Principper for Samtidighed og Styresystemer

Processer og Tråde

René Rydhof Hansen

Februar 2008

Mål

- At kunne forklare forskellige allokeringsstrategier for blokke
- At kunne forklare udviklerperspektivet på filsystemer (API/interface/systemkald)
- At kunne forklare hvordan kataloger er implementeret
- At kunne forklare hvad tråde og processer er
- At kunne forklare hvad gensidig udelukkelse er og hvordan det opnås

Opgaver

- Opgave 1: Grundlæggende OS begreber
- Opgave 2: Grundlæggende filsystembegreber
- Opgave 3: inodes
- Opgave 4: Filoperationer

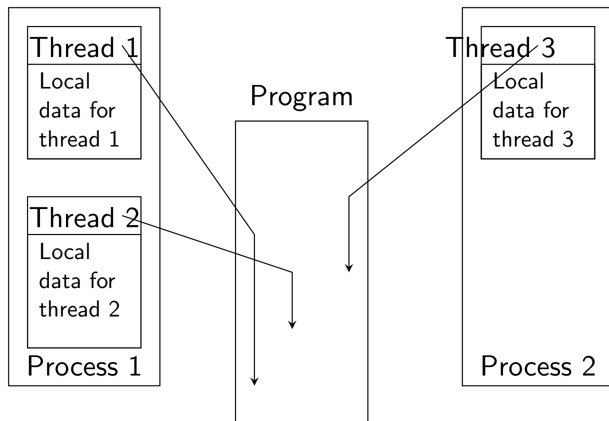
Processer og Tråde

- Processer
  - Den omgivelse et program udføres i ([English])
  - Et kørende program
- Tråde
  - Den del af processen der udføres ([English])
  - Den størrelse der udføres af styresystemet
  - "Letvægtsprocesser"

Multithreading og Multiprogrammering

- Multithreading
  - En opdeling af processer i flere tråde
  - I systemer uden multithreading: kun processer
- Multiprogrammering
  - Opdeling af tasks der kan udføres samtidigt på delte processorer
  - Implementeres ved hurtige skift mellem de enkelte tasks

Tråde



Tråde i Java

```
public class MyThread extends Thread
{
  public void run()
  {
    for(int i=1; i <= 1000; i++)
    {
      System.out.println(i);
    }
  }
}

MyThread thread1 = new MyThread();
MyThread thread2 = new MyThread();
thread1.start();
thread2.start();
```

Hvad er resultatet af følgende program?

```
int i = 0;

void thread()
{
  for(int j = 0; j < 10000;j++)
  {
    i++;
  }
}

t1 = run(thread);
t2 = run(thread);
wait(t1);
wait(t2);
print(i);
```

Hvad er resultatet af følgende program?

```
int i = 0;

void thread()
{
  for(int j = 0; j < 10000;j++)
  {
    i++;  // i++ is non-atomic
  }
}

t1 = run(thread);
t2 = run(thread);
wait(t1);
wait(t2);
print(i);
```

Gensidig udelukkelse — Vigtige begreber

- Race Condition
  - Tilstand hvor resultatet afhænger af den relative hastighed af de enkelte tråde
  - Det vil sige af den faktiske ordning (interleaving) af hændelserne i trådene
  - Vanskeligt at fejlfinde
- Kritisk region
  - Programfragment der giver anledning til race conditions
  - Engelsk: "Critical sections" eller "critical regions"
- Gensidig udelukkelse
  - En tilstand hvor der blandt en mængde tråde kun er en enkelt tråd der kan tilgå en bestemt ressource eller udføre en bestemt del af programteksten
  - Engelsk: "mutual exclusion"
- Atomisk
  - Hændelse eller sekvens af hændelser der sker uafbrydeligt

Gensidig udelukkelse

Kritiske regioner skal udføres under gensidig udelukkelse!

Højniveausprog

`a = a + 1` ⟶

```
ld a, r1
add r1,1
st r1,a
```

- Sætninger i højniveausprog er ikke atomiske
- Processorinstruktioner kan antages at være atomiske
- Parallel skrivning til lagercelle har uforudsigeligt resultat

mutex algoritmer

```
bool flag[2] = { false, false };
```

```
thread P0
{
  while flag[1]
  {
  }
  flag[0] = true;

  /* kritisk region */

  flag[0] = false;
}
```

```
thread P1
{
  while flag[0]
  {
  }
  flag[1] = true;

  /* kritisk region */

  flag[1] = false;
}
```

Algoritmen er forkert!

mutex algoritmer

```
bool flag[2] = { false, false };

thread P0
{
  while flag[1]
  {
  }
  flag[0] = true;

  /* kritisk region */

  flag[0] = false;
}

thread P1
{
  while flag[0]
  {
  }
  flag[1] = true;

  /* kritisk region */

  flag[1] = false;
}
```

Algoritmen er forkert!

Decker's mutex-algoritme

```
bool flag[2] = {false, false};
int turn = 0;

process P0 {
  flag[0] = true;
  while flag[1] {
    if turn == 1 {
      flag[0] = false;
      while turn == 1 { }
      flag[0] = true;
    }
  }
  /* kritisk region */
  turn = 1;
  flag[0] = false;
}
```

```
process P1 {
   flag[1] = true;
  while flag[0] {
   if turn == 0 {
      flag[1] = false;
     while turn == 0 { }
      flag[1] = true;
   }
  }
   /* kritisk region */
   turn = 0;
   flag[1] = false;
}
```

- Ikke særlig effektiv
- Problem: compiler kan allokere visse variable til registre (dermed ikke delt mellem trådene)

Test and Set

```
int mutex = 0;
```

```
thread P0
{
 while test_and_set(mutex)==1
 {
 }
 /* kritisk region */
 mutex = 0;
}
```

```
thread P1
{
 while test_and_set(mutex)==1
 {
 }
  /* kritisk region */
  mutex = 0;
}
```

- `test_and_set` findes ofte som en atomisk processorinstruktion

Exchange

- `xchange(a,b)` bytter `a` og `b`
- Hvis `xchange` er atomisk, hvordan kan den så bruges til at sikre gensidig udelukkelse?

Opsummering og næste gang

- Filsystemer fra et udviklerperspektiv
  - API/interface/systemkald
  - Allokeringsstrategier for diskblokke
  - Kataloger
- Næste gang: semaforer, synkronisering, ...