

INF2440 Eksamen 2016

løsningsforslag

Arne Maus, PSE
ifi, UiO

Oppgave 1 (10 poeng)

- a) Beskriv meget kortfattet de to viktigste egenskapene ved tråder i et Java-program.
1. En tråd er sekvensielt programdel (et objekt) som kjører i parallell med evt. andre tråder i programmet.
 2. En tråd deler/ser samme adresserom i RAM de andre trådene i programmet.
- b) Terminerer programmet alltid når maintråden er ferdig? Begrunn svaret.
1. Nei, fordi hvis main-tråden har startet andre tråder som ikke er ferdige med sin run()-metode, fortsetter programmet. Et program terminerer først når alle trådene i programmet terminerer.

Oppgave 2 (20 poeng)

- a) Tenk deg et program hvor det er en tråd som usynkronisert både leser og øker en variabel 'x' (initielt = 0) med 1 hver gang den er ferdig med en deloppgave, mens de andre trådene går i løkke, leser 'x' og skal terminere når $x == 20$. Kan dette gå galt – vil alle 'lese-trådene' sikkert terminere hvis det løses minst 22 slike deloppgaver? Begrunn svaret (maksimalt 10 linjer).

Dette går ikke alltid bra fordi alle tråder ser ikke nødvendigvis samme verdi på felles variable, og de er heller ikke garantert å se alle verdiene av en slik variabel x. En eller flere tråder kan derfor risikere å ikke se verdien 20 på x (men f.eks 19 og så 21), og vil derfor ikke terminere.

- a) Tenk deg et program hvor det er en tråd som usynkronisert både leser og øker en variabel 'x' (initielt = 0) med 1 hver gang den er ferdig med en deloppgave, mens de andre trådene går i løkke, leser 'x' og skal terminere når $x \geq 20$. Kan dette gå galt – vil alle leser-trådene sikkert terminere hvis x økes til minst 22? Begrunn svaret (maksimalt 10 linjer).

Dette går bra fordi om selv om ikke alle tråder ser ikke nødvendigvis samme verdi på felles variable, vil de før eller siden se den siste verdien av x ($==22$) og terminere.

Oppgave 3 (25 poeng)

Se på koden i Vedlegg 1, der vi har et lite parallelt program som skriver litt ut i main og i run. Merk den litt uvanlige initiering av CyclicBarrier b i forhold til hvor mange tråder vi har.

```
import java.util.concurrent.*;
class Problem {
    // felles data og metoder A
    static int num = 3;
    CyclicBarrier b = new CyclicBarrier(num);

    public static void main(String [] args) {
        Problem p = new Problem();
        p.utfoer(num+1);           // num+1 == 4
        System.out.println(" Main-tråden TERMINERER");
    } // end main

    void utfoer (int antT) {
        Thread [] t = new Thread [antT];
        for (int i =0; i< antT; i++)
            ( t[i] = new Thread(new Arbeider(i))).start();
        try{
            for (int i =0; i< antT; i++) t[i].join();
        } catch(Exception e) {}
    } // end utfoer
```

```
class Arbeider implements Runnable {
    // lokale data og metoder B
    void sync() { try{ b.await()
        catch (Exception e) { return;}
    }
    int ind;

    Arbeider (int in) {ind = in;}

    public void run( ) {
        // kalles når tråden er startet
        sync();
        System.out.println("A");
        sync();
        System.out.println("B");
    } // end run
} // end indre klasse Arbeider
} // end class Problem
```

Oppgave 3 (25 poeng)

Se på koden i Vedlegg 1, der vi har et lite parallelt program som skriver litt ut i main og i run. Merk den litt uvanlige initiering av CyclicBarrier b i forhold til hvor mange tråder vi har.

- a) Hva skriver programmet ut (hvor mange A-er og hvor mange B-er og kommer teksten 'Main terminerer' ut)?

Programmet skriver bare ut 4 A-er og 2 B-er. Vi starter 4 tråder, men synkroniserer på en Barriere som slipper igjennom 3. Først slipper da 3 gjennom og skriver A. Når de når neste kall på Barrieren slipper de løs systemenn som sier A. Siden en av de ledige plassene til å slippe gjennom blir brukt av tråden som sa A, kommer bare to tråder videre og sier B.

- b) Terminerer programmet – begrunn svaret.

Programmet terminerer ikke fordi ikke alle trådene er ferdige – to tråder sitte fast i den siste synkroniseringa.

- c) Hvis du endrer `num =3` til `num=2`, hva skriver programmet ut da?

Programmet skriver da ut 3 A-er og 3 B-er og "Main terminerer.

Vi starter nå 3 tråder, og to kommer da først gjennom første synkronisering, mens den siste tråden kører i første sync(). Så kommer første tråd fram til andre sync() og slipper den siste tråden gjennom så den skriver ut A og den selv sier B. Så når den siste tråden kommer fram til andre sync(), er det allerede køet opp en tråd der. Begge slipper da gjennom og skrive B og terminerer . Siden alle trådene terminerer skrives så 'Main terminerer' ut.

- d) Hvis **num=2** som i pkt. c), terminerer da programmet – begrunn svaret
Programmet terminerer – se pkt c).
- e) Som du ser står metoden :'**void sync()**' deklarerert i området B, lokalt i Arbeider-trådene . Tenkt deg at du flyttet denne deklarasjonen opp til området A i den ytre klassen. Beskriv kort hva dette vil ha å si for oppførselen av programmet i pkt. a)-d).

Intet endres ved hvor metoden sync() er deklarerert fordi den bruker den samme global deklarererte CyclicBarrier b i begge tilfellene.

Oppgave 4 (50 poeng)

Vi antar at du har løst Oblig2 med Erathostenes Sil. Vi er interessert i å teste om det **siste desimale siffer** i alle primtall under 1 milliard er jevnt fordelt (dvs. om det er jevnt over like mange 1,3,7 og 9-ere som er siste siffer i primtallene) . Det er opplagt at det er bare to primtall som slutter på 2 og 5, nemlig tallene 2 og 5 selv, og ingen andre primtall slutter på 0,2,4,6 og 8 (fordi slike tall er delbare på 2).

Du lager en Erathostenes sil slik:

```
ErathosthesSil s = new ErathosthesSil(1000000000);
```

Vi antar at du kaller denne bare en gang og nytter den sekvensielle Silen du laget i Oblig2. i både punktene a) og b) nedenfor.

Erathosthes Sil inneholder en metode:

```
int nextPrime(int i) {  
    // returnerer neste primtall >'i'  
    return ..;  
}
```

Denne skal du bruke til å undersøke påstanden at siste siffer i primtallene er meget jevnt fordelt (primtallsforskerne undersøker virkelig dette nå).

```
ReentrantLock lock = new ReentrantLock();  
int [] antSisteSiffer = new int [10]; // only 1,3,7,9 used  
ErathostenesSil s = new ErathostenesSil(1000000000);
```

For mange var det største problemet:

- Hvordan finne verdien siste siffer av et stort tall M ?

Svar:

```
int sisteSiffer = M%10; // Resten etter divisjon med 10
```

Oppgave 4.a) Skriv et sekvensielt program som lager en tabell over hvor mange av primtallene > 10 som ender på 1,3,7 eller 9. Bruk da silen du allerede har laget og metoden `int nextPrime(int i)` (som du ikke skal skrive, men anta er riktig og bare kan bruke). Beregn også det siste sifferet som færrest primtall slutter på i prosent av det siste sifferet som flest primtall slutter på. Skriv formelen for svaret.

Sensurkommentar: Dette er lett, så her og i 4b trekkes det for to mangler:

- **Søket etter neste primtall starter på 0 istedenfor 10**
 - **IKKE skrevet formel for % minste av største , eller gal formel.**
- (jeg har nyttet Arrays.sort for å sortere de 4 tallene – andre løsninger kunne vært valgt.)**

Kommentar: Dette er effektiv parallellisering, i min løsning tok sekvensiell løsning

Tid sekvensiell: 1210.333614ms

antallSisteSiffer[1]: 12711386

antallSisteSiffer[3]: 12712498

antallSisteSiffer[7]: 12712313

antallSisteSiffer[9]: 12711333

minste i % av største:99.990835790102

Tid parallel: 385.660646ms

```
void tellSisteSiffer(int [] ant, int fra, int tom) {
    int p = s.nextPrime(fra);
    while (p < tom) {
        ant[p%10]++;
        p = s.nextPrime (p);
    }
} // end tellSisteSiffer
.....
```

```
.....  
// ---sekvensiell-----  
  
tid = System.nanoTime();  
tellSisteSiffer( antSisteSiffer,10,1000000000 -10);  
System.out.println("Tid sekvensiell: "+  
    ((System.nanoTime() -tid)/1000000.0)+"ms");  
  
for (int i = 0; i <10; i++) {  
    System.out.println("antallSisteSiffer["+i+"]: "+antSisteSiffer[i]);  
}  
Arrays.sort(antSisteSiffer);  
System.out.println("minste i % av største:"+  
    (100.0*antSisteSiffer[6]/antSisteSiffer[9]));
```

Oppgave 4.b) Skriv et parallelt program som finner samme tabell som i oppgave 4.a. (Du skal altså bare parallellisere det å finne denne tabellen – ikke det å lage Erathosenes Sil eller det å finne den største og minste av 4 tall). Bruk Modell2-koden og forklar bare hvor det du skriver skal plasseres inn der.

```
void addLocal(int [] local){
    lock.lock();
    try{
        for(int i = 0; i < 10; i++){
            antSisteSiffer[i] += local[i];
        }
    } finally { lock.unlock();}
} // end addLocal
```

```
class Arbeider implements Runnable {
    int ind; // lokale data og metoder B
    int start, end,num;
    int [] antSiffLokal= new int[10];
```

```
    Arbeider (int in) {ind = in; }
    public void run( ) {
        num= (1000000000-10)/antT;
        start = ind*num +10;
        end = start + num ;
        if (ind == antT -1) end += (1000000000-10)%antT;
        tellSisteSiffer(antSiffLokal, start, end) ;
        addLocal (antSiffLokal);
    } // end run
```

```
} // end indre klasse Arbeider
```

Oppgave 5 (40 poeng)

Skriv et sekvensielt og parallelt program som søker etter og finner ut om, og eventuelt da hvor, et tall 's' er i en usortert array: `int [] a = new int[n]`. Det er mulig at 's', det tallet du leter etter, finnes flere steder i arrayen, og da er det likegyldig hvilke av stedene du svarer.

Du skal bruke Modell2 koden i vedlegget og bare skrive én metode (med nok parametre) som søker i arrayen og som nyttes av både den sekvensielle løsningen og den parallelle løsningen. I tillegg i den parallelle løsningen kan det komme kode `run()` metoden som sammenligner svaret fra de k trådene du starter, men det er kanskje ikke tilfellet her? Begrunn valget ditt på dette punktet.

Prøv å lage den parallelle koden slik at alle trådene avslutter raskt når en av de andre trådene har funnet 's'.

Svaret skal foreligge enten som `-1` i en globalt synlig variabel `int svarIndex`; (i området A i vedlegget) som betyr at tallet ikke fantes; eller som et tall `>= 0` i `svarIndex` som da sier en plass i `a[]` du finner tallet 's' som du leter etter. Gi også en vurdering av om denne oppgaven følger Amdahl lov eller Gustavsons lov når vi øker n, lengden av `a[]`.

Sensur-kommentarer:

Den sekvensielle er enkel, man må selvsagt trekke for om man fortsetter å søke etter å ha funnet tallet 's'

Den parallelle kunne vært skrevet om lag som den sekvensielle, men her forsøker vi å dele opp tallområdet for hver tråd i 10 smådeleler, slik at vi kanskje kan returnere raskere fordi selv om den tråden som evt. finner 's' terminerer raskt, så må de trådene som har ikke finner 's' fortsette søket selv etter at 's' er funnet – derfor ideen om å teste på `svarIndex >= 0` som vi ikke har råd til å teste for ofte på, men f.eks si 10x per tråd. Ekstra bonus til kandidater som prøver noe tilsvarende (dvs. ikke full poengsum til de som ikke gjør det)

Denne oppgaven burde kunne få linær speedup – fordi det knapt er sekvensiell kode, og den er konstant $O(1)$, mens resten er $O(n)$. Dvs følger Gustavsson lov (bedre speedup når n øker)

Sekvensiell:

```
void finnTall(int [] a,int s,int start, int end){
    for (int i = start; i < end; i++) {
        if (a[i] == s) {
            svarIndex =i;
            return;
        }
    } // end for
} // end finnTall
```

Parallell:

```
public void run() {
    // deler området til denne tråden i 10 -smådeler
    num= n/antT;
    start = ind*num ;
    end = start + num ;
    int count =0;
    if (ind == antT -1) end += n%antT;
    incr = num /10;

    for (count = 0; count < 10; count++){
        if (count == 9) incr = end - start;
        if (svarIndex < 0) finnTall(a,s,start, start+incr);
        start += incr;
    }
}
} // end run
```



```

void swapSort (int [] a , int left, int right) {
    int rh = (right+left)/2-1, t;

    for (int i = left; i < (right-left)/2; i++){
        if (a[i] > a[rh+i]) {
            t = a[i];
            a[i] = a[rh+i];
            a[rh+i] = t;
        }
    }

    if (right-left > 10) {
        swapSort(a,left ,rh);
        swapSort(a,rh+1, right);
    }
} // end swapSort

```

Kall :

```

swapSort(a,0,a.length);
insertSort (a, 0, a.length-1); // finally sort all with insertion-sort

```

- 2. Lag en parallell versjon av dette sekvensielle programmet (bare parallelliser ombyttingene – ikke det avsluttende kallet på innstikksortering).**

Her må det trekkes en del poeng (10) hvis man parallelliserer med tråder helt i bunn (klart undervist at det bare er toppen av rekursjonstreet som skal parallelliseres med tråder).

Løsninger med en int level er undervist, kan alternativt kutte hvis f.eks lengden av (right-left) < 30000, og da bruke vanlig rekursjon,

```

void swapPara(int [] a , int left, int right, int level) {
    int rh = (right+left)/2-1, n = right-1, t;
    for (int i = left; i < (right-left)/2; i++){
        if (a[i] > a[rh+i]) {
            t = a[i];
            a[i] = a[rh+i];
            a[rh+i] = t;
        }
    }

    if (level > 0 && right-left > 10) {
        Thread t1,t2;
        t1 = new Thread(new Arbeiter( left,rh-1,level-1 )); t1.start();
        t2 = new Thread(new Arbeiter( rh, right,level-1)); t2.start();

        try {t1.join(); }catch(Exception e){return;}
        try {t2.join(); }catch(Exception e){return;}
    }else if ( right-left > 10){
        swapPara(a,left, rh, level-1);
        swapPara(a, rh, right, level-1);
    }

} // end swapPara

```

```
class Arbeider implements Runnable {  
    // lokale data og metoder B  
    int left,right, level;  
  
    Arbeider (int left, int right, int level) {  
        this.left = left;  
        this.right = right;  
        this.level = level;  
    }  
  
    public void run( ) {  
        // kalles når tråden er startet –  
        swapPara (a,left,right, level);  
    } // end run  
} // end indre klasse Arbeider
```

Kall:

```
swapPara(a,0,a.length, maxLevel);  
innstikkSortSort (a, 0, a.length-1);
```

Til sist : **Gi din vurdering om dette er en god idé eller ganske dårlig idé – begrunn svaret.**

Kjøringer viser at Swapsort grovt sett er dobbelt så rask både sekvensiell og parallell versjon som innstikkSort, men det finnes bedre alternativer (Shellsort) .

Kandidatene kan jo ikke kjøre programmene på eksamen slik at de kan om lag si hva de vil. Egentlig er dette en god teoretisk ide, å gjøre arrayen lettere å sortere for innstikkSort, men vi har ikke tatt et best mulig grep her.

Lykke til med: først Oblig4 og
så på eksamen 14 juni!