

TDIU16 – Semaforen

Om att vänta – men inte i onödan

Filip Strömbäck, Klas Arvidsson

Planering

Vecka	Fö/Se	Lab
13	Fö: C + Syscall (påsk)	C ¹ , halt, exit, console
14	-	console, filhantering
15	Fö: Sema, Se: Process	main, första processen
16	Fö: Lås, cond.	exec, exit
17	Fö: Implementation	exec, exit, wait
18	Fö: Deadlock	Synkronisering
19	-	Synkronisering, accesskontroll
20	Se: Deadlock + tenta	Synkronisering, accesskontroll
21	-	Synkronisering, accesskontroll

¹lämpligt att demonstrera första passet

- 1 Varför vänta?
- 2 Semaforen
- 3 Binär semafor
- 4 Räknande semafor

Ett posthämtningsproblem

En bonde hämtar posten

- Posten har dragit ner på servicen till gården, det är 3 km till brevlådan
- Posten brukar komma runt kl 13
- Bonden kommer till lådan kl 13:25
- Lådan är tom
- Har brevbäraren passerat?
- Hur ska bonden göra för att slippa gå i onödan?

Ett järnvägsproblem

- Järnvägsknut med två spår österut, ett västerut
- Ett X2000 kommer västerifrån och fortsätter mot sydost klockan 16:10 enligt tidtabell
- Ett Green Cargo 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 mellan bergen

Mellan två bergstoppar går en gångbro

- Bron är inte särskilt stadig och mycket lång
- Bron är så högt upp att den ofta helt eller delvis är inbäddad i moln (dimma)
- Sikten är alltså lika med noll
- 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

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`)

- 1 Varför vänta?
- 2 **Semaforen**
- 3 Binär semafor
- 4 Räknande semafor

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);
```

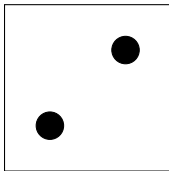
Andra funktionsnamn

- 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



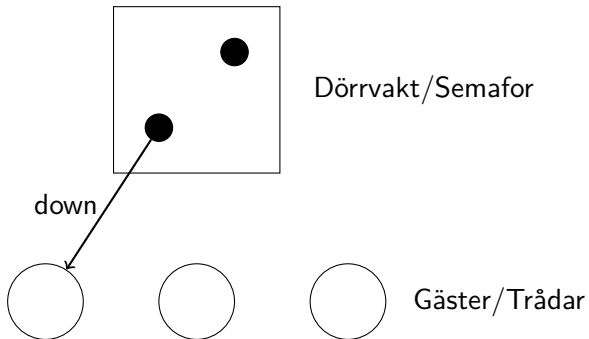
Dörrvakt/Semafor



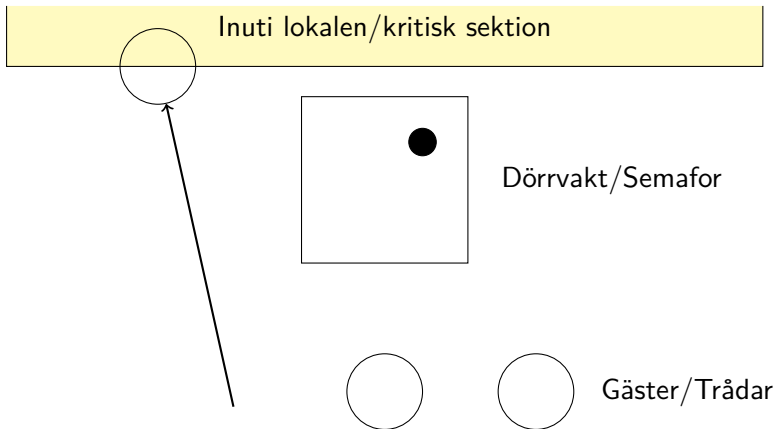
Gäster/Trådar

Exempel

Inuti lokalen/kritisk sektion

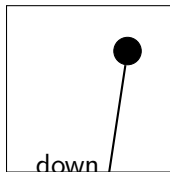


Exempel



Exempel

Inuti lokalen/kritisk sektion

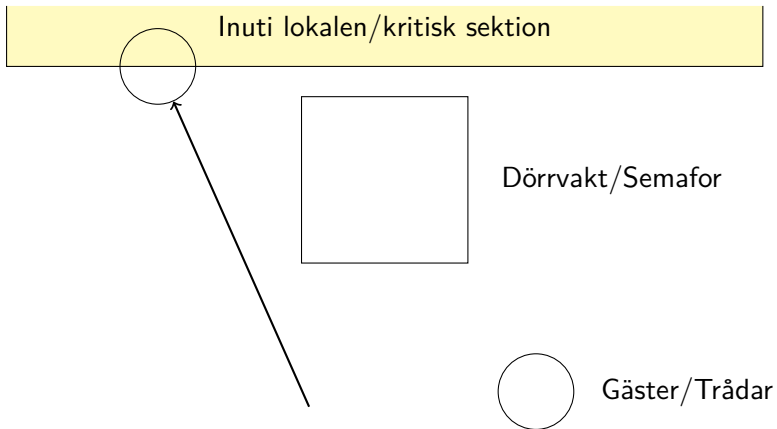


Dörrvakt/Semafor



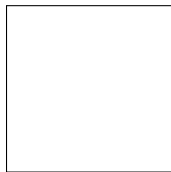
Gäster/Trådar

Exempel



Exempel

Inuti lokalen/kritisk sektion



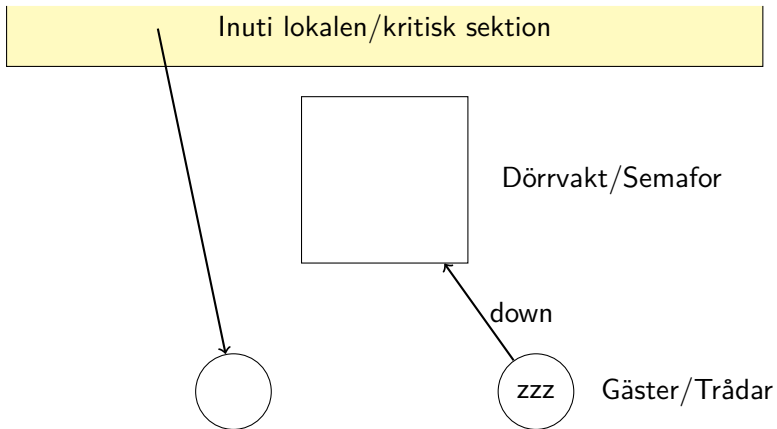
Dörrvakt/Semafor

down



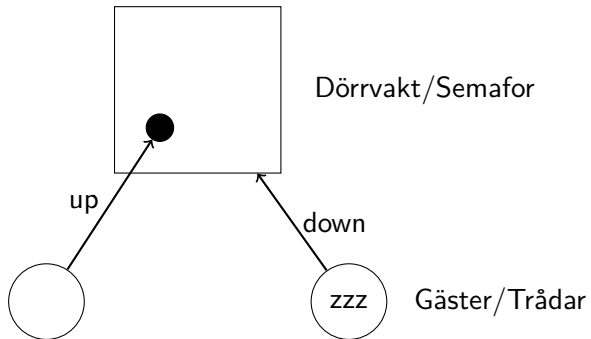
Gäster/Trådar

Exempel



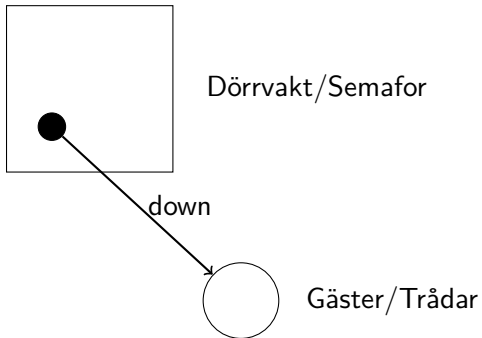
Exempel

Inuti lokalen/kritisk sektion

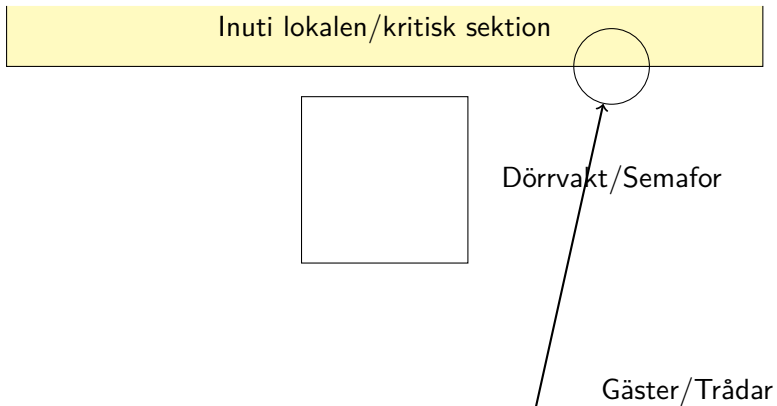


Exempel

Inuti lokalen/kritisk sektion



Exempel



- 1 Varför vänta?
- 2 Semaforen
- 3 **Binär semafor**
- 4 Räknande semafor

Summering av fält

```
// En stor array fylld med ettor.  
int array[2_000_000] = {1, 1, 1, 1, ...};  
  
// Variabler för att lagra summor och delsummer.  
int sum = 0, suml = 0, sumh = 0;  
  
// Två funktioner som summerar var sin del.  
void sum_low();  
void sum_high();
```

Summering av fält

```
void sum_low() {  
    for (int i = 0; i < 1_000_000; i++)  
        suml += array[i];  
}
```

```
void sum_high() {  
    for (int i = 0; i < 1_000_000; i++)  
        sumh += array[i + 1_000_000];  
}
```


Summering av fält

```
int main() {  
    thread_create(sum_low, ...);  
    thread_create(sum_high, ...);  
    sum = suml + sumh;  
    printf("Summa: %d\n", sum);  
    return 0;  
}
```

Summering av fält

```
int main() {  
    thread_create(sum_low, ...);  
    thread_create(sum_high, ...);  
    sum = suml + sumh;  
    printf("Summa: %d\n", sum);  
    return 0;  
}
```

Vad blir resultatet?

Summering av fält (EJ OK)

```
void sum_low() {  
    for (int i = 0; i < 1_000_000; i++)  
        suml += array[i];  
    has_suml = true;  
}
```

```
void sum_high() {  
    for (int i = 0; i < 1_000_000; i++)  
        sumh += array[i + 1_000_000];  
    has_sumh = true;  
}
```

Summering av fält (EJ OK)

```
int main() {
    thread_create(sum_low);
    thread_create(sum_high);
    while (!has_suml)
        ;
    while (!has_sumh)
        ;
    sum = suml + sumh;
    printf("Summa: %d\n", sum);
    return 0;
}
```

Hur mycket CPU-tid behövs?

Summering av fält (EJ OK)

```
int main() {
    thread_create(sum_low);
    thread_create(sum_high);
    while (!has_suml)
        ;
    while (!has_sumh)
        ;
    sum = suml + sumh;
    printf("Summa: %d\n", sum);
    return 0;
}
```

Hur mycket CPU-tid behövs? **Busy wait! Ej OK!**

Binär semafor

- Har två lägen (d.v.s. binär) 1 och 0
- Talar om ifall en händelse har inträffat eller inte
 - X2000 har passerat
 - Brevbäraren har passerat
- Talar om ifall en resurs är tillgänglig eller inte
 - Spåret västerut är fritt
 - Posten finns i lådan
- Kan starta på vilket läge som helst

Summering av fält (OK)

```
void sum_low() {  
    for (int i = 0; i < 1_000_000; i++)  
        suml += array[i];  
    sema_up(&has_suml);  
}
```

```
void sum_high() {  
    for (int i = 0; i < 1_000_000; i++)  
        sumh += array[i + 1_000_000];  
    sema_up(&has_sumh);  
}
```

Summering av fält (OK)

```
int main() {
    sema_init(&has_suml, 0);
    sema_init(&has_sumh, 0);
    thread_create(sum_low);
    thread_create(sum_high);

    sema_down(&has_suml);
    sema_down(&has_sumh);
    sum = suml + sumh;
    printf("Summa: %d\n", sum);
    return 0;
}
```


Behöver vi två semaforer?

Behöver vi två semaforer?

```
int main() {
    sema_init(&has_sums, /* ??? */);
    thread_create(sum_low);
    thread_create(sum_high);

    // Hur ska vi vänta?

    sum = suml + sumh;
    printf("Summa: %d\n", sum);
    return 0;
}
```

Behöver vi två semaforer? (OK)

```
int main() {
    sema_init(&has_sums, 0);
    thread_create(sum_low);
    thread_create(sum_high);

    for (int i = 0; i < 2; i++)
        sema_down(&has_sums);
    sum = suml + sumh;
    printf("Summa: %d\n", sum);
    return 0;
}
```

Behöver vi två semaforer? (FEL)

```
int main() {
    sema_init(&has_sums, 2); // Finns inget ännu...
    thread_create(sum_low);
    thread_create(sum_high);

    // FEL: Väntar inte. 'has_sums' är redan 2.
    sema_down(&has_sums);
    sum = suml + sumh;
    printf("Summa: %d\n", sum);
    return 0;
}
```

- 1 Varför vänta?
- 2 Semaforen
- 3 Binär semafor
- 4 Räkande semafor

Räknande semafor

- Räknar *antalet* **tillgängliga** resurser
- Avgör när en användare av resursen måste vänta
 - När antalet är noll
- Avgör när användaren kan sluta vänta
 - När antalet blir ett eller mer
- En utökning av en *binär semafor*
 - Kan göra allt som binära semaforer kan, och mer

Räknande semafor

- Väntar bara då vi försöker räkna ner en semafor som redan är 0
- Vi kan alltså **inte** använda den för att räkna upptagna resurser. Då skulle vi...
 - ...räkna upp när en resurs förbrukas, dvs. sluta vänta när en resurs förbrukas... **(FEL)**
 - ...räkna ner när en resurs blir ledig, dvs. kanske vänta på något som precis blev ledigt... **(FEL)**

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!

Summering av fält igen (nästa fö)

sumh och suml \Rightarrow sum

```
void sum_low() {  
    for (int i = 0; i < 1_000_000; i++)  
        sum += array[i];  
}
```

```
void sum_high() {  
    for (int i = 0; i < 1_000_000; i++)  
        sum += array[i + 1_000_000];  
}
```

Finns det några problem med denna lösning?

Filip Strömbäck, Klas Arvidsson

www.liu.se