Computersystems 2 Theorie

7. IPC & threads

Karel de Grote Hogeschool

11-2024

Inhoud

- Interprocess communication (IPC)
- Threads
- Threads in Linux
- Multi-processor
- Gelijktijdigheid
- Dead-lock
- Herhalingsvragen

Interprocess communication (IPC)

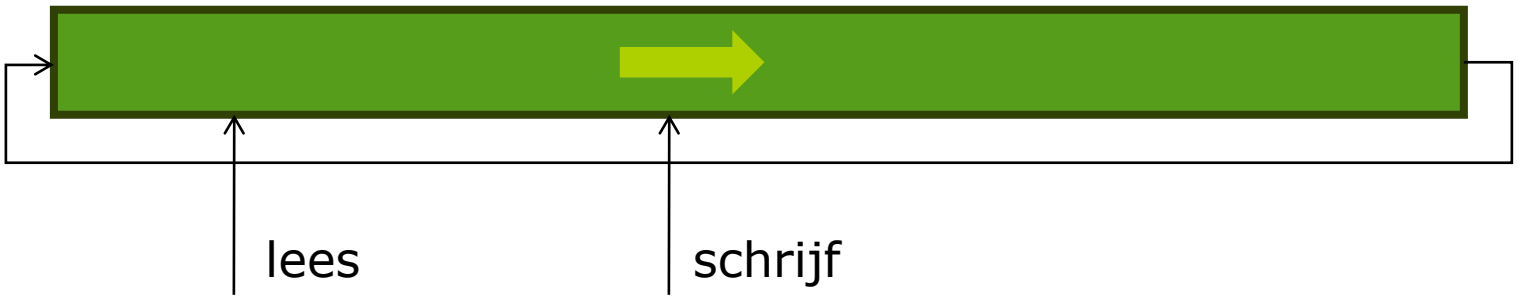
- processen hebben eigen stuk afgeschermd geheugen
- processen moeten kunnen communiceren
  - copy/paste
  - parallel processing
  - bij µ-kernel

Interprocess communication (IPC)

- Unix
  - pipes
  - Berichten/message queue
  - shared memory
- Windows
  - pipes, clipboard, files

Unix en Windows pipes

- ieder proces heeft stdin, stdout en stderr
- koppel stdout van 1 proces aan stdin van ander proces
- cat <filename> | grep 'class' | sort
- FIFO
- OS heeft per pipe een circulaire buffer waarin geschreven en gelezen kan worden:
- C: via pipe() system call



Unix berichten

- Unix voorziet message queues, toegankelijk via een id
- Unix system calls
  - int msgget(key, flags)
    - creëert een message queue
    - geeft id terug
  - void msgsnd(qid, message, size, flags)
    - zendt een bericht naar een gegeven queue
  - void msgrcv(qid, message, maxSize, type, flags)
    - haalt een bericht van een queue
      - type=0: FIFO
      - type>0: leest bericht van dit type
      - type<0: leest bericht met laagste type (=prioriteit)
  - void msgctl(qid, cmd, data)
    - o.a. gebruikt om queue te verwijderen

Unix berichten

client

```
#define MSGKEY 0x7B
struct msgform {
  long  type;
  char  message[255];
  };
int main() {
  int msgid=msgget(MSGKEY,0777);
  printf(“Ga bericht sturen op de queue\n");
  struct msgform message;
  message.type = 1;
  strcpy(message.message, "Hello, world!");
  msgsnd(msgid,&message,255,0);
}
```

server

```
int main() {
  int msgid=msgget(MSGKEY,0777|IPC_CREAT);
  printf("message queue %d created.", msgid);
  printf("Nu wachten...\n");
  struct msgform message;
  msgrcv(msgid, &message, 255, 1, 0);
  printf("bericht ontvangen: ");
  printf("%s\n", message.message);
  msgctl(msgid, IPC_RMID, 0);
  }
```

Compileer server.c en client .c en start de server in een terminalvenster.

Met welk commando kan je in Linux de message queues zien?

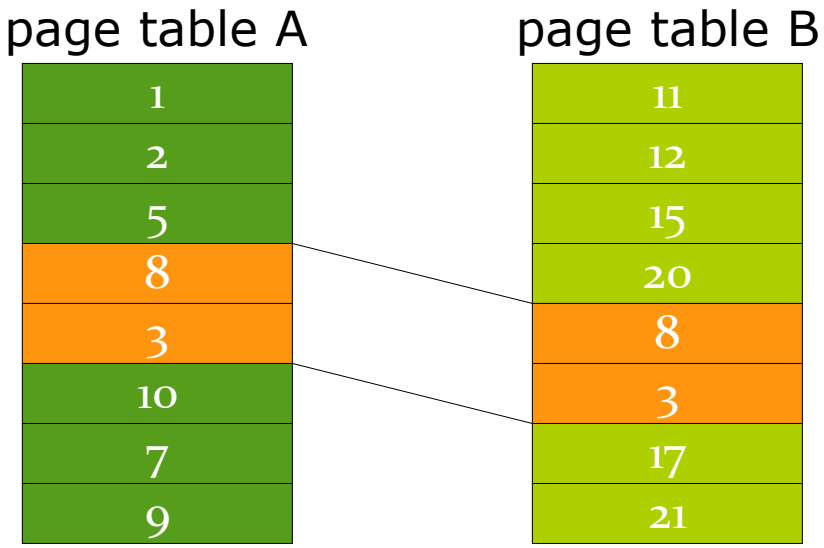
Start daarna de client in een ander terminal venster.

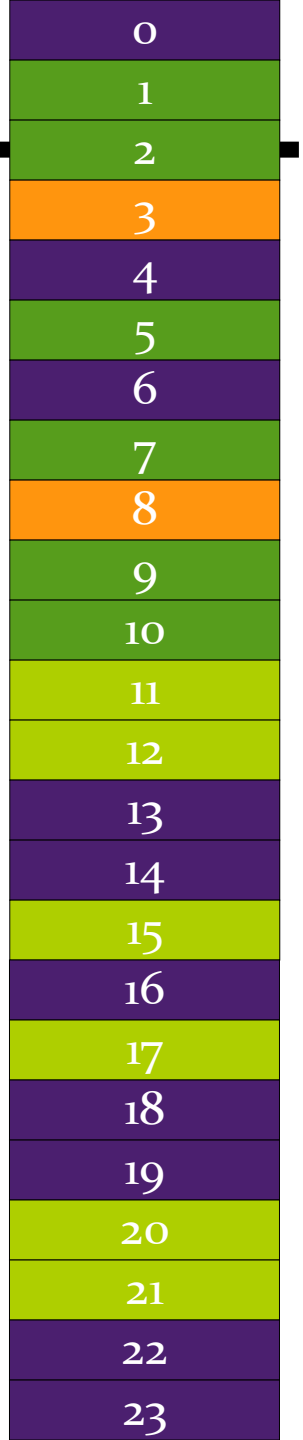
Unix shared memory

- Unix kan een stuk geheugen aan verschillende processen toewijzen
- dit geheugen wordt niet vrijgegeven als een proces stopt
- system calls
  - int shmget(key, size, flags)
  - void *shmat(mid, address, flags)
    - mapt shared memory op een adres
  - void shmdt(address)
    - unmapt shared memory
  - void shmctl(mid, cmd, data)

Unix shared memory

- shared memory wordt geïmplementeerd adhv paging





Inhoud

- Interprocess communication (IPC)
- Threads
- Threads in Linux
- Multi-processor
- Gelijktijdigheid
- Dead-lock
- Herhalingsvragen

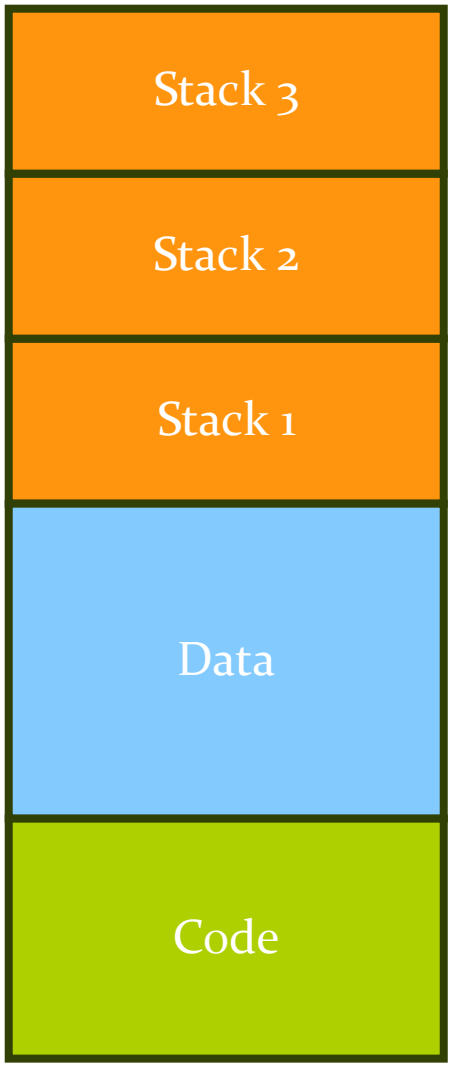
Threads

- problemen met processen
  - processen opstarten duurt "lang"
  - context-switch duurt "lang"
  - sommige applicaties hebben nood aan verschillende processen die met dezelfde data (variabelen) werken
  - IPC mechanismen zijn dan te complex

Threads

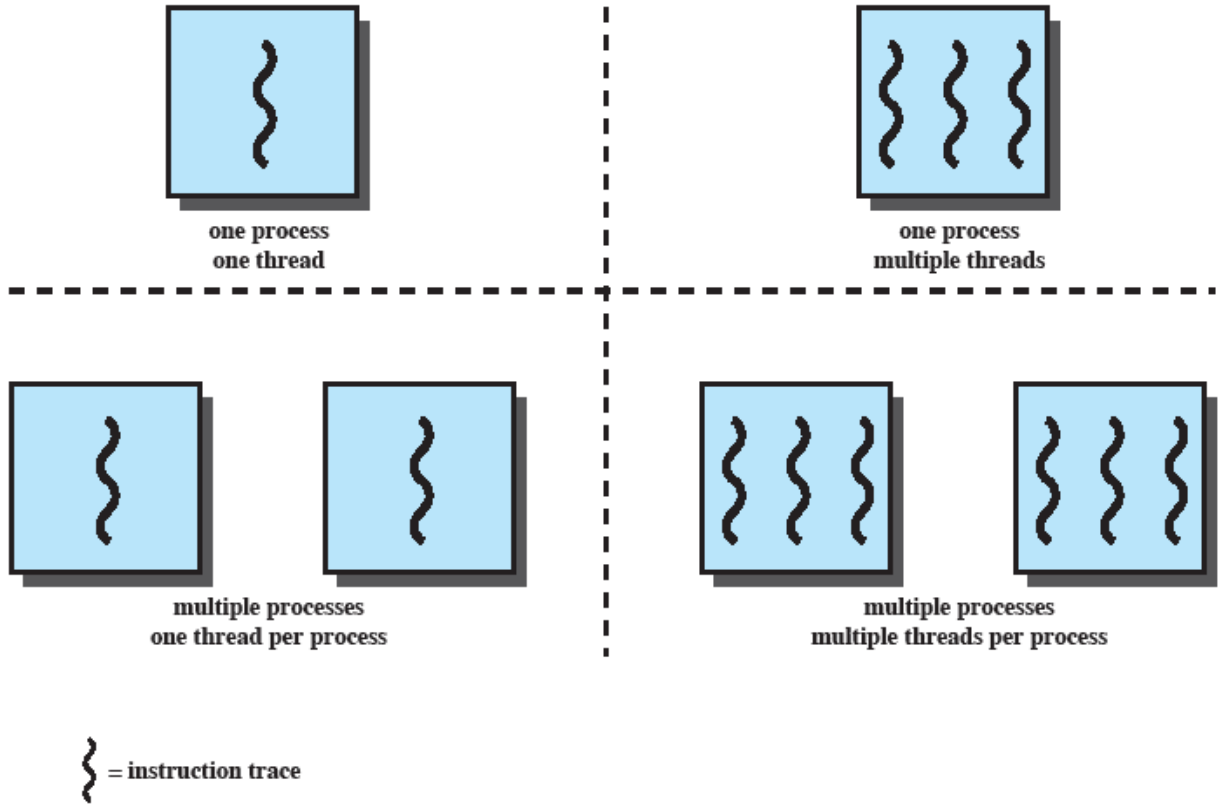
- light-weight process = thread
- 1 proces kan verschillende threads hebben
- threads delen
  - code-segment
  - data-segment
- threads hebben wel eigen stack
- context-switch is eenvoudiger

Threads



1 proces
3 threads

Threads


one process
one thread
one process
multiple threads
multiple processes
one thread per process
multiple processes
multiple threads per process
= instruction trace

Threads: implementatie

- user-level threading
  - N:1
  - N threads : 1 OS scheduling entiteit
  - ieder proces heeft zelf een scheduler voor threads
  - kernel weet zelfs niet dat er threads zijn
  - voordeel: geen switch naar kernel-mode (snel)
  - nadeel:
    - meer logica in proces
    - als proces in 'ready' staat, dan staan alle threads ook in wacht
    - threads van 1 proces draaien op 1 processor/core (zelf bij multi-core / multi-processor)

Threads: implementatie

- kernel-level threading
  - 1:1
  - 1 thread : 1 OS scheduling entiteit
  - OS regelt scheduling van threads
  - voordeel:
    - transparant voor de processen
    - OS kan threads schedulen, onafhankelijk van processen
    - threads van 1 proces kunnen op verschillende processoren / cores draaien
  - nadeel: switch naar kernel-mode nodig (trager)

Threads: implementatie

- hybrid threading
  - M:N
  - M thread : N OS scheduling entiteiten
  - Combinatie van 1:1 en M:1
  - Combineert voordelen, meer complexe implementatie

Threads

- communicatie tussen threads is veel eenvoudiger
  - data- en code-memory is altijd shared
  - opgelet: lokale variabelen en parameters staan op de stack en zijn dus niet shared!

Inhoud

- Interprocess communication (IPC)
- Threads
- Threads in Linux
- Multi-processor
- Gelijktijdigheid
- Dead-lock
- Herhalingsvragen

Threads in Linux

```
#include<pthread.h>
pthread_t tid1, tid2;
char bericht[30];

void* doThread1(void *arg)
{
    printf("1 stuurt naar 2\n");
    strcpy(bericht,"Bericht van 1");
    sleep(10);
}

void* doThread2(void *arg)
{
    sleep(10);
    printf("2 ontvangt : %s\n", bericht);
}
```

```
int main(void)
{
    pthread_create(&tid1, NULL, &doThread1, NULL);
    printf("\n Thread 1 created\n");
    pthread_create(&tid2, NULL, &doThread2, NULL);
    printf("\n Thread 2 created\n");
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    printf("\n Threads done!\n");
    return 0;
}
```

gcc threadvb.c -o threadvb **-pthread**

Compileer vorig vb. en test dit uit.

Met welk commando kan je in Linux de threads zien?

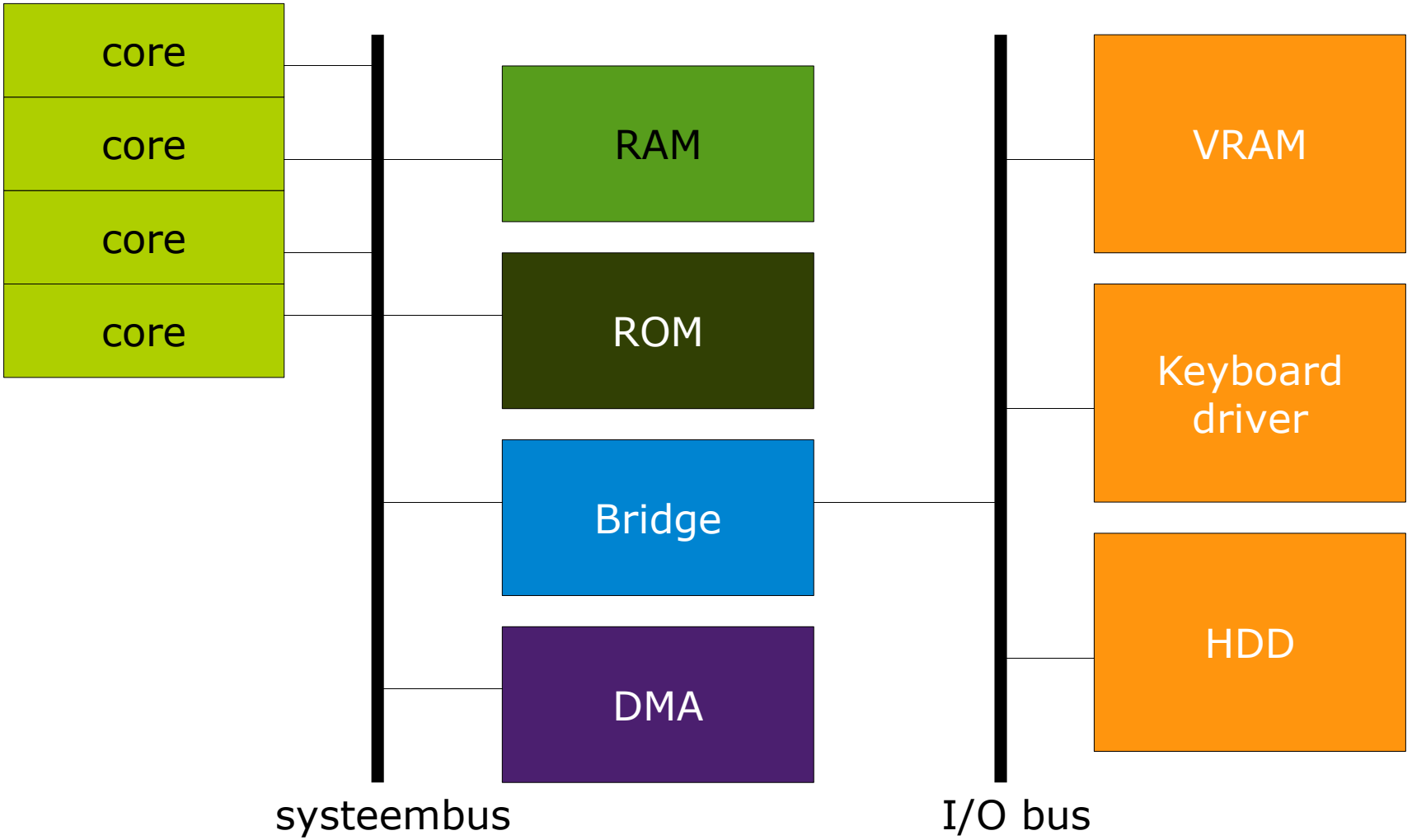
Inhoud

- Interprocess communication (IPC)
- Threads
- Threads in Linux
- Multi-processor
- Gelijktijdigheid
- Dead-lock
- Herhalingsvragen

Multi-processoren

- multi-processor/multi-core
  - meerdere 'cores' of meerdere processoren
  - meestal shared memory
- gevolgen voor OS
  - 1 ready-queue, meerdere processoren
  - load-balancing
  - conflicten verwerken als processoren dezelfde resource willen gebruiken

Architectuur met multi-processor/core

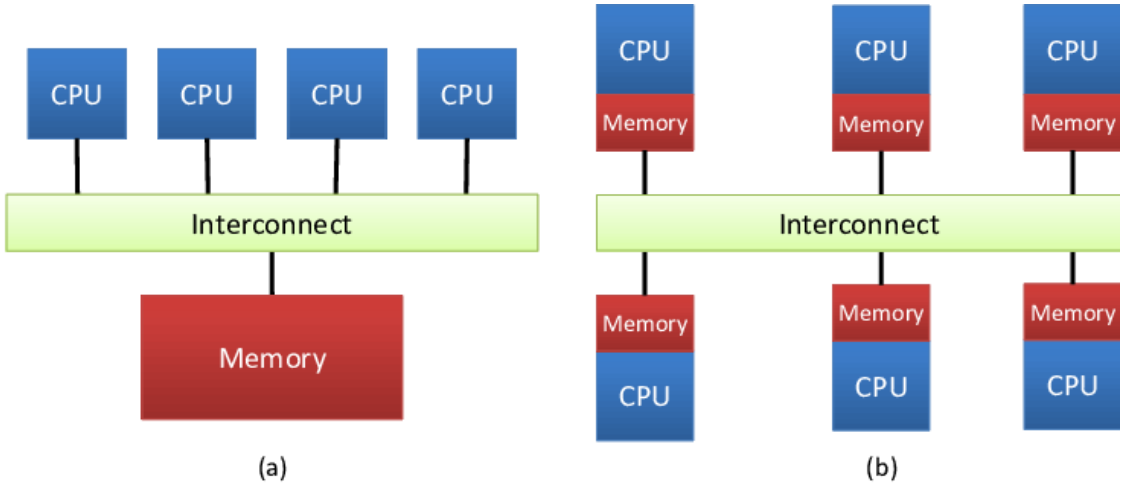


Shared Memory Multi-processor: strategieën

- master-slave
  - 1 processor is de master, de anderen zijn slave
  - de master draait de kernel en doet scheduling
  - redelijk eenvoudig te implementeren (1 processor master over geheugen en I/O)
  - master kan wel bottleneck worden
- symmetric (SMP)
  - kernel kan worden uitgevoerd op elke processor
  - iedere processor doet eigen scheduling
  - ingrijpende wijzigingen in de kernel-code voor synchronisatie
    - verschillende processoren kunnen dezelfde code willen uitvoeren
    - verschillende processoren kunnen in hetzelfde deel van het geheugen lezen en schrijven
  - kernel gebouwd met meerdere processen/threads

Shared memory / Distributed memory

- Supercomputers met 1000's cpu's/cores (bv. Cray van ECMWF 260.000 cores!): geen shared memory architectuur, maar distributed memory architectuur (want connectie naar shared memory is bottleneck.



Inhoud

- Interprocess communication (IPC)
- Threads
- Threads in Linux
- Multi-processor
- Gelijktijdigheid
- Dead-lock
- Herhalingsvragen

Gelijktijdigheid vb.

```
#include<pthread.h>
pthread_t tid1, tid2;
char naam[10];

void* doThread1(void *arg)
{
  scanf("%s",naam);
  sleep(3);
  printf("%s\n",naam);
}
void* doThread2(void *arg)
{
  sleep(2);
  scanf("%s",naam);
  printf("%s\n",naam);
}
```

```
int main(void)
{
   pthread_create(&tid1, NULL, &doThread1, NULL);
   pthread_create(&tid2, NULL, &doThread2, NULL);
   pthread_join(tid1, NULL);
   pthread_join(tid2, NULL);
   return 0;
}
```

Gelijktijdigheid

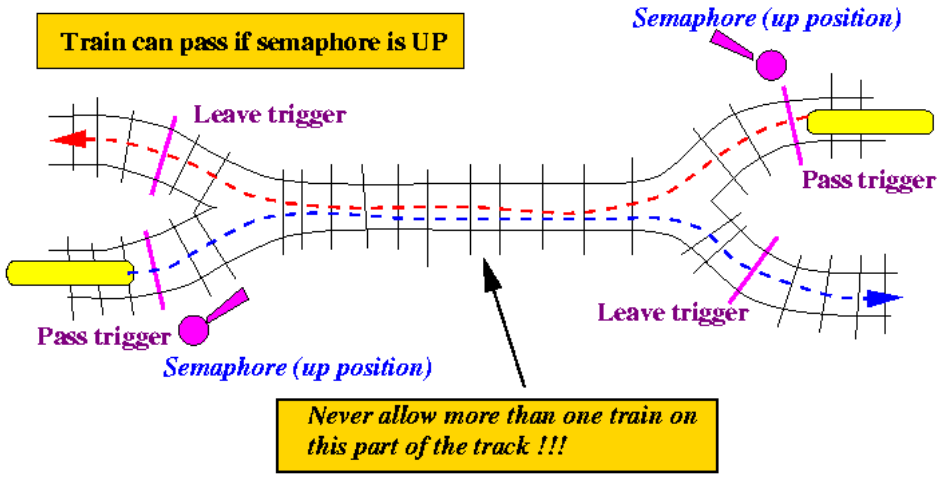
- meerdere processoren/threads = probleem met gelijktijdig aanspreken van bepaalde resources (bv. geheugen)
- sommige stukken code mogen maar door 1 proces/thread tegelijk worden uitgevoerd
  - = "critical section"
  - er moet een soort locking mogelijk zijn
  - deze locking moet light-weight zijn
- deze problematiek bestaat ook met 1 processor!!!

Gelijktijdigheid

- wait() en signal() zijn 'atomic operations'
  - geïmplementeerd en gegarandeerd door OS
  - "Semaforen"

0: sein neer (stop)
1: sein op (mag doorrijden)

Semaforen in Linux



- sem_wait()
  - Als 0: wacht
  - Als > 0 (sein op): doe semafoor -1 (sein neer) en ga door
- sem_post()
  - Doe semafoor +1 (sein op)
- sem_init()
  - Initialiseer semafoor
- sem_wait() en sem_post() zijn 'atomic operations'

Semaforen

```
#include<pthread.h>
#include <semaphore.h>
pthread_t tid1, tid2;
char naam[10];
sem_t sema;

void* doThread1(void *arg)
{
  sem_wait(&sema); /*semafoor NEER*/
  scanf("%s",naam);
  printf("%s\n",naam);
  sem_post(&sema); /*semafoor OP*/
}
```

```
void* doThread2(void *arg)
{
  sem_wait(&sema); /* semafoor NEER */
  scanf("%s",naam);
  printf("%s\n",naam);
  sem_post(&sema); /* semafoor OP */
}
int main(void)
{
    sem_init(&sema, 0, 1);  /* semafoor OP */
    pthread_create(&tid1, NULL, &doThread1, NULL);
    pthread_create(&tid2, NULL, &doThread2, NULL);
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    sem_destroy(&sema);
}
```

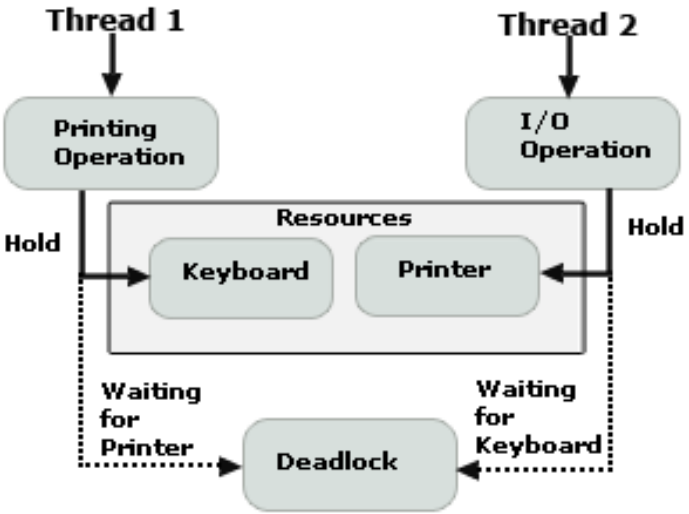
Inhoud

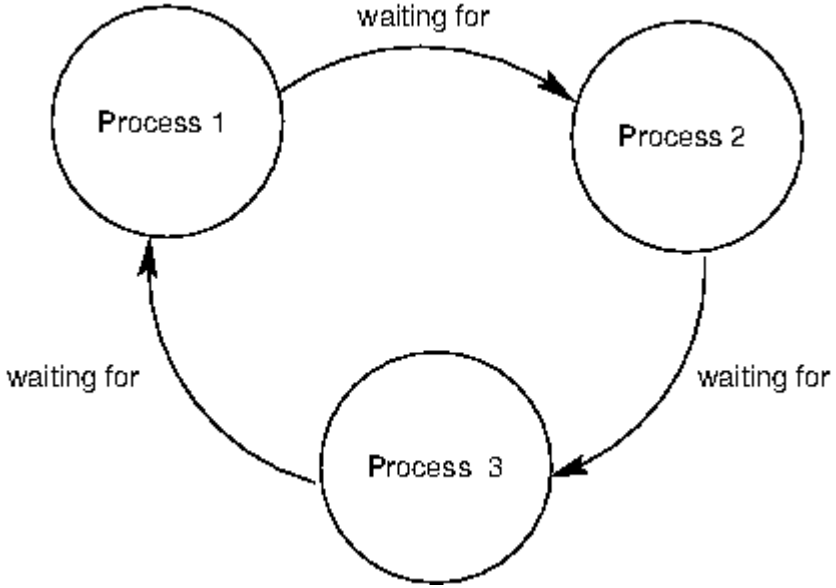
- Interprocess communication (IPC)
- Threads
- Threads in Linux
- Multi-processor
- Gelijktijdigheid
- Dead-lock
- Herhalingsvragen

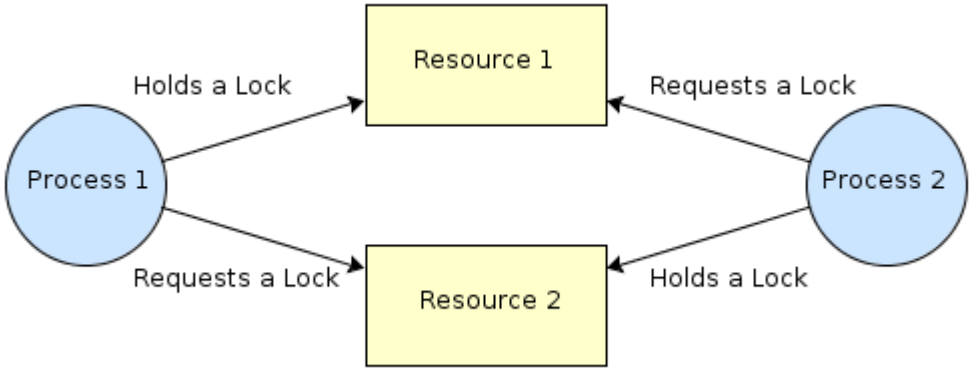
Deadlock

Wat is een deadlock?

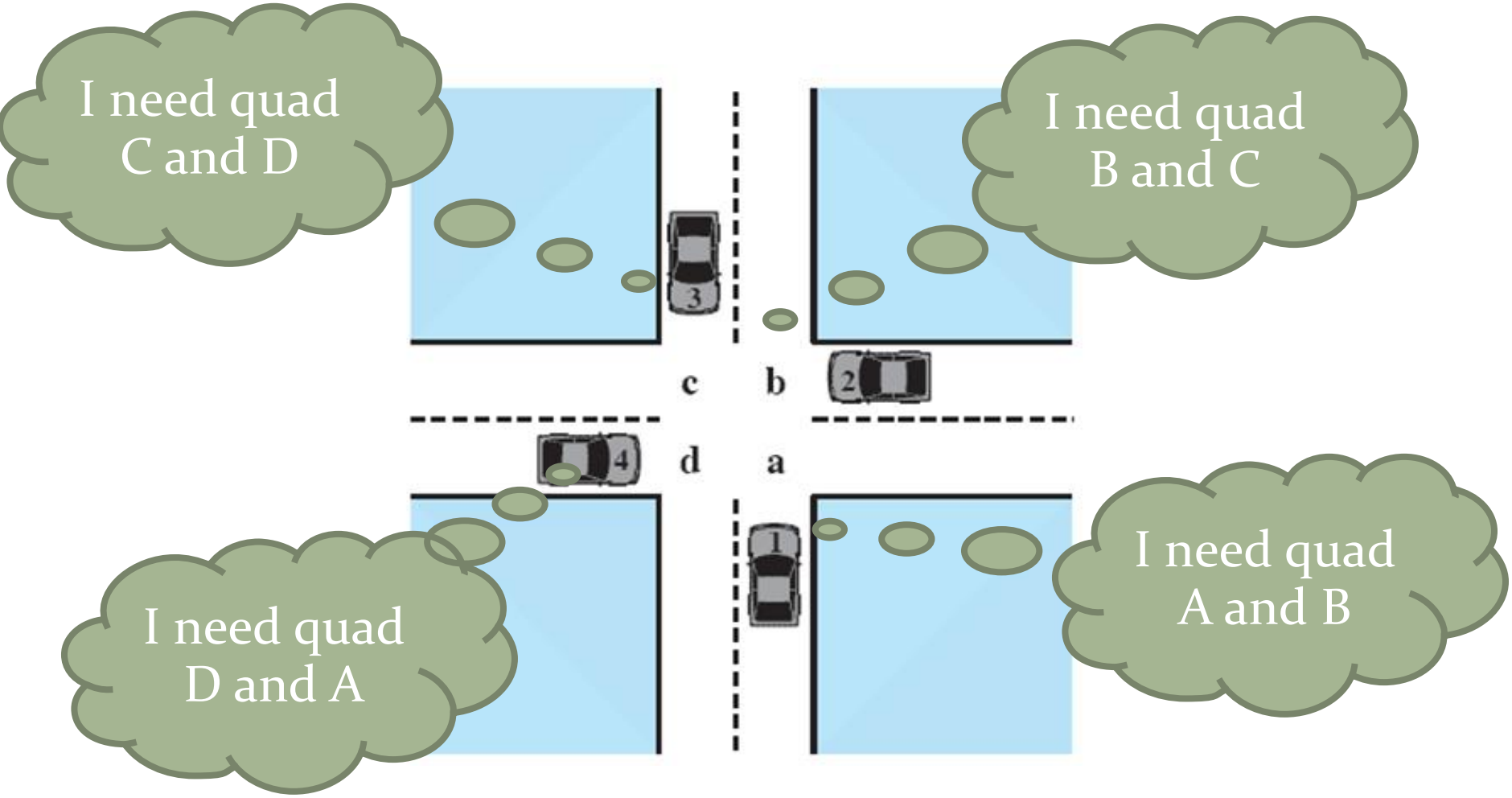
Deadlock


Thread 1
Thread 2
Printing Operation
I/O Operation
Resources
Hold
Hold
Keyboard
Printer
Waiting for Printer
Waiting for Keyboard
Deadlock

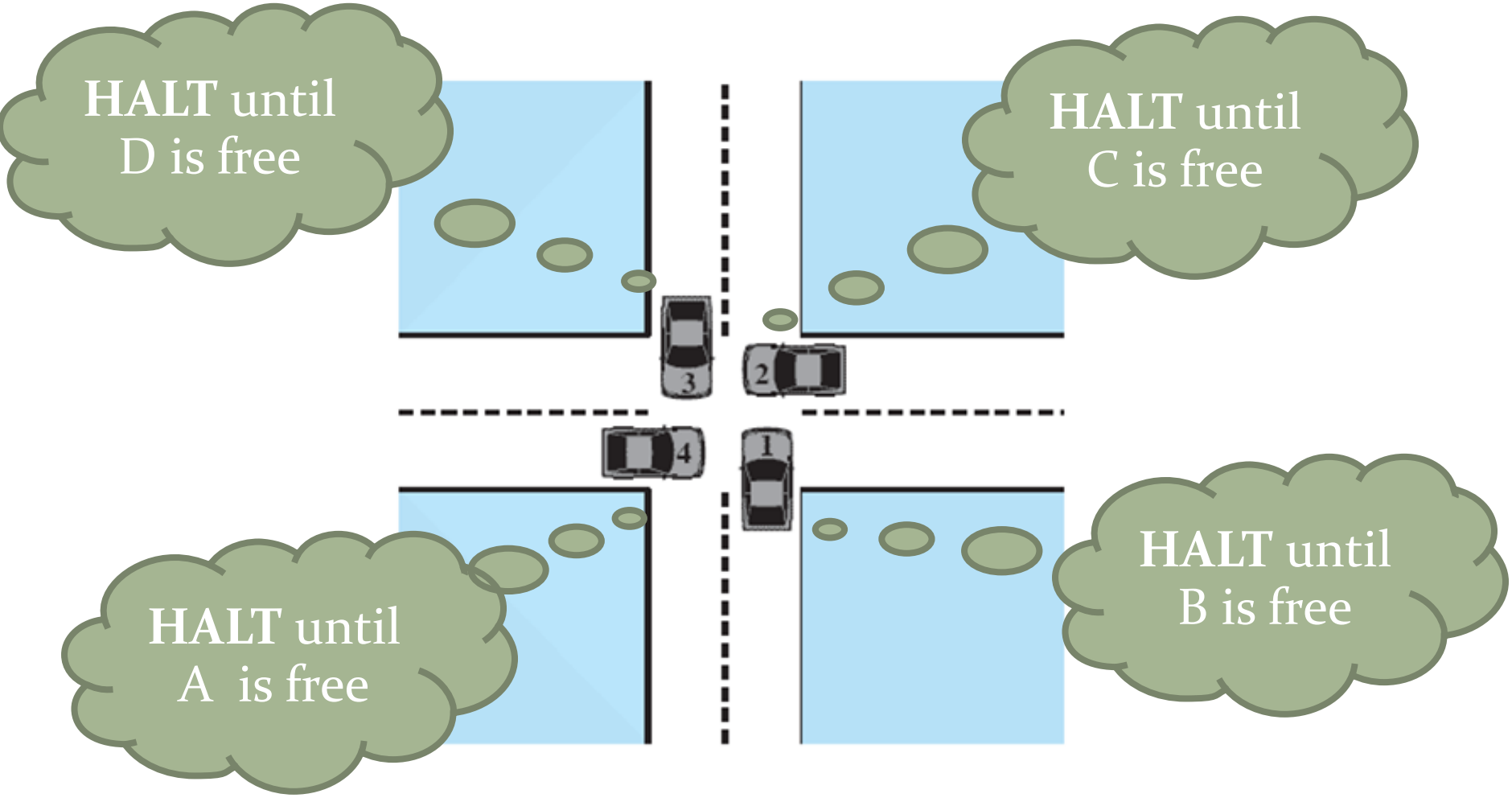

waiting for
Process 1
Process 2
waiting for
waiting for
Process 3


Resource 1
Holds a Lock
Requests a Lock
Process 1
Process 2
Requests a Lock
Holds a Lock
Resource 2

Deadlock


I need quad C and D
I need quad B and C
I need quad D and A
I need quad A and B
c
b
d
a
1
2
3
4

Deadlock


HALT until
D is free
HALT until
C is free
HALT until
A is free
HALT until
B is free
3
2
4
1

Deadlock

```
void* doThread1(void *arg)
{
    sem_wait(&sema1);
    printf("1 gelocked, nu nog 2...\n");
    sleep(1);
    sem_wait(&sema2);
    printf("1 en 2 gelocked!\n");
    sem_post(&sema2);
    sem_post(&sema1);
}
```

```
void* doThread2(void *arg)
{
  sem_wait(&sema2);
  printf("2 gelocked, nu nog 1...\n");
  sleep(1);
  sem_wait(&sema1);
  printf("2 en 1 gelocked!\n");
  sem_post(&sema1);
  sem_post(&sema2);
}
```

Inhoud

- Interprocess communication (IPC)
- Threads
- Threads in Linux
- Multi-processor
- Gelijktijdigheid
- Dead-lock
- Herhalingsvragen

Herhalingsvragen

- Wat zijn pipes?
- Welke stappen moet je doorlopen om een bericht van een proces naar een ander te sturen in Unix?
- Welke stappen moet je doorlopen om een shared memory segment op te zetten in Unix?
- Hoe kan het OS een deel geheugen delen tussen verschillende processen?
- Welke geheugensegmenten worden gedeeld door threads?
- Wat is het verschil tussen een proces en een thread?
- Wat zijn de voor- en nadelen van threads?
- Wat is het verschil tussen een user-level thread en een kernel-level thread? Wat zijn de voor- en nadelen?
- Waarom worden lokale variabelen niet gedeeld tussen threads en globale wel?
- Wat is load-balancing in een multiprocessor?
- Teken de architectuur van een multiprocessor machine
- Wat is het verschil tussen een master-slave en een SMP multiprocessor OS?
- Wat is een 'critical section'?
- Leg uit wat een semafoor is. Welke operaties kan je erop uitvoeren en wat doen deze?
- Leg uit wat deadlocks zijn. Wanneer treden ze op?