Wiki »

Exercise3 Posix Threads

Exercise 1 Creating Posix Threads

Exercise 1.1 Fundamentals

- **What is the name of the Posix function that creates a thread?**
  Funktionen

  ```
  pthread_create()
  ```

  Opretter nye tråde.

- **Which arguments does it take and what do they represent?**
  Funktionen *pthread_create()* skal have 4 parametre:

  ```
  pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*func)(voi
  ```

  1. pthread_t *thread: indeholder en pointer hvor informationen tråden laver bliver gemt i.
  2. const pthread_attr_t **attr: Er en pointer til et objekt der specificerer attributter for den oprettede tråd. Hvis dette er en nullptr bliver tråden oprettet med default værdier.**
     #void (func)(void *): Er en funktions pointer af typen void, som pointer til den memory der er tilknyttet funktionen. Funktionen tager et void argument og returnerer en void*.
  3. void **arg: Returtypen er void**. Dette betyder at der kan sættes en pointer til ethvert objekt for at kalde funktionen. arg kan være en nullptr eller pege på en global eller heap variabel.

- **What happens when a thread is created?**
  Når en tråd oprettes, eksekveres funktionen hvorefter den returnerer en void pointer og derefter termineres.

- **What is a function pointer?**
  En funktions pointer peger på den eksekverbare kode af en funktion. Generelt kan pointers bruges til at kalde funktioner og videregive funktioner som argumenter i andre funktioner.

- **A function is the be supplied to the aforementioned function:**
  - **Which argument(s) does it take?**
    Der skal bruges en void pointer
  - **What is the return type/value?**
    Returtypen/værdien er en void pointer
  - **What can they be used for and how?**
    Alle typer kan bruges som parameter i en void pointer, men efter brug skal de "type castes" tilbage til den type de havde før de blev brugt som en void pointer.

Exercise 1.2 First Threading program

Herunder ses koden for programmet der opretter en enkelt tråd som udskriver "Hello world" før den terminerer:

```
#include <iostream>
```

```
#include <thread>
#include <pthread.h>
#include <string>

using namespace std;

struct Data
{
string s = "Hello world 42 ";
int i = 42;
};

void* hello(void* ptr){

Data* data = (Data*)ptr;
cout << data->s << "i: " << data->i <<  endl;
return 0;
}

int main(int argc, char *argv[])
{
pthread_t tid;

Data data;
pthread_create(&tid,nullptr, hello, &data);
pthread_join(tid, nullptr);

return 0;

}
```

Her ses det i shell'en at programmet kører som forventet:

```
stud@stud-virtual-machine:~/Desktop/ISU/lektion3/exe1.2$ ./exe
Hello world 42 i: 42
```

Exercise 2 Two threads

Herunder ses den kode til programmet som opretter to tråde der udskriver til stdout hvert sekund sammen med hvor mange gange den pågældende tråd har udskrevet. Trådene terminerer efter de har udskrivet 10 gange til stdout:

```
#include <pthread.h>
#include <thread>
#include <unistd.h>
#include <iostream>

using namespace std;

void* hello1(void* arg)
{
for(int i=0;i<10;i++){
    cout << "Hello #" << i << "from thread " <<  pthread_self() <<  endl;
    //printf("Hello #%d from thread 1",i);
    sleep(1);
```

```
    }
    return NULL;
}

int main(int argc, char *argv[])
{

pthread_t t0,t1;

pthread_create(&t0,NULL,hello1,NULL);
pthread_create(&t1,NULL,hello1,NULL);

pthread_join(t0,NULL);
pthread_join(t1,NULL);

exit(EXIT_SUCCESS);

}
```

Herunder ses screenshot af programmet der eksekveres i shell'en:

```
stud@stud-virtual-machine:~/Desktop/ISU/lektion3/exe2$ ./exe
Hello #0from thread 139860700600064
Hello #0from thread 139860692207360
Hello #1from thread 139860700600064
Hello #1from thread 139860692207360
Hello #2from thread 139860700600064
Hello #2from thread 139860692207360
Hello #3from thread 139860700600064
Hello #3from thread 139860692207360
Hello #4from thread 139860700600064
Hello #4from thread 139860692207360
Hello #5from thread 139860700600064
Hello #5from thread 139860692207360
Hello #6from thread 139860700600064
Hello #6from thread 139860692207360
Hello #7from thread 139860700600064
Hello #7from thread 139860692207360
Hello #8from thread 139860700600064
Hello #8from thread 139860692207360
Hello #9from thread 139860700600064
Hello #9from thread 139860692207360
stud@stud-virtual-machine:~/Desktop/ISU/lektion3/exe2$
```

Questions to consider:

- **What happens if function main() returns immediately after creating the threads? Why?**
  Hvis main() funktionen terminerer umiddelbart efter trådene oprettes, termineres trådene også, og derved den funktion som bliver udført. For at stoppe dette bruges funktionen:

  ```
  pthread_join();
  ```

  Denne funktion bruges til at main() funktionen venter på at trådene afslutter før den går videre (og derved terminerer)

- **The seemingly easy task of passing the ID to the thread may present a challenge; In your chosen solution what have you done? Have you used a pointer or a scalar?**

Vi bruger hverken pionter eller scalar idet vi har anvendt funktionen pthread_self(), som returnerer ID på den tråd, som kalder funktionen.

Exercise 3 Sharing data between threads

Herunder ses den kode til programmet som opretter to tråde, hvorefter det inkrementerer og aflæser. De to tråde deler en *unsigned integer* som er kaldt *shared* som er sat til 0. Denne variabel bliver så talt op hvert sekund i takt med at der bliver aflæst og skrevet til terminalen med *cout*.

```
#include <pthread.h>
#include <thread>
#include <unistd.h>
#include <iostream>

using namespace std;

unsigned int shared;

void* inc(void* arg)
{
    while(true){
        shared++;
        sleep(1);
    }
    return NULL;
}

void *read(void* arg){
    while(true){
        cout << shared << endl;
        sleep(1);
    }
    return NULL;
}

int main(int argc, char *argv[])
{
shared = 0;

pthread_t t0,t1;

pthread_create(&t0,NULL, read,NULL);
pthread_create(&t1,NULL,inc,NULL);

pthread_join(t0,NULL);
pthread_join(t1,NULL);

exit(EXIT_SUCCESS);

}
```

Herunder ses screenshot af programmet der eksekveres i shell'en:

```
stud@stud-virtual-machine:~/Desktop/ISU/lektion3/exe3$ ./exe
0
1
3
3
4
6
7
8
9
10
^C
stud@stud-virtual-machine:~/Desktop/ISU/lektion3/exe3$
```

- **Are there any problems in this program? Do you see any?**

Når programmet køres, kan det ses i terminalen at en af trådene i nogle tilfælde bliver eksekveret 2 gange før at den anden når det. Dette resulterer i at der bliver skrevet flere ens tal ud i terminalen lige efter hinanden. Dette kan ses på billedet nedenfor.

Exercise 4 Sharing a Vector class between threads

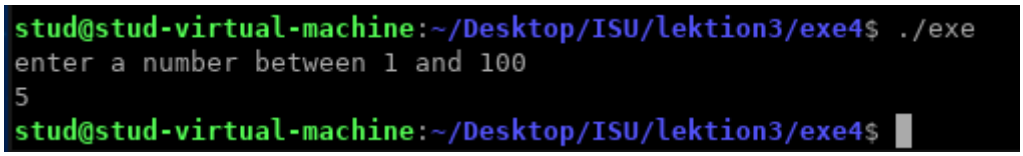
Herunder ses den kode til programmet som opretter en tråde, der kalder setAndTest(), som sætter og tester værdierne på det Vector objekt som bliver delt imellem trådene. *Writer* funtkionen bruger så denne metode, samt giver en error hvis tallet ikke er sat som en gyldig værdi.

```
#include <pthread.h>
#include <thread>
#include <unistd.h>
#include <iostream>
#include <vector>
#include "vector.hpp"

using namespace std;

unsigned int shared;

void* writer(void* arg)
{
if ((*(Vector*)arg).setAndTest(*(int*)pthread_self()) == false){
    cout << "error" << endl;
}
return NULL;
}

int main(int argc, char *argv[])
{
int input;
Vector v1;

cout << "enter a number between 1 and 100" << endl;
cin >> input;

pthread_t t[input];

for(int i=0;i<input;i++){
    pthread_create(&t[i],NULL,writer,&v1);
    sleep(1);
```

```
}

for(int i=0;i<input;i++){
    pthread_join(t[i],NULL);
}

return 0;

}
```

Der ses ingen problemer, fordi at der er et helt sekund pr. tråd, hvilket gør at den når at lukke ned igen inden en ny tråd åbnes.

Herunder ses screenshot af programmet der eksekveres i shell'en:



Exercise 5 Tweaking parameters

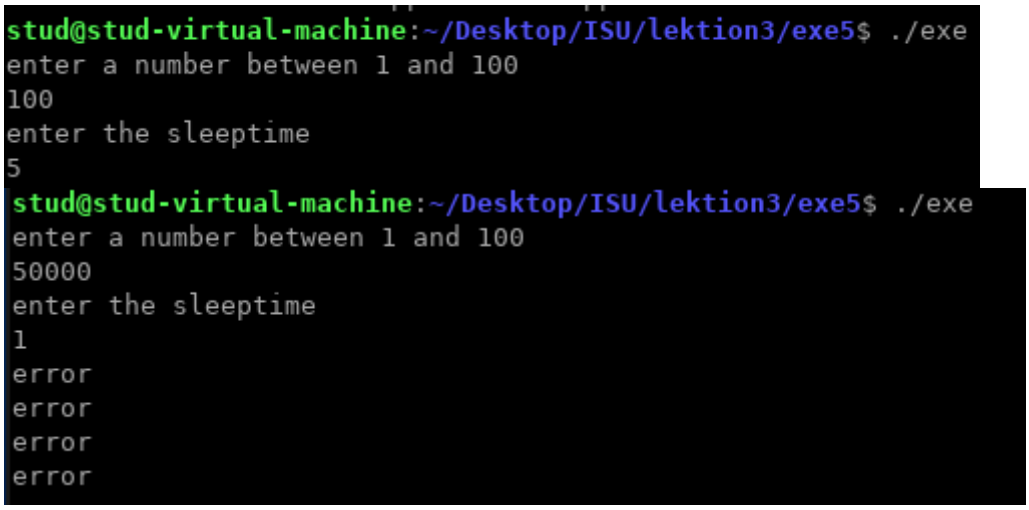
Herunder ses den modificerede kode fra Exercise 4, som er i stand til at modtage et brugervalgt delay, som er i millisekunder.

```
#include <pthread.h>
#include <thread>
#include <unistd.h>
#include <iostream>
#include <vector>
#include "vector.hpp"

using namespace std;

unsigned int shared;

void* writer(void* arg)
{
if ((*(Vector*)arg).setAndTest(*(int*)pthread_self()) == false){
    cout << "error" << endl;
}
return NULL;
}

int main(int argc, char *argv[])
{
int input;
int inputtime;
Vector v1;

cout << "enter a number between 1 and 100" << endl;
cin >> input;
cout << "enter the sleeptime" << endl;
cin >> inputtime;

pthread_t t[input];
```

```
for(int i=0;i<input;i++){
    pthread_create(&t[i],NULL,writer,&v1);
    usleep(inputtime);
}

for(int i=0;i<input;i++){
    pthread_join(t[i],NULL);
}

return 0;

}
```

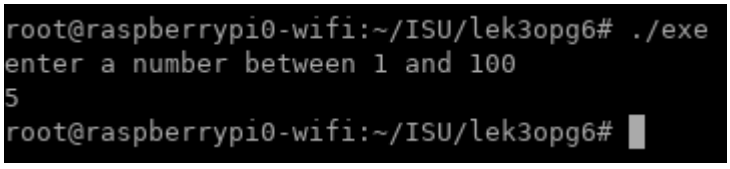
Der ses problemer hvis der sættes for mange tråd til for lidt tid. Dette skyldes at trådene ikke når at blive oprettet færdigt.

Herunder ses screenshot af programmet der eksekveres i shell'en:



Exercise 6 Testing on target

For at programmet kan køres på RPI'en er der avet om i makefilen, så den kompilerer korrekt. Istedet for g++ kompilerer den med arm-rpizw-g++.
Nedenfor ses eksekveringen i terminalen på RPI'en ved kørsel af programmet fra Exercise 4.



Der opleves ikke nogle problemer ved kørsel på RPI'en. Da vi igen brugte koden fra Exercise 4, har vi igen et sekund delay imellem oprettelsen af hver tråd, som gør at vi ikke oplever nogle fejl i kørslen af programmet.

- **Tilføjet af Andreas Elgaard Sørensen for cirka en måned siden**

Review

Learning objective #1: Forstå og forklar threads

I viser at i har forstået grundprincipperne for threads. Hvordan de oprettes, parameterene for thread_create() og returværdierne.
I går tilpas i detaljer uden at overgøre det.

Learning objective #2: Lav en thread og print til terminal

I har kunne oprette en thread korrekt og vide hvornår den skal termineres igen inden, så i kan få den til at udskrive til terminale. I har kunne oprette en makefile, der compilerer sorucefilen korrekt.
I kan med fordel næste gang udelukke så store kode udsnit i jeres rapport, men bare tage det vigtigste med, og have resten i repository.

Learning objective #3: Del data mellem threads

I har forstået at kunne oprette to threads og herefter dele data mellem dem. I har forstået hvilke problemer der kan opstå når data deles, uden at man tager forholdsregler.

Learning objective #4: Test på target

I forstår hvordan i compiler på Rpi, eftersom der skal ændres i den anvendte compiler, såden kan anvendes på på target.

Feedback

Journalen er fint lavet, og kommer godt omkring opgaverne. I har alle relevante kode udsnit med, men i kunne overveje om der skal skrives lidt mere i detaljer hvilke problemer der kan opstå når threds deler data og hvordan det kan undgåes, samt programmet kører så godt på target. Generelt en rigtigt god besvarelse.

Must haves

Relevant files in repository - Fuldført!
Makefiles in repository - Fuldført!

Konklusion

Alle læringsmål og relevante krav til den pågældende opgave er opfyldt, og vi har derfor givet opgave karakteren 'ok'.

1-2.png (5,63 KB) Emil Christian Foged Klemmensen, 2019-02-26 14:11
2.png (27,2 KB) Emil Christian Foged Klemmensen, 2019-02-26 14:13
3.png (11,2 KB) Emil Christian Foged Klemmensen, 2019-02-26 14:46
4.png (10,5 KB) Emil Christian Foged Klemmensen, 2019-02-26 14:47
51.png (7,5 KB) Emil Christian Foged Klemmensen, 2019-02-26 14:48
52.png (8,72 KB) Emil Christian Foged Klemmensen, 2019-02-26 14:48
6.png (7,28 KB) Emil Christian Foged Klemmensen, 2019-02-26 14:49