TDIU16 – Semaforen

Om att vänta – men inte i onödan

Filip Strömbäck, Klas Arvidsson

LINKÖPING UNIVERSITY

1 Kursinformation
2 Varför vänta?
3 Hur ska vi vänta?
4 Semaforen
5 Använda semaforer

LINKÖPING UNIVERSITY

Planering

Vecka	Fö/Se	Lab	Deadline
14	Fö: C + Syscall (+påsk)	1ab: C[1], 1c: intro	–
15	–	2: Systemanrop	–
16	Fö + Se: Semaforen	3ab: Processhantering	1, 2
17	Fö: Lås, cond, deadlock	3cd: Processhantering	–
18	Se: Synkronisering	4ab: Synkronisering	3
19	Fö: Låsimplementation	4bc: Synkronisering	–
20	–	4: Synkronisering, 5: Säkerhet	–
21	–	5: Säkerhet	4, 5

[1] lämpligt att demonstrera första passet

Moment i morgon

13:15: **Seminarie lab 3**

- 4 pass, 2 för Di/EL, 2 för IP
- Välj pass i mån av plats

15:15: **Lab, genomgång av lab 3a**

- Di/EL: SU15/16 + SU17/18. Genomgång i SU17/18, börja där
- IP: Glas – dvs. *inte* i IP-salar som vanligt
- Notera: endast demo av 3a (fristående), och ev 3b

1 Kursinformation
2 Varför vänta?
3 Hur ska vi vänta?
4 Semaforen
5 Använda semaforer

LINKÖPING UNIVERSITY

Hur körs ett trådat program?

Operativsystemets syn:

- Trådar körs "parallellt" med varandra
- Använder time-sharing och/eller flera CPU-kärnor

$\Rightarrow$ Olika trådar kör "en bit" i taget

Kompilatorns syn:

- All kod är enkeltrådad om vi inte anger något annat

$\Rightarrow$ Gör att kompilatorn kan generera effektiv kod

Exempel: Hur körs ett trådat program? (1)

```
int result;

void thread_fn() {
  for (int i = 0; i < 100000; i++)
    result += 2;
}

int main(void) {
  thread_new(&thread_fn);
  printf("result=%d\n", result);
  return 0;
}
```

Exempel: Hur körs ett trådat program? (2, EJ OK)

```
int result;
bool done;

void thread_fn() {
  for (int i = 0; i < 100000; i++)
    result += 2;
  done = true;
}
```

```
int main(void) {
  thread_new(&thread_fn);
  while (!done)
    ;
  printf("result=%d\n", result);
  return 0;
}
```

Exempel: Hur körs ett trådat program? (2, EJ OK)

```
int result;
bool done;

void thread_fn() {
  for (int i = 0; i < 100000; i++)
    result += 2;
  done = true;
}
```

```
int main(void) {
  thread_new(&thread_fn);
  while (!done)
    ;
  printf("result=%d\n", result);
  return 0;
}
```

Varför är detta problematiskt?

Exempel: Hur resonerar kompilatorn?

Vad blir `a` och `b` nedan?

A

```
a = 10;
b = 20;
a = a * 5;
b = a + b;
```

B

```
a = 10;
a = a * 5;
b = 20;
b = a + b;
```

C

```
a = 50;
b = 70;
```

Exempel: Hur resonerar kompilatorn?

Vad blir `a` och `b` nedan?

A

```
a = 10;
b = 20;
a = a * 5;
b = a + b;
```

B

```
a = 10;
a = a * 5;
b = 20;
b = a + b;
```

C

```
a = 50;
b = 70;
```

Gäller detta om vi introducerar trådar?

Exempel: Hur resonerar kompilatorn?

Är dessa ekvivalenta?

```
while (!done)
  ;
```

```
if (!done) {
  while (true)
    ;
}
```

1 Kursinformation
2 Varför vänta?
3 Hur ska vi vänta?
4 Semaforen
5 Använda semaforer

LINKÖPING UNIVERSITY

Ett järnvägsproblem

- Järnvägsknut med två spår österut, ett västerut
- Ett X2000 kommer västerifrån och fortsätter mot sydost klockan 16:10 enligt tidtabell
- Ett godståg kommer från nordost och fortsätter västerut

Du är lokförare på godståget. Klockan är 16:15 när du är framme vid knuten. Vad gör du när du kommer fram? Kör? Väntar? Hur länge?

Bron runt en bergstopp

En liten gångbro går runt en bergstopp

- Bron är lång, smal, och inte särskilt stadig
- Du kan inte se hela bron på samma gång
- Bron kan bära max 5 personer åt gången

Går du ut på bron? Väntar du? Hur länge?

Dörrvakten

- Har order uppifrån pga brandregler:
  - Släpp in max N personer
- Garanterar att det aldrig är fler i lokalen än order
- Kommer fler måste de vänta tills det finns plats
- Måste hålla koll på både hur många som kommer och hur många som går

1 Kursinformation
2 Varför vänta?
3 Hur ska vi vänta?
4 Semaforen
5 Använda semaforer

LINKÖPING UNIVERSITY

Semaforen

- Har order från programmeraren pga resursbegränsning eller krävd händelseordning:
  - Släpp in max N trådar
- Garanterar att det aldrig är fler insläppta än order
- Kommer fler trådar måste de vänta tills det finns plats
- Anropas både vid insläpp (`down`) och utsläpp (`up`)

Semaforen i Pintos

```
#include "threads/sync.h"

struct semaphore sema;

// Initiera semaforen till N resurser
sema_init(&sema, N);

// Försök räkna ner, väntar kanske
sema_down(&sema);

// Räkna upp, väcker trådar som väntar
sema_up(&sema);
```

Andra funktionsnamn

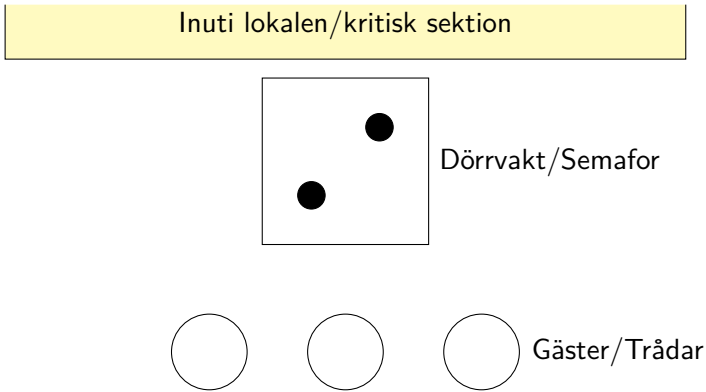
- För att räkna ner eller vänta

`P()`	Proberen, ursprungligt
`Wait()`	Mer beskrivande
`Down()`	Pintos, bättre?

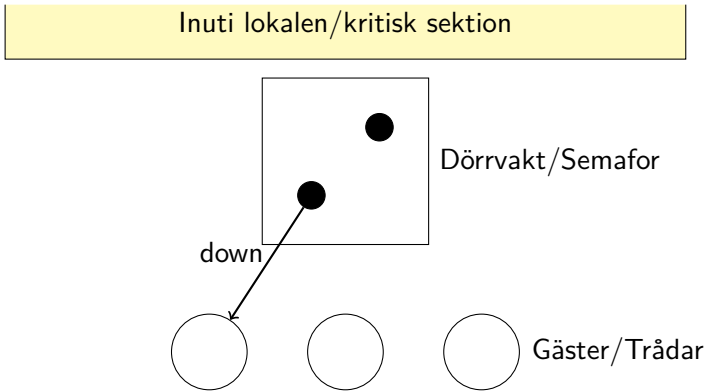
- För att räkna upp och signalera

`V()`	Verhogen, ursprungligt
`Signal()`	Mer beskrivande
`Up()`	Pintos, bättre?

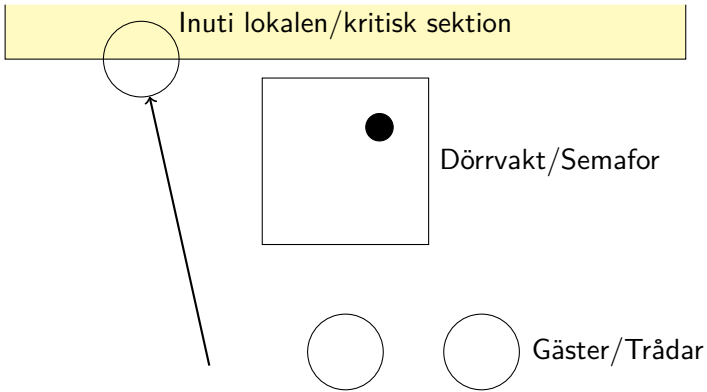
Exempel:


Inuti lokalen/kritisk sektion
Dörrvakt/Semafor
Gäster/Trådar

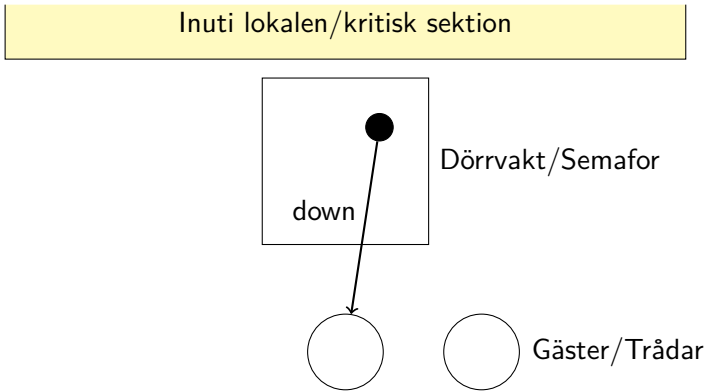
Exempel:


Inuti lokalen/kritisk sektion
Dörrvakt/Semafor
down
Gäster/Trådar

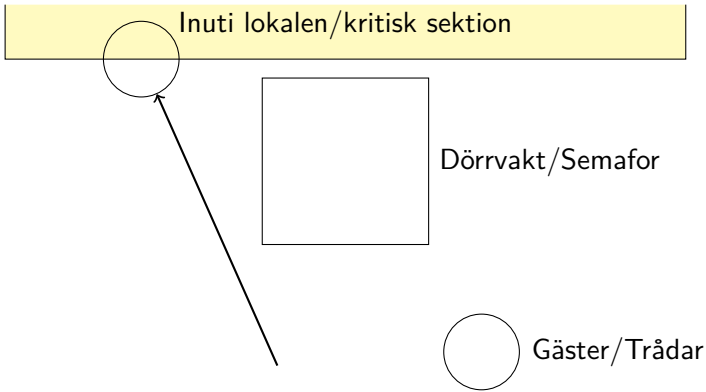
Exempel:


Inuti lokalen/kritisk sektion
Dörrvakt/Semafor
Gäster/Trådar

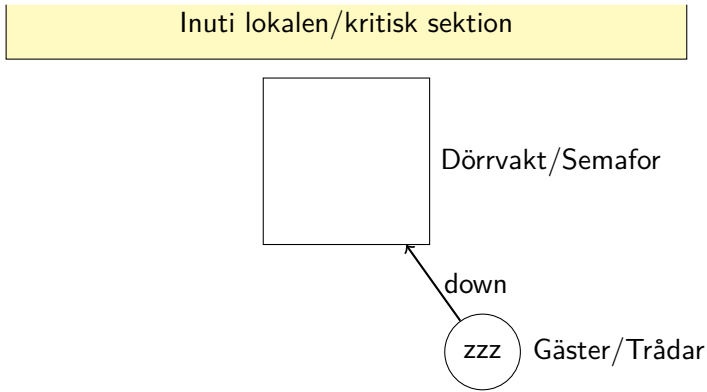
Exempel:


Inuti lokalen/kritisk sektion
Dörrvakt/Semafor
down
Gäster/Trådar

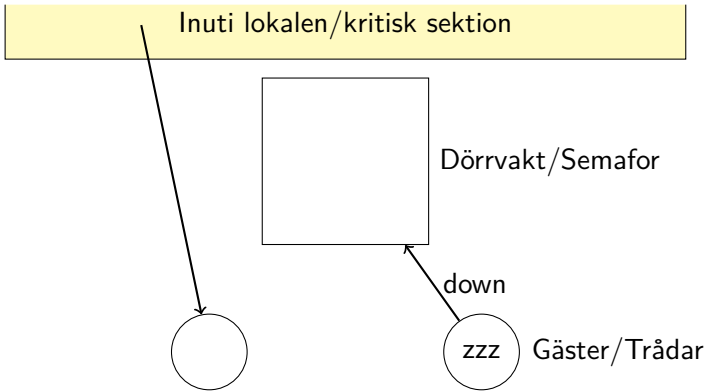
Exempel:


Inuti lokalen/kritisk sektion
Dörrvakt/Semafor
Gäster/Trådar

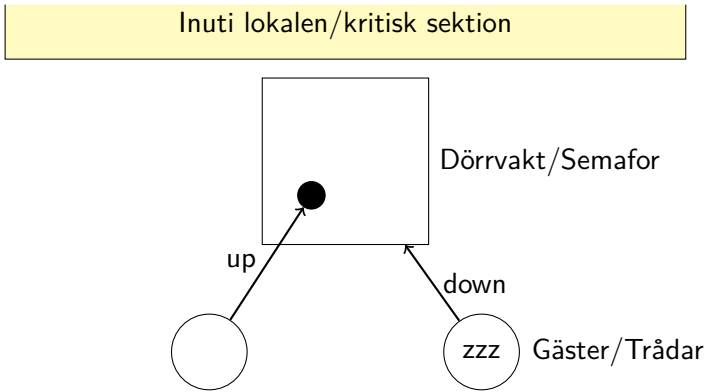
Exempel:


Inuti lokalen/kritisk sektion
Dörrvakt/Semafor
down
zzz
Gäster/Trådar

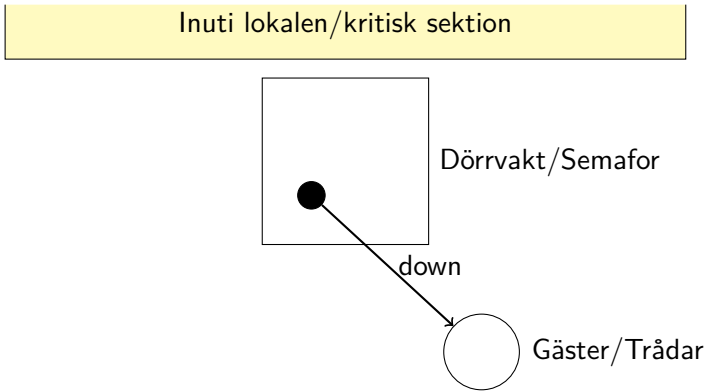
Exempel:


Inuti lokalen/kritisk sektion
Dörrvakt/Semafor
down
zzz
Gäster/Trådar

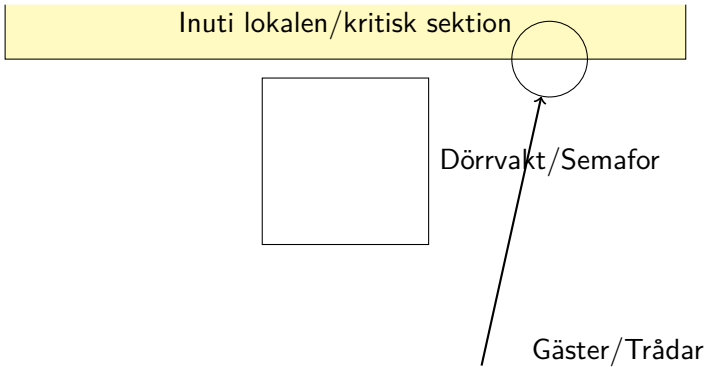
Exempel:


Inuti lokalen/kritisk sektion
Dörrvakt/Semafor
up
down
zzz
Gäster/Trådar

Exempel:


Inuti lokalen/kritisk sektion
Dörrvakt/Semafor
down
Gäster/Trådar

Exempel:


Inuti lokalen/kritisk sektion
Dörrvakt/Semafor
Gäster/Trådar

1 Kursinformation
2 Varför vänta?
3 Hur ska vi vänta?
4 Semaforen
5 Använda semaforer

LINKÖPING UNIVERSITY

Hur kan man tänka?

- Semaforen räknar *hur många* **tillgängliga** resurser som finns.
  - Finns det plats på spåret västerut?
  - Hur många personer till får plats på bron/i lokalen?
- Kan *starta* i vilket läge som helst.
- Kan sedan bara modifieras med `sema_up` och `sema_down`.
- Kontroll: vi väntar då semaforen är 0 och vi kör `sema_down` (= när inga resurser är tillgängliga och vi behöver en till)!

Synkronisering med semafor (OK)

```
int result;
struct semaphore has_result;

void thread_fn() {
  for (int i = 0; i < 100000; i++)
    result += 2;
  sema_up(&has_result);
}
```

```
int main(void) {
  sema_init(&has_result, 0);
  thread_new(&thread_fn);
  sema_down(&has_result);
  printf("result=%d\n", result);
  return 0;
}
```

Behöver vi två semaforer?

```
int result_a;
int result_b;

struct semaphore has_a;
struct semaphore has_b;
```

```
void thread_fn_a(void) {
  for (int i = 0; i < 100000; i++)
    result_a += 2;
  sema_up(&has_a);
}

void thread_fn_b(void) {
  for (int i = 0; i < 100000; i++)
    result_b += 2;
  sema_up(&has_b);
}
```

Behöver vi två semaforer?

```
void thread_fn_a(void) {
  for (int i = 0; i < 100000; i++)
    result_a += 2;
  sema_up(&has_a);
}

void thread_fn_b(void) {
  for (int i = 0; i < 100000; i++)
    result_b += 2;
  sema_up(&has_b);
}
```

```
int main(void) {
  sema_init(&has_a, 0);
  sema_init(&has_b, 0);
  thread_new(&thread_fn_a);
  thread_new(&thread_fn_b);
  sema_down(&has_a);
  printf("a=%d\n", result_a);
  sema_down(&has_b);
  printf("b=%d\n", result_b);
  return 0;
}
```

Behöver vi två semaforer?

```
void thread_fn_a(void) {
  for (int i = 0; i < 100000; i++)
    result_a += 2;
  sema_up(&has_ab);
}

void thread_fn_b(void) {
  for (int i = 0; i < 100000; i++)
    result_b += 2;
  sema_up(&has_ab);
}
```

```
int main(void) {
  sema_init(&has_ab, 0);
  thread_new(&thread_fn_a);
  thread_new(&thread_fn_b);
  sema_down(&has_ab);
  printf("a=%d\n", result_a);
  sema_down(&has_ab);
  printf("b=%d\n", result_b);
  return 0;
}
```

Behöver vi två semaforer?

```
void thread_fn_a(void) {
  for (int i = 0; i < 100000; i++)
    result_a += 2;
  sema_up(&has_ab);
}

void thread_fn_b(void) {
  for (int i = 0; i < 100000; i++)
    result_b += 2;
  sema_up(&has_ab);
}
```

```
int main(void) {
  sema_init(&has_ab, 0);
  thread_new(&thread_fn_a);
  thread_new(&thread_fn_b);
  sema_down(&has_ab);
  sema_down(&has_ab);
  printf("a=%d\n", result_a);
  printf("b=%d\n", result_b);
  return 0;
}
```

Bounded Buffer

- En applikation som ska skyffla data mellan två (eller fler) nätverkskort
  - En tråd tar emot data och lägger till i en kö
  - En tråd läser data från kön och skickar vidare
- Kön (buffer) har begränsad (bounded) storlek
  - Vad gör vi om kön är full när vi skall lägga till mer data?
  - Vad gör vi om kön är tom när vi ska läsa data?

Bounded Buffer

```
typedef unsigned char byte;
const int SIZE = 256;

struct Buffer {
  int buffer[SIZE];
  byte rpos = 0, wpos = 0;
  int free = SIZE;
};

int  get(struct Buffer *b);
void put(struct Buffer *b, int data);
```

Bounded Buffer (EJ KOMPLETT)

```
int get(struct Buffer *b) {
  if (b->free == SIZE)
    /* Vad gör vi här? */;
  ++b->free;
  return b->buffer[b->rpos++];
}
void put(struct Buffer *b, int data) {
  if (b->free == 0)
    /* Vad gör vi här? */;
  --b->free;
  b->buffer[b->wpos++] = data;
}
```

Bounded Buffer (EJ OK)

```
int get(struct Buffer *b) {
  while (b->free == SIZE)
    ;
  ++b->free;
  return b->buffer[b->rpos++];
}
void put(struct Buffer *b, int data) {
  while (b->free == 0)
    ;
  --b->free;
  b->buffer[b->wpos++] = data;
}
```

Bounded Buffer (OK)

```
typedef unsigned char byte;
const int SIZE = 256;

struct Buffer {
  int buffer[SIZE];
  byte rpos = 0, wpos = 0;
  struct semaphore free = SIZE;
  struct semaphore filled = 0;
};

int  get(struct Buffer *b);
void put(struct Buffer *b, int data);
```

Bounded Buffer (OK)

```
void get(struct Buffer *b) {
  sema_down(&b->filled);
  int r = b->buffer[b->rpos++];
  sema_up(&b->free);
  return r;
}
void put(struct Buffer *b, int data) {
  sema_down(&b->free);
  b->buffer[b->wpos++] = data;
  sema_up(&b->filled);
}
```

Flera trådar?

- Fortfarande inte OK om mer än en tråd läser eller mer än en tråd skriver. Varför?
  - Flera olika trådar som kör samtidigt
  - Gemensam data som används samtidigt
- Vad händer när olika trådar samtidigt modifierar gemensam data?
  - Fundera på `i++`
  - Fundera på ordningen av de två sista raderna i *Bounded buffer (EJ OK)*.
- Mer på nästa föreläsning!

Vad händer här? (Nästa föreläsning)

```c
int result;
struct semaphore has_result;

void thread_fn() {
  for (int i = 0; i < 100000; i++)
    result += 2;
  sema_up(&has_result);
}
```

```c
int main(void) {
  sema_init(&has_result, 0);
  thread_new(&thread_fn);
  for (int i = 0; i < 100000; i++)
    result += 5;
  sema_down(&has_result);
  printf("result=%d\n", result);
  return 0;
}
```

Filip Strömbäck, Klas Arvidsson

www.liu.se

LINKÖPING UNIVERSITY