake te zijn van mutual exclusion bij openen/sluiten van files.

Potentiële Problemen bij Mutual Exclusion

- deadlock: Een situatie waarin twee of meer processen niet verder kunnen omdat elk proces wacht totdat een van de anderen iets heeft gedaan.
- livelock: Een situatie waarin twee of meer processen voortdurend hun toestand veranderen in reactie op veranderingen in een ander proces zonder dat er vooruitgang wordt geboekt.
- starvation: Een situatie waarin een proces dat iets wil doen nooit wordt geselecteerd door de scheduler.
- busy waiting: Een situatie waarin een proces tijdens het wachten om de kritieke sectie binnen te gaan (veel) processortijd gebruikt.

Hoe Realiseren we Mutual Exclusion in Software?

```
var turn : 0..1;
```

(idee: `turn=0` betekent "proces 0 aan de beurt")

proces 0

```
⋮
while turn ≠ 0
    do {nothing};
"critical section";
turn:=1;
⋮
```

proces 1

```
⋮
while turn ≠ 1
    do {nothing};
"critical section";
turn:=0;
⋮
```

Oplossing werkt – mutual exclusion gegarandeerd – maar

- processen moeten om-en-om hun kritieke sectie binnen (mogelijkheid van starvation)

- busy waiting

Mutual Exclusion?

```
var flag : array[0..1] of bool;
```

(idee: `flag[0]=true` betekent "process 0 is in kritieke sectie")

proces 0

```
⋮
while flag[1]
   do {nothing};
flag[0]:=true;
"critical section";
flag[0]:=false;
⋮
```

proces 1

```
⋮
while flag[0]
   do {nothing};
flag[1]:=true;
"critical section";
flag[1]:=false;
⋮
```

Werkt niet – mutual exclusion niet gegarandeerd.

Mutual Exclusion?

proces 0

```
⋮
flag[0]:=true;
while flag[1]
   do {nothing};
"critical section";
flag[0]:=false;
⋮
```

proces 1

```
⋮
flag[1]:=true;
while flag[0]
   do {nothing};
"critical section";
flag[1]:=false;
⋮
```

Werkt wel – mutual exclusion gegarandeerd – maar mogelijk deadlock
(allebei te beleefd)

Mutual Exclusion?

proces 0

```
⋮
flag[0]:=true;
while flag[1] do
   { flag[0]:=false;
     "wacht even";
     flag[0]:=true;}
"critical section";
flag[0]:=false;
⋮
```

proces 1

```
⋮
flag[1]:=true;
while flag[0] do
   { flag[1]:=false;
     "wacht even";
     flag[1]:=true;}
"critical section";
flag[1]:=false;
⋮
```

Werkt wel, geen deadlock maar mogelijk livelock.

Dekker's Algoritme

(idee: gebruik `turn` als beide processen in kritieke sectie willen)

process 0

```
⋮
flag[0]:=true;
turn:=1;
while flag[1] do
 if turn=1 then
   {flag[0]:=false;
    while turn=1
      do nothing;
    flag[0]:=true;}
"critical section";
turn:=1;
flag[0]:=false;
⋮
```

process 1

```
⋮
flag[1]:=true;
turn:=0;
while flag[0] do
 if turn=0 then
   {flag[1]:=false;
    while turn=0
      do nothing;
    flag[1]:=true;}
"critical section";
turn:=0;
flag[1]:=false;
⋮
```

Correct, maar nog steeds busy waiting.

Model Checking

Redeneren over parallelle programma's is moeilijk en gebeurt in tekstboek Stallings erg ad-hoc.

Een meer systematische aanpak is mogelijk, met automatische beslissingsprocedures (dmv model-checking).

Dit komt aan bod in practicumopgave en in (master)cursus Analysis of Embedded Systems).

Peterson's Algoritme

(Idee: gebruik `turn` als beide processen in kritieke sectie willen.)

process 0

```
⋮
flag[0]:=true;
turn:=1;
while flag[1] and turn=1
   do {nothing};
"critical section";
flag[0]:=false;
⋮
```

process 1

```
⋮
flag[1]:=true;
turn:=0;
while flag[0] and turn=0
   do {nothing};
"critical section";
flag[1]:=false;
⋮
```

Nog steeds busy waiting ...

Mutual Exclusion mbv Hardware

- disable interrupts

- speciale machineinstructies
  - set & test
  - exchange

Disabling Interrupts

```
...
disable interrupts;
"critical section";
enable interrupts;
...
```

Alleen voor uniprocessor machines: voorkomt interleaving.

Gevaarlijk! Alleen voor korte routines van OS zelf.

Test & Set

Atomaire operatie die variable probeert te veranderen en een boolean resultaat teruggeeft of het gelukt is.

```
function testset (var i: integer) : boolean;
    { if i = 0 then { i:=1;
                      return true ; }
                else { return false ; }
    }
```

Mutual exclusion mbv Testset

```
...
while ( !testset(bolt) ) "do nothing" ;
"critical section";
bolt:=0;
...
```

Mutual exclusion in hardware is simpel maar

- busy waiting?
- starvation?
- deadlock?
- werkt dit op multiprocessor machine?

Betere primitieven: semaforen of message passing