IN1805 I – Operating System Concepten

Hoofdstuk 3: Processes

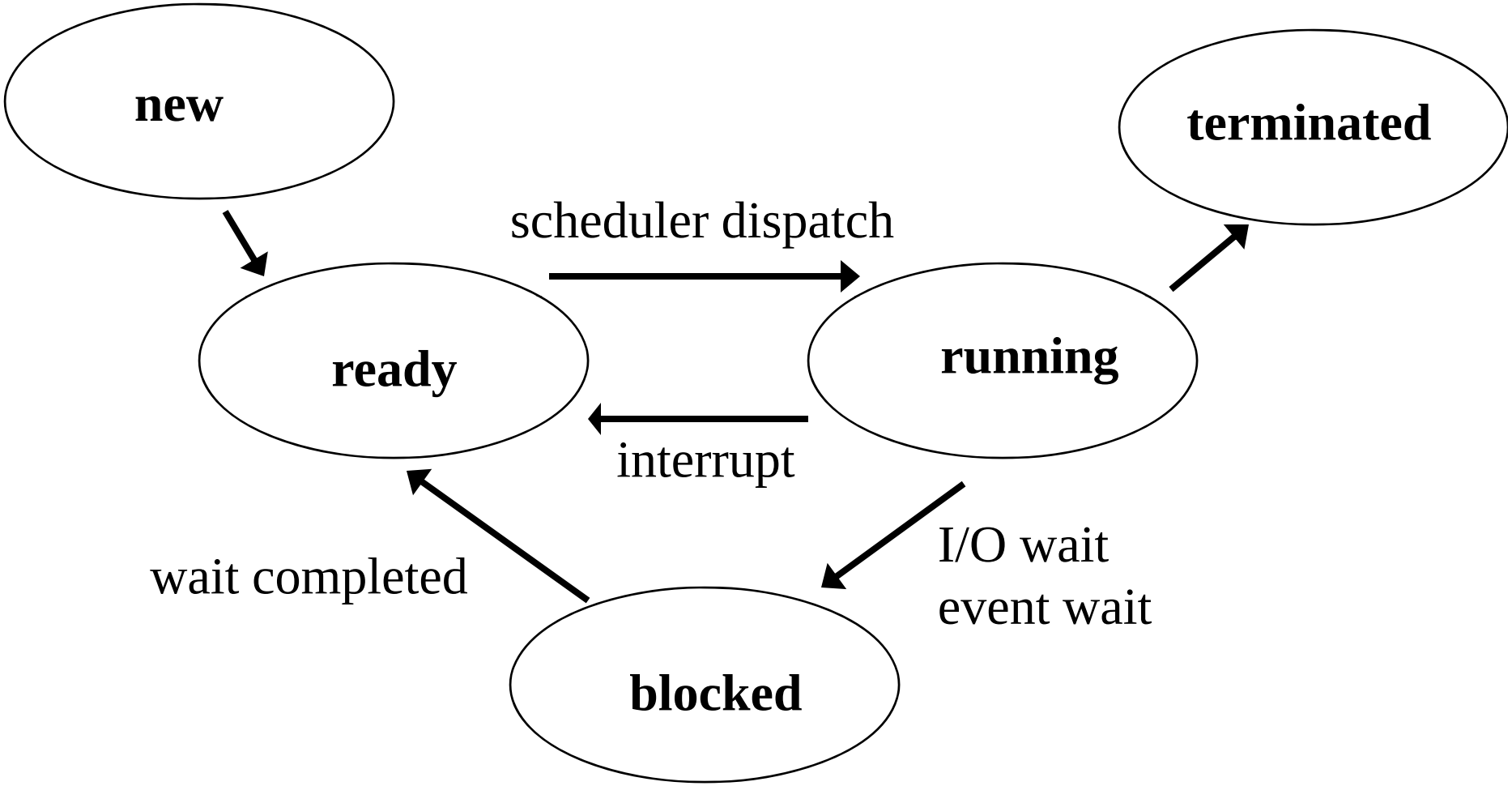
Het begrip 'Proces'

- Een proces is de uitvoering van een programma
- Bij een proces hoort
  - een programma (de code)
  - Program Counter (Instructiewijzer)
  - stack
  - data (data sectie)
- twee of meer processen kunnen hetzelfde programma uitvoeren
- processen kunnen nieuwe processen creeren
- onderscheid systeemprocessen en user processen

Proces-toestand

- Mogelijke proces-toestanden:
  - nieuw
  - running (heeft de CPU)
  - waiting (wacht op iets anders dan de CPU, **blocked**)
  - ready (is klaar om de CPU te gebruiken)
  - beeindigd
- In een systeem met 1 processor is hoogstens 1 proces running
- veel processen zullen waiting (blocked) zijn
- veel processen zullen ready zijn

Proces-toestanden



Process Control Block (PCB)

- Een PCB is de representatie van een proces, bevat o.a.
  - proces toestand
  - waarde program counter bij laatste interrupt
  - inhoud registers bij laatste interrupt
  - accounting informatie
  - scheduling informatie
- PCB's vormen queues d.m.v. linked lists

Queues en Schedulers (1)

- Queues:
  - job queue (alleen bij batch systemen)
  - ready queue
  - I/O queues
  - andere blocked queues
- Proces verhuist voortdurend van queue.
- Het selecteren van processen gebeurt door schedulers

Queues en Schedulers (2)

- Schedulers:
  - Long term scheduler (bij batch systemen) welke job uit de job queue wordt in behandeling genomen?
  - Short term scheduler : welk proces uit de Ready queue krijgt de CPU ?
  - Medium term scheduler: besluit over in- en uitswappen, doel aanpassen vam Multiprogrammeringslevel (MPL)

Context Switch

- Context switch, switcht CPU van 1 proces naar ander proces
- redt status van oude proces, herstelt status nieuwe proces
- overhead
- hardware support

Proces-creatie (1)

- Proces kan nieuwe processen creeren
  - creator: parent (ouderproces)
  - nieuwe proces: child (kindproces)
- Child kan zelf ook weer processen creeren, etc.
- Procesboom
- Mogelijkheden m.b.t. executie
  - parent loopt concurrent met child, of
  - parent wacht (blocked) tot laatste child geeindigd
- Mogelijkheden m.b.t geheugenruimte
  - child heeft duplicaat van ouder, of
  - child heeft eigen programma

Proces-creatie (2)

Voorbeeld: UNIX

- proces heeft procesid (integer)
- creatie nieuw proces d.m.v. **fork**
- programma (code) kind is copie van ouder
- verschil:
  - kind ziet als resultaat van **fork** 0
  - ouder ziet als resultaat van **fork** de procesid van het kind
- ouder en kind lopen concurrent
- ouder kan wachten op eindigen van kind door **wait**

proces creatie (2a)

UNIX Programma voorbeeld

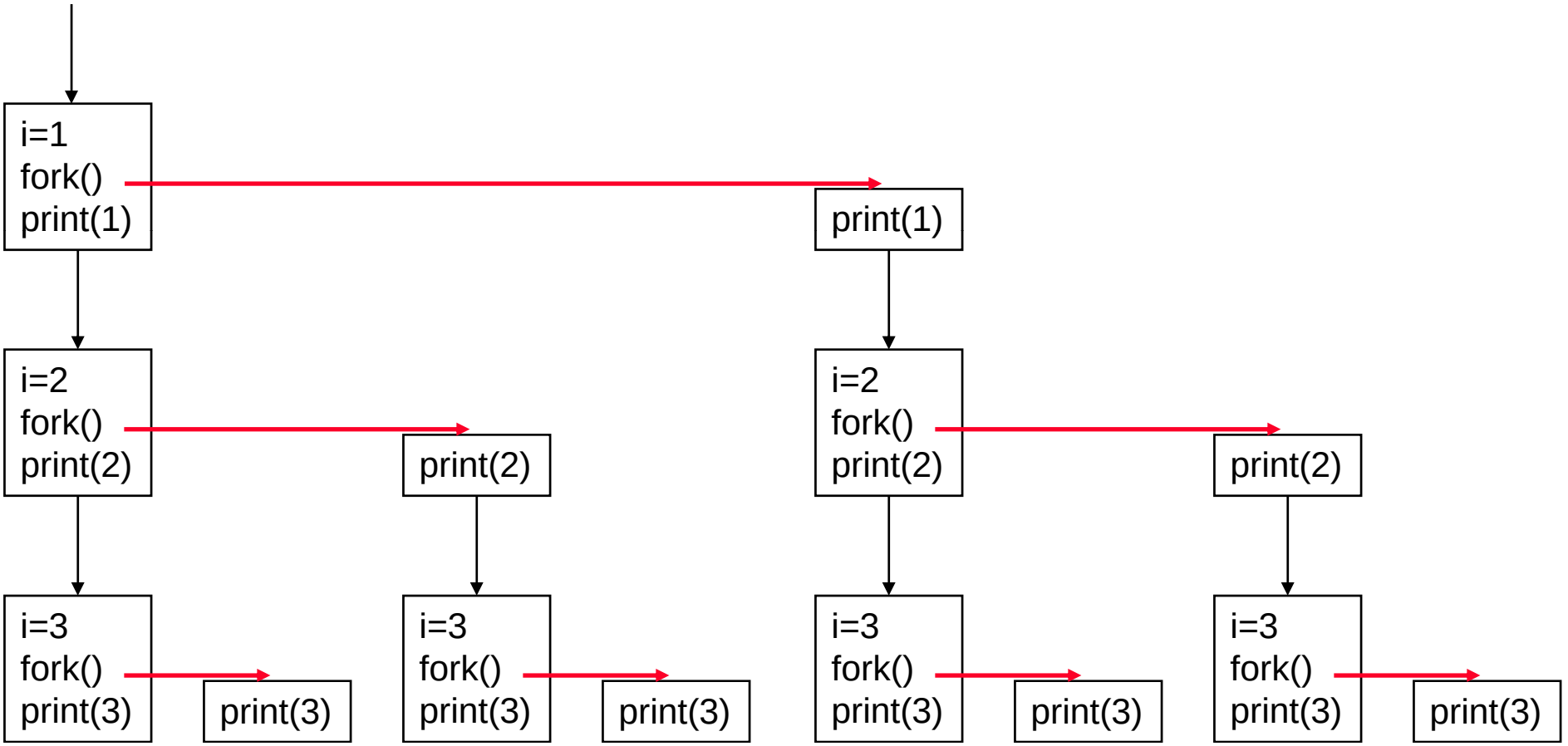
```
int pidval, statchild = 0;  int *statloc;
. . .
pidval = fork();
if (pidval == 0) {                  /* I am the child */
    execvp(progname, args)   /* load a new program */
}
else {                                  /* I am the parent */
    statloc = &statchild;          /* statloc points to location where
                                        the child status has to be reported  */
    wait( statloc);
}
exit (0);
```

Oefening fork1

```
int main()
{ int  pidval;
  int  i;
  for (i = 1; i <= 3; i++ ){
          pidval = fork();
          printf(“i = %d\n", i);
       }
}
```

Vraag: Als dit programma wordt uitgevoerd, wat is dan het resultaat (de printuitvoer) ?

Oefening fork1: procesboom



Oefening fork1: antwoord

```
37 duticai /home/heijnsd/testfork: fork1
i=1
i=2
i=2
i=3
i=3
i=1
i=3
i=2
i=2
i=3
i=3
i=3
i=3
i=3
38 duticai /home/heijnsd/testfork:
```

LET OP: andere volgordes ook mogelijk!

Proces-beëindiging

Voorbeeld: UNIX (vervolg)

- Een proces eindigt door d.m.v. **exit**
- ouder kan kind beeindigen, b.v. door **kill ,**

  reden hiervoor b.v.
  - kind heeft zijn resources verbruikt (b.v. maximale CPUtijd)
  - taak van het kind is niet langer nodig
  - ouder wil eindigen, maar OS verbiedt dat kind blijft bestaan zonder ouder.

Concurrent Processes

Concurrent = samenlopen in de tijd

Concurrent processen kunnen zijn:

- **independent**, zo'n proces kan dan niet door andere processen worden beinvloed
- **cooperative**, zo'n proces kan wel worden beinvloed door andere processen

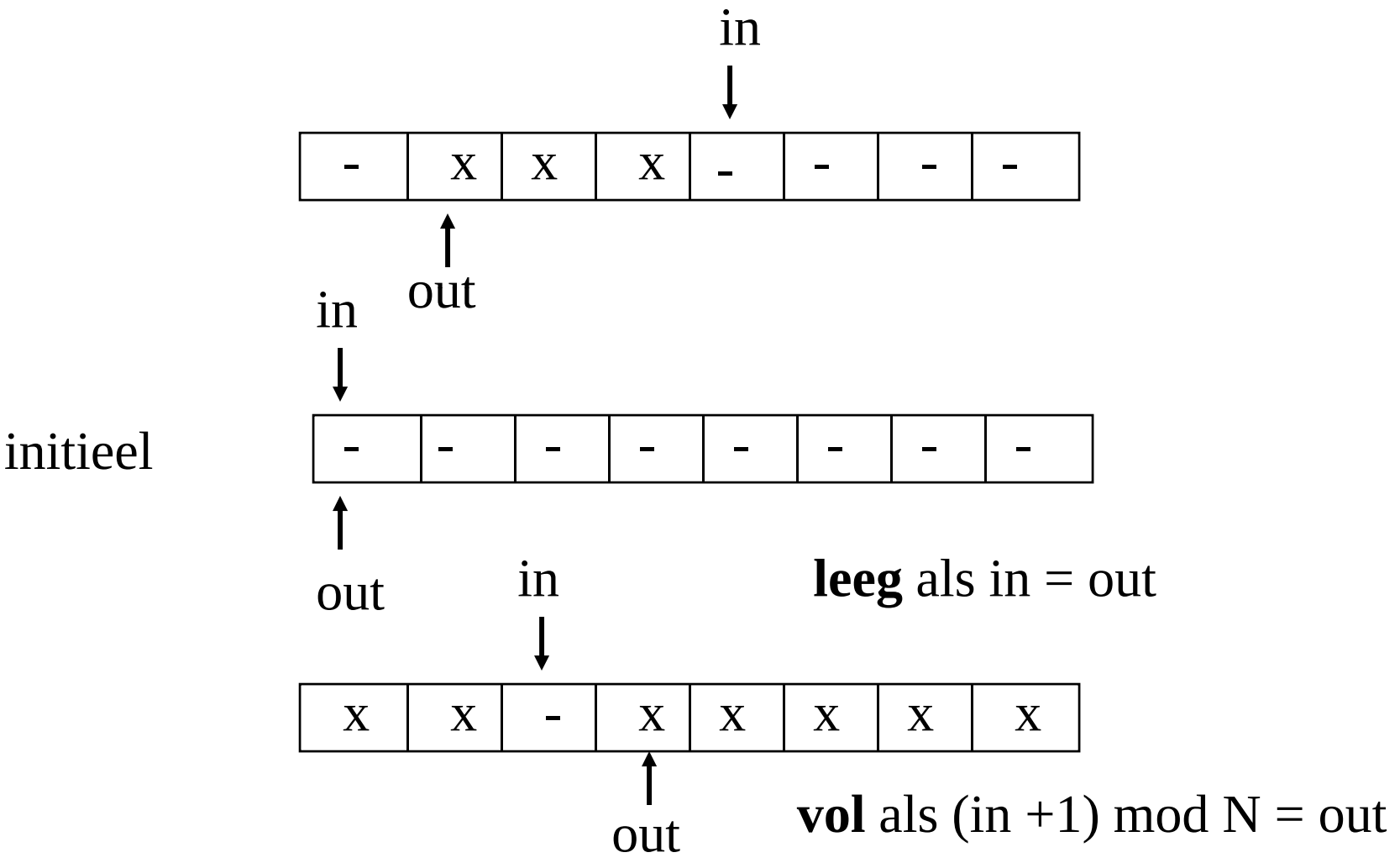
PAUZE

Inter Process Communication (IPC)

Twee modellen:

- Shared Memory
  - Het snelst
  - Voornamelijk binnen één systeem
- Message Passing
  - Het eenvoudigst
  - Ook geschikt tussen verschillende systemen

Shared Memory (1)
Producer-Consumer probleem, met bounded buffer



Shared Memory (2)
Consumer probleem met bounded buffer

Een oplossing:

shared

```
#define BUFFER_SIZE 10
typedef struct { ....} item;
item buffer[BUFFER_SIZE];
int in=0, out=0;
```

producer

```
while (1) {
  /* produce item in nextProduced */
  while(((in + 1)%BUFFER_SIZE)== out)
     ; /* do nothing */
  buffer[in] = nextProduced;
  in = (in + 1) % BUFFER_SIZE;
}
```

consumer

```
while (1) {
  while (in == out)
    ;  /* do nothing */
  nextConsumed = buffer[out];
  out = (out + 1) % BUFFER_SIZE;
  /* consume the item
              in nextConsumed */
}
```

Message Passing (1)

- primitieven:
  - write(to, message)
  - read(from, message)
- aspecten van message passing :
  - synchroon/asynchroon (blocking/nonblocking)
  - adressering: direct/indirect
  - buffering (kanaalcapaciteit)

Message Passing (2)
synchroon/asynchroon

- Algemeen:
  - Synchroon = blocking,
  - asynchroon = nonblocking
- Mogelijkheden:
  - blocking send
  - nonblocking send
  - blocking receive
  - nonblocking receive
- Combinaties zijn mogelijk.
- blocking is makkelijker voor programmeur
- nonblocking verkort de responstijd

Message Passing (3)
directe /indirecte communicatie

- Directe communicatie:
  - symmetrisch:
    - send(P, message) *zend bericht naar P*
    - receive(Q, message) *ontvang een bericht van Q*
  - asymmetrisch:
    - send(P, message) *zend bericht naar P*
    - receive(id, message) *ontvang een bericht, id wordt ingevuld bij ontvangst*
- Indirecte communicatie: (via een postbus)
    - send(X, message) *zend bericht naar postbus X*
    - receive(X, message) *ontvang bericht van postbus X*

  vraag: Wie is de eigenaar van de postbus? Systeem of gebruiker?

Message Passing (4)
buffering

Buffering : bepaalt het aantal berichten dat onderweg kan zijn (queue van messages), mogelijkheden:

- Zero Capacity:
  - max queue lengte is 0,
  - zender moet wachten tot het bericht is ontvangen.
- Bounded Capacity:
  - queue heeft beperkte lengte n,
  - zender moet wachten als de queue vol is.
- Unbounded Capacity
  - queue is in principe oneindig groot

Message Passing (5)
voorbeelden:

- via een file
  - eenvoudig, asynchroon, capaciteit is file size, adressering indirect
- via een pipe
  - vergelijkbaar met file, maar messages blijven in het geheugen
  - (gebeurt o.a. bij UNIX wanneer de gebruiker via de shell een pipe definieert. b.v. who | sort , geeft gesorteerde lijst van ingelogde gebruikers)
- via sockets
  - universeel, ook tussen processen in verschillende systemen
  - synchroon of asynchroon, capaciteit bepaald door de verbinding
  - adressering direct

IN1805 I – Operating System Concepten

Hoofdstuk 4: Threads

Threads(1)

- Thread = Light Weight Process (LWP)
- Binnen een proces meerdere threads mogelijk
- Threads hebben eigen Thread-ID, Program Counter, register set, stack
- Threads hebben geen eigen geheugen, daardoor geen bescherming tegen elkaar
- voorbeeld: File Server
- voordelen:
  - Meer activiteiten simultaan (b.v. display en datacommunicatie)
  - betere benutting resources
  - sneller switchen
  - bij geschikt support meer profijt van multiprocessor

Threads(2)

Support op verschillende wijze mogelijk

- user-level support d.m.v. thread library (user threads)
  - onafhankelijk van OS
  - snel te creëren en managen
  - OS ziet geheel als één proces
  - verdeling CPUtijd (door OS) op basis van processen
  - voordeel: sneller switchen
  - nadeel: 1 thread kan alle andere ophouden (b.v. bij System Call)
  - Applicatie beter portable dan bij kernel threads
- door kernel (kernel threads)
  - minder snel dan user threads
  - verdeling van CPUtijd (door OS) op basis van threads
- beide

Threads (3)

Multithreading modellen.

Algemeen:

- Op user niveau zoveel threads te definiëren als nuttig lijkt
- Processor alleen toe te wijzen op basis van kernel threads

Relatie tussen user threads en kernel threads

- Many to One
  - Efficient threads creëren, en switchen
  - geen voordeel van multiprocessor
- One to one
  - multiprocessor biedt voordelen
  - Creëren van kernel threads is ‘duur’
- Many to Many
  - meest flexibel: voordelig bij multiprocessors, aantal kernel threads beperkt