

Over Processen en Kritieke Secties

Frits Vaandrager

Radboud Universiteit Nijmegen

Operating systemen zijn gebouwd rondom notie proces

Gebruiker werkt in omgeving met groot aantal processen:

- je shell is een process
- je www browser is een proces
- je mail programma is een proces
- het downloaden van updates wordt geregeld door een proces
- . . .

Processen zijn onafhankelijk en kunnen elkaar niet direct beïnvloeden

Elk proces heeft eigen stuk geheugen

Enkele van de 92 processen vanochtend op mijn PC:

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
4	S	0	1	0	0	76	0	-	609	-	?	00:00:04	init
1	S	0	2	1	0	94	19	-	0	-	?	00:00:00	ksoftirqd/0
5	S	0	3	1	0	65	-10	-	0	-	?	00:00:01	events/0
1	S	0	4	3	0	66	-10	-	0	-	?	00:00:00	khelper
1	S	0	5	3	0	75	-10	-	0	-	?	00:00:00	kacpid
1	S	0	27	3	0	65	-10	-	0	-	?	00:00:00	kblockd/0
1	S	0	28	1	0	75	0	-	0	-	?	00:00:00	khubd
1	S	0	39	3	0	75	0	-	0	-	?	00:00:03	pdflush
...													
0	T	4005	32525	32407	0	76	0	-	3370	finish	pts/2	00:00:02	xpdf
0	S	4005	325	32407	0	76	0	-	3388	-	pts/2	00:00:00	xpdf
0	R	4005	379	32407	0	76	0	-	675	-	pts/2	00:00:00	ps

Meerdere CPU's: echte multiprogramming en true concurrency

1 CPU: slechts 1 proces tegelijk; multiprogramming door interleaving

Hiervoor is nodig:

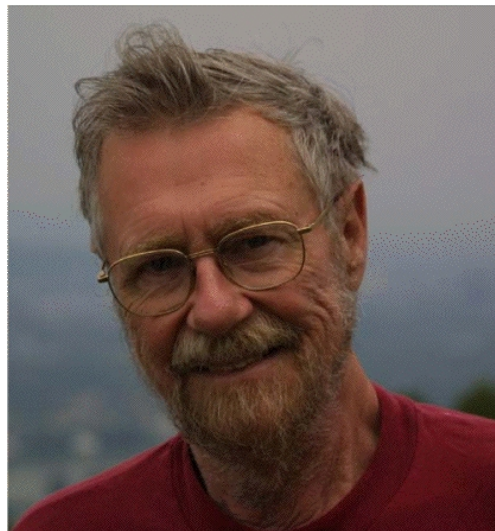
- switchen tussen processen
- administratie van context van processen
- afschermen van processen van elkaar

Operating system creëert illusie van meerdere CPU's met meerdere geheugens

De notie van proces is een cruciale ontdekking!!!!!!!!!!!!!!!!!!!!!!!!!!!!



Corbató



Dijkstra



Thompson & Ritchie

Processes zijn zelden geheel onafhankelijk:

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

Hoe organiseren we dit?

Voorbeeld: twee processen die allebei een bestand afdrukken

P1

```
print(A1);  
print(A2);
```

P2

```
print(B1);  
print(B2);
```

Wat komt eruit de printer?

Voorbeeld: twee threads veranderen dezelfde variabele

Instructies niet atomair want CPU voert machinecode uit!

P1

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

P2

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

Als c initiëel 1 is, welke waarden kan c dan aannemen?

Race conditie

Situatie waarin meerdere processen (threads) een gedeelde variabele lezen en schrijven, en het uiteindelijke resultaat afhangt van de relatieve timing van hun executies.

Race condities zijn slecht: we willen niet dat resultaat berekening afhangt van toevallige omstandigheden

Oplossing: Kritieke secties

- 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"  
print(A1);  
print(A2);  
"leave critical section"
```

P2

```
"enter critical section"  
print(B1);  
print(B2);  
"leave critical section"
```

Potentiële problemen met kritieke secties

- **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 instructies uitvoeren maar er geen 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 kritieke secties in software?

Poging 1

```
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 (gevaar van **starvation**)
- **busy waiting**

Kritieke secties: Poging 2

```
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.

Kritieke secties: Poking 3

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;  
⋮
```

Mutual exclusion gegarandeerd, maar mogelijk livelock
(allebei te beleefd).

Kritieke secties: Poging 4

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, maar wederom mogelijk **livelock**.

Kritieke secties: Dekker's oplossing

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

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

Correct, maar tricky en nog steeds **busy waiting**.

Kritieke secties: Peterson's oplossing

(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;  
⋮
```

Correct, elegant, maar nog steeds **busy waiting** ...

Mutual exclusion met hulp van hardware

- schakel interrupts uit
- speciale machineinstructies
 - test & set
 - exchange

Uitschakelen 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 ; }  
  }
```

Kritieke secties met behulp van Test & Set

```
...  
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