erasmus

HOGESCHOOL BRUSSEL

IT Essentials

Deel III: Operating Systems

2: Processen en threads

INHOUD

- Processen
- Threads
- Interprocess communicatie (IPC)
- Scheduling

erasmus
HOGESCHOOL BRUSSEL

PROCESSEN

Wat is een process?

- Een programma in uitvoering
  - Program counter
  - Registers
  - Variabelen
- Elke process heeft een virtuele CPU, in realiteit multiprogramming
- Verschil programma en process
  Voorbeeld: recept – kok - ingrediënten

erasmus HOGESCHOOL BRUSSEL

PROCESSEN

Process creation

- 4 mogelijke gebeurtenissen voor process creation
  - Opstarten van het systeem (o.a. **daemons**)
  - Een process-creation system call van een ander process (bijvoorbeeld via *fork*)
  - User request
  - Opstarten van batch job

erasmus HOGESCHOOL BRUSSEL

PROCESSEN

Process termination

- 4 mogelijke gebeurtenissen voor process termination
  - Normale exit (vrijwillig)
  - Error exit (vrijwillig)
  - Fatale error (onvrijwillig)
  - Kill door een ander process (onvrijwillig)

erasmus HOGESCHOOL BRUSSEL

PROCESSEN

Process hiërarchie

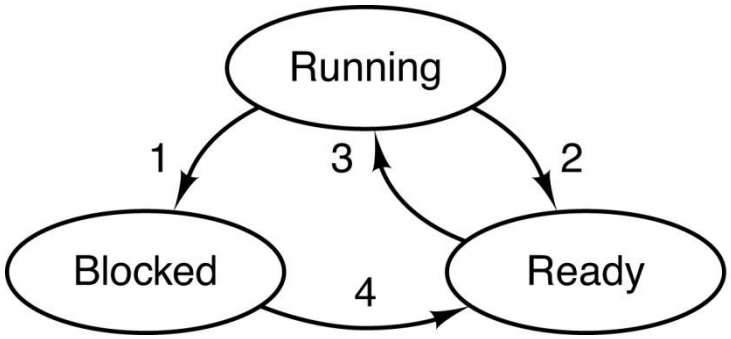
- Linux (zie process tree eerder)
- Windows werkt met handles (die mogen doorgegeven worden)

erasmus
HOGESCHOOL BRUSSEL

PROCESSEN

Process status

- 3 mogelijke statussen voor processen
  - Running
  - Ready
  - Blocked



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

erasmus
HOGESCHOOL BRUSSEL

PROCESSEN

Process implementatie

– Process tabel

Meta-data over alle processen

Process management	Memory management	File management
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

erasmus HOGESCHOOL BRUSSEL

THREADS

Wat is een thread?

- "Lightweight process"
- "process in een process"
  Vergelijkbaar met een process, maar met een gedeelde address space
- Niet in elk OS mogelijk

erasmus
HOGESCHOOL BRUSSEL

THREADS

Voordelen

- Eenvoudiger in programmatie
- Sneller om aan te maken en af te sluiten dan processen (10-100x)
- Multithreading (zie ook multiprogramming) mogelijk bij blocking threads
- Threads kunnen verdeeld worden over multicores en multi-CPU's

erasmus
HOGESCHOOL BRUSSEL

THREADS

Eigenschappen van een thread

- Geen afscherming tussen verschillende threads mogelijk en/of noodzakelijk
  - Geschreven door dezelfde programmeur
  - In tegenstelling tot processen niet in contentie maar in samenwerking
  - Threads maken zelf ruimte voor elkaar (‘yield’)
- Zelfde mogelijke statussen bij threads als bij processen
- Elke thread heeft zijn eigen stack
  - met voor elke frame in de stack een geroepen procedure die nog niet is teruggekomen
  - “De uitvoeringsgeschiedenis”

erasmus HOGESCHOOL BRUSSEL

INTERPROCESS COMMUNICATIE

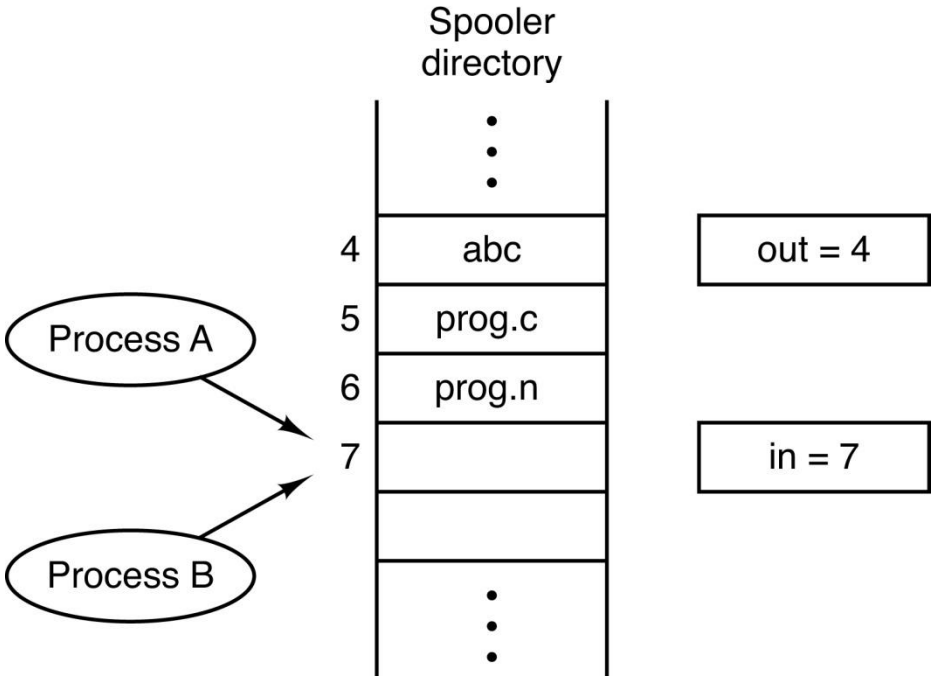
- Vaak noodzakelijk, bijvoorbeeld pipe
- 3 issues
  1. Informatie moet doorgestuurd worden
  2. Zorgen dat verschillende processen elkaar niet in de weg zitten
  3. De juiste volgorde in uitvoering
- De laatste 2 ook geldig voor threads

erasmus
HOGESCHOOL BRUSSEL

INTERPROCESS COMMUNICATIE

Race conditions

2 of meer processen gebruiken gedeelde data en het resultaat is afhankelijk van welk proces wanneer wordt uitgevoerd



INTERPROCESS COMMUNICATIE

Race condities

Zeer moeilijk te debuggen:

- Geen error
- Niet reproduceerbaar
- Komen maar zelden voor

erasmus
HOGESCHOOL BRUSSEL

INTERPROCESS COMMUNICATIE

Critical regions

- Enkel een probleem wanneer processen toegang vereisen tot shared data = **critical region** van het proces
- Oplossing: voorkomen dat meer dan 1 proces tegelijkertijd toegang krijgt tot gedeelde data (memory, files,…)= **Mutual exclusion**
- Bedoeling: slechts 1 proces mag zich in de critical region bevinden

erasmus
HOGESCHOOL BRUSSEL

INTERPROCESS COMMUNICATIE

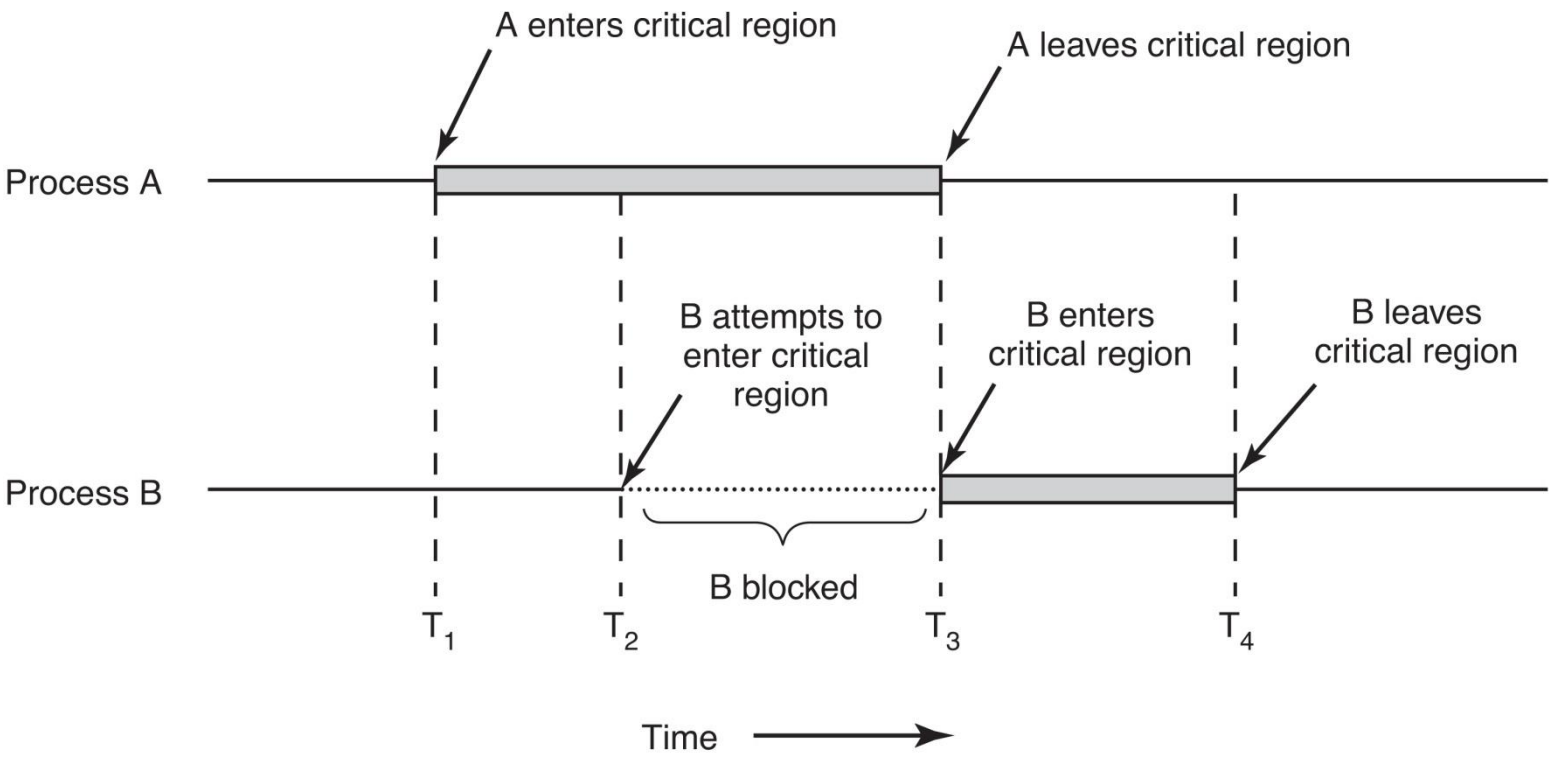
Critical regions

– 4 voorwaarden voor het vermijden van race condition

1. geen 2 processen tegelijkertijd in hun critical region
2. Geen veronderstellingen over CPU aantal of snelheid
3. Een process buiten zijn critical region mag nooit een ander process blokkeren
4. Een process mag nooit eeuwig moeten wachten om in zijn critical region te geraken

erasmus HOGESCHOOL BRUSSEL

INTERPROCESS COMMUNICATIE



INTERPROCESS COMMUNICATIE

Mutual exclusion via busy waiting

Enkel bruikbaar voor korte acties

Interrupts uitzetten

- Ok voor kernel mode (als het niet te lang duurt)
- Te gevaarlijk voor usermode
- Niet bruikbaar bij multicore CPU's

Lock variabele

- Een bit (lock variabele) gebruiken die van 0 naar 1 verandert bij het binnengaan van de critical region en terug bij het verlaten

INTERPROCESS COMMUNICATIE

Lock variabele

- Critical region mag enkel binnengegaan worden bij een 0
- Probleem bij switch van actieve process net na het lezen van de lock variabele

Strict alternation

- Variabele *turn* die bepaalt welk process aan de beurt is om zijn critical region te betreden

erasmus
HOGESCHOOL BRUSSEL

INTERPROCESS COMMUNICATIE

Strict alternation

```
while (TRUE) {
    while (turn != 0)        /* loop */ ;
    critical_region( );
    turn = 1;
    noncritical_region( );
}
```

(a)

```
while (TRUE) {
    while (turn != 1)        /* loop */ ;
    critical_region( );
    turn = 0;
    noncritical_region( );
}
```

(b)

- Probleem wanneer process in critical region wil geraken, maar turn staat op ander process dat druk bezig is met non-critical region processing: Voorwaarde 3 niet vervuld

INTERPROCESS COMMUNICATIE

Peterson's oplossing

- Combinatie van 2 voorgaande
- Ook oplossing als beiden processen quasi simultaan de critical region proberen te bereiken er plots geswitched wordt
- De 4 condities zijn voldaan

erasmus
HOGESCHOOL BRUSSEL

INTERPROCESS COMMUNICATIE

```
#define FALSE  0
#define TRUE   1
#define N      2                        /* number of processes */

int turn;                               /* whose turn is it? */
int interested[N];                      /* all values initially 0 (FALSE) */

void enter_region(int process);         /* process is 0 or 1 */
{
    int other;                          /* number of the other process */

    other = 1 – process;                /* the opposite of process */
    interested[process] = TRUE;         /* show that you are interested */
    turn = process;                     /* set flag */
    while (turn == process && interested[other] == TRUE)  /* null statement */ ;
}

void leave_region(int process)          /* process: who is leaving */
{
    interested[process] = FALSE;        /* indicate departure from critical region */
}
```

INTERPROCESS COMMUNICATIE

Sleep en Wakeup

Blocken van process ipv busy waiting

Busy waiting: *polling* voor beschikbaarheid critical region>< Sleep en wakeup: *push* => efficiënter

Beide zijn system calls

Mogelijkheid tot race conditions

erasmus
HOGESCHOOL BRUSSEL

INTERPROCESS COMMUNICATIE

Sleep en Wakeup

Het producer-consumer probleem:

- 1 proces maakt (produce) items en zet die op de queue
- een ander proces verbruikt (consume) items van de queue
- **count** houdt het aantal op de queue bij
  - Producer: +1
  - Consumer: -1

erasmus HOGESCHOOL BRUSSEL

INTERPROCESS COMMUNICATIE

Sleep en Wakeup

Probleem: count is onbeschermd

wanneer het consumerproces wordt geswitched net wanneer het de count (0) heeft gelezen, en het producerproces dit weer naar 1 zet.

Consumer ontvangt de wakeup niet, producer blijft de buffer vullen tot deze ook in sleep gaat

erasmus
HOGESCHOOL BRUSSEL

INTERPROCESS COMMUNICATIE

Sleep en Wakeup

Oplossing: werken met een wakeup bit:
    Proces zet bit op 1 bij een wakeup, ander proces zet deze terug op 0 maar gaat niet in sleep.

Meerdere processen, vereisen meerdere bits

Alternatieve oplossing via semaphoren

erasmus HOGESCHOOL BRUSSEL

INTERPROCESS COMMUNICATIE

Semaphoren

Integer die het aantal wakeup calls bijhoudt voor later gebruik

Vaak meerdere semaphoren in gebruik

Down en **up**:

- down verlaagt de semaphoor waarde bij het gebruiken van een wakeup
- Up verhoogt

Steeds een **Atomic** operatie: lezen van de waarde, wijzigen van de waarde en mogelijks in sleep gaan gebeurt zonder mogelijke onderbrekingen

erasmus HOGESCHOOL BRUSSEL

INTERPROCESS COMMUNICATIE

Semaphoren

Dubbel gebruik:

- **Synchronisatie**: full en empty semaphoren: de producer stopt bij volle buffer, de consumer bij een lege
- **Binary semaphoren** voor mutual exclusion (**mutex**)

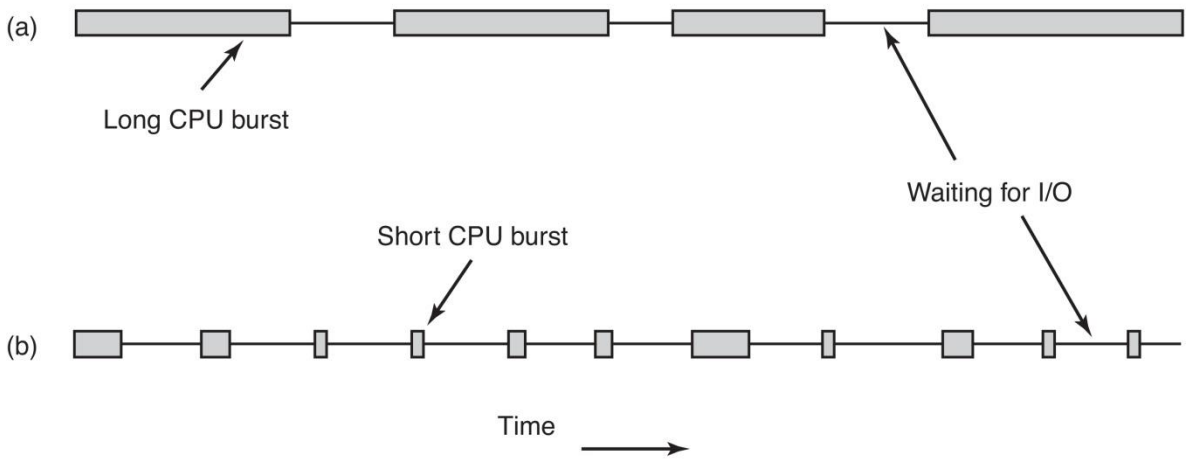
SCHEDULING

Veel processen, weinig CPU(’s): wie mag eerst?

Scheduler (onderdeel van het OS) via een scheduling algoritme

Switchen is niet efficiënt

2 soorten processen: compute-bound versus I/O-bound



SCHEDULING

Non-preemptive (NP) (aka cooperative) : processen maken ruimte voor elkaar. Worden niet onderbroken door de CPU.

preemptive (P): de scheduler beslist en onderbreekt

Onderscheid tussen omgeving

- *Batch*: eenvoudig: volgende job eerst, maar soms ook time-sharing
- *Personal computers/interactieve omgevingen*: meestal maar 1 actief programma en ruimschoots voldoende CPU, maar ongeduldige gebruiker
- *Servers*: wel een belangrijke issue
- *Mobile devices*: vaak ook beperkt in CPU

SCHEDULING

Andere omgevingen, andere noden

All systems

Fairness - giving each process a fair share of the CPU
Policy enforcement - seeing that stated policy is carried out
Balance - keeping all parts of the system busy

Batch systems

Throughput - maximize jobs per hour
Turnaround time - minimize time between submission and termination
CPU utilization - keep the CPU busy all the time

Interactive systems

Response time - respond to requests quickly
Proportionality - meet users' expectations

Real-time systems

Meeting deadlines - avoid losing data
Predictability - avoid quality degradation in multimedia systems

erasmus
HOGESCHOOL BRUSSEL

SCHEDULING

- Batchomgevingen (inventarissen, payrolls, banks,…)
  - First come, first served (NP)
  - Shortest job first (NP)

  - Shortest remaining time next (P)

SCHEDULING

- interactieve omgevingen
  - Round Robin (P):
    - » vast tijdsinterval (quantum) gedurende dewelke elk proces mag lopen
    - » Lengte van quantum: efficiëntie versus response time. Meestal tussen 20 en 50 msec.
  - Priority Scheduling (P)
    - » Vermijden dat een hoog prioriteits-process oneindig blijft lopen via quantum of prioriteitsverlaging naargelang verloop
    - » Opdeling in prioriteitsklassen, round robin binnen dezelfde klasse

erasmus HOGESCHOOL BRUSSEL

SCHEDULING

- interactieve omgevingen
  - Guaranteed scheduling (P):
    - » Eerlijk verdelen van CPU tijd per process
    - » ratio van CPU tijd per process
    - » Process met laagste CPU tijd krijgt de volgende beurt
  - Lottery scheduling (P):
    - » Processen krijgen loterijticket
    - » Hoge prioriteitsprocessen krijgen er meer
    - » Door snelle wissel (en dus vele loterijtrekkingen) hoge responstijd en eerlijke verdeling
  - Fair Share scheduling (P):
    - » CPU eerlijk verdeeld per gebruiker

erasmus HOGESCHOOL BRUSSEL

SCHEDULING

- Realtime omgevingen: time is of the essence
  - Vaak NP omdat de processen zo zijn geschreven dat ze steeds snel klaar zijn en dus niet dienen onderbroken te worden

erasmus
HOGESCHOOL BRUSSEL