Wiki »

Exercise 3 - Posix Threads

Exercise 1 - Creating Posix Threads

Exercise 1.1 Fundamentals

• What is the name of the POSIX function that creates a thread?

pthread_create

• Which arguments does it take and what do they represent?

(*thread, *attr, *start, *arg)
*thread argument points to af buffer of type pthread_t with a unique identifier.
*attr argument points to at pthread_attr_t object, that specifies various attributes for the new thread.
*start argument is used to commence execution of a new thread, by calling the function identified by start with the argument *arg.

• What happens when a thread is created?

It will try to run the function, which the thread has a pointer to, with the CPU time.

• What is a function pointer?

A pointer to a function, which the thread is initialized with. Then the thread knows where it shall go to and run the commands, when it gets the CPU time.

• A function is to be supplied to the aforementioned function:
– Which argument(s) does it take?

void* that can be casted as any type.

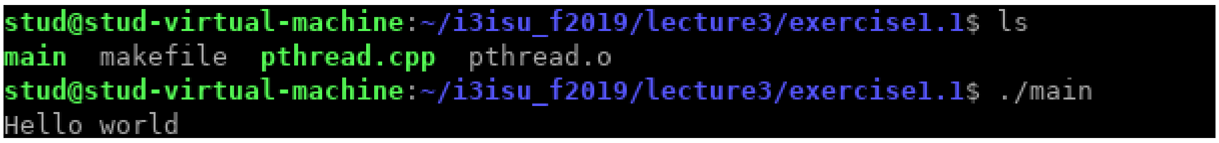
– What is the return type/value

void - no return value

– What can they be used for and how?

Running functions parallel, using threads to switch between them.

Having answered the questions, create program with a single thread that writes Hello world before terminating.



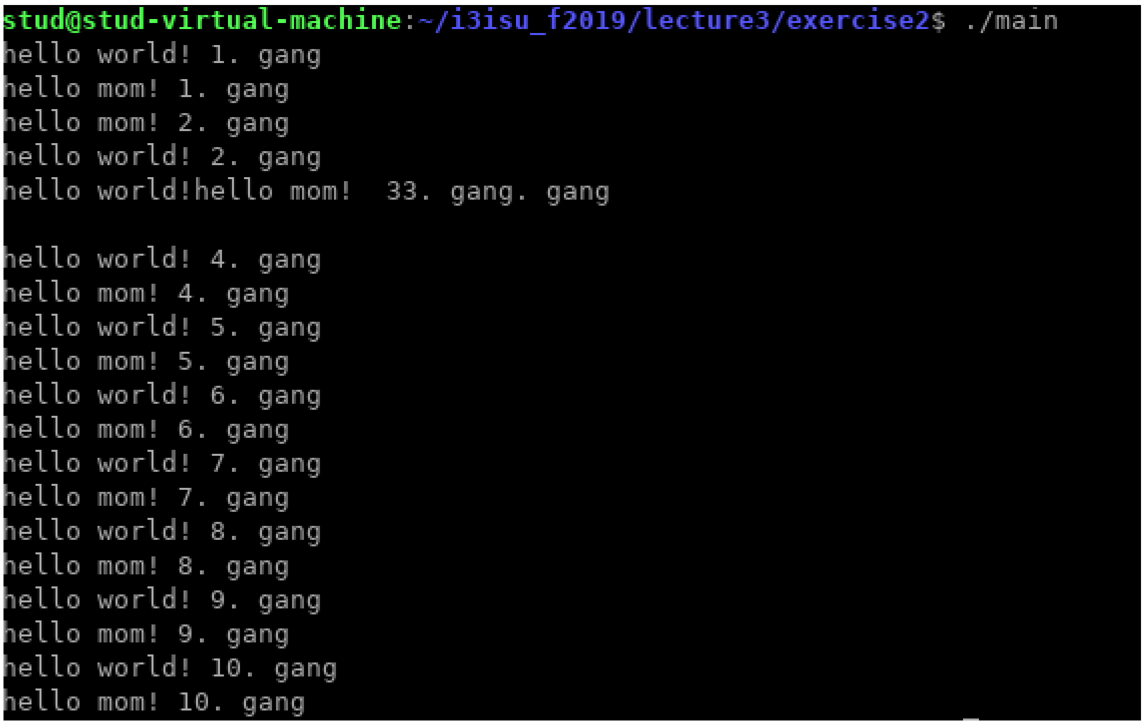
```
stud@stud-virtual-machine:~/i3isu_f2019/lecture3/exercise1.1$ ls
main  makefile  pthread.cpp  pthread.o
stud@stud-virtual-machine:~/i3isu_f2019/lecture3/exercise1.1$ ./main
Hello world
```

Exercise 2 - Two Threads

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

```
#include <stdlib.h>
#include <string>
#include <unistd.h>

using namespace std;

void* thrFunc(void *ptr)
{
    for( int i = 0; i<10; i++)
    {
        string *s = (string*) ptr;
        cout << *s << " " << (i+1) << ". gang" << endl;
        sleep(1);
    }
    return nullptr;
}

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

    pthread_t thr1;
    pthread_t thr2;

    string s1 = "hello world!";
    string s2 = "hello mom!";
    void *sptr;

    pthread_create(&thr1, nullptr, thrFunc, &s1);
    pthread_create(&thr2, nullptr, thrFunc, &s2);
    pthread_join(thr1, &sptr);
    pthread_join(thr2, &sptr);

    return 0;
}
```

Billede af eksekvering af "Two Threads"



```
stud@stud-virtual-machine:~/i3isu_f2019/lecture3/exercise2$ ./main
hello world! 1. gang
hello mom! 1. gang
hello mom! 2. gang
hello world! 2. gang
hello world!hello mom!  33. gang. gang

hello world! 4. gang
hello mom! 4. gang
hello world! 5. gang
hello mom! 5. gang
hello world! 6. gang
hello mom! 6. gang
hello world! 7. gang
hello mom! 7. gang
hello world! 8. gang
hello mom! 8. gang
hello world! 9. gang
hello mom! 9. gang
hello world! 10. gang
hello mom! 10. gang
```

• What happens if function main() returns immediately after creating the threads? Why?

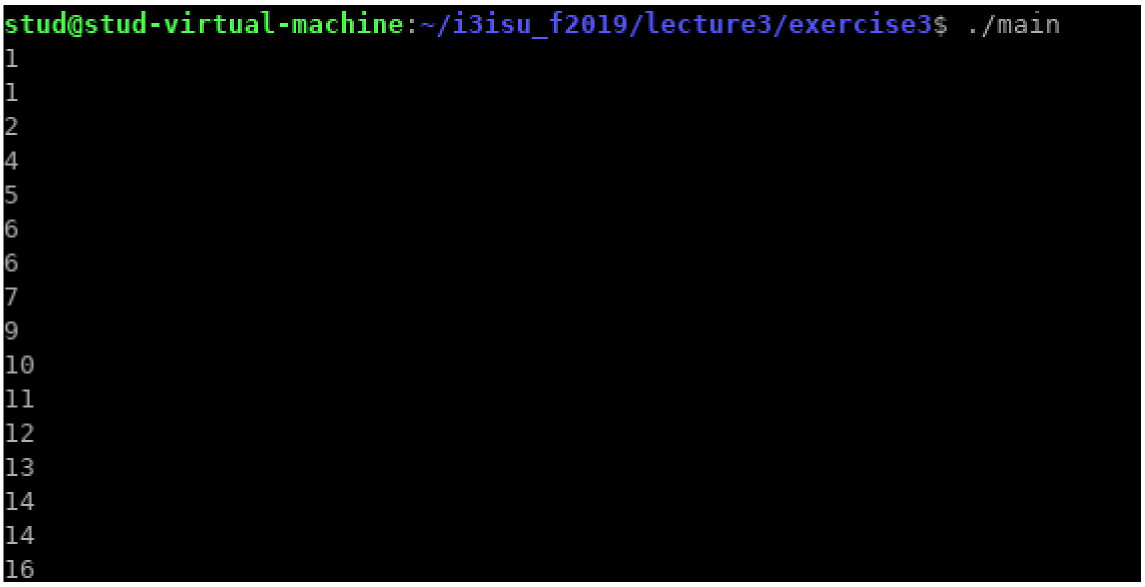
Then the program will close along with the so-called mainthread and the child threads, that will not be executed. This is because main closes quickly before the the function is called.

• 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?

We used a string variable.

Exercise 3 - Sharing data between threads

```
#include <pthread.h>
#include <iostream>
#include <stdlib.h>
#include <string>
#include <unistd.h>

using namespace std;

static unsigned int shared = 0;

void* thrFuncInc(void *ptr)
{
    shared = 0;
    while(1)
    {
        shared++;
        sleep(1);
    }
    return nullptr;
}
void* thrFuncRd(void *ptr)
{
    while(1)
    {
        cout << shared << endl;
        sleep(1);
    }
    return nullptr;
}

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

    pthread_t incr;
    pthread_t read;

    pthread_create(&incr, nullptr, thrFuncInc, nullptr);
    pthread_create(&read, nullptr, thrFuncRd, nullptr);
    pthread_join(incr, nullptr);
    pthread_join(read, nullptr);

    return 0;
}
```



```
stud@stud-virtual-machine:~/i3isu_f2019/lecture3/exercise3$ ./main
1
1
2
4
5
6
6
7
9
10
11
12
13
14
14
16
```

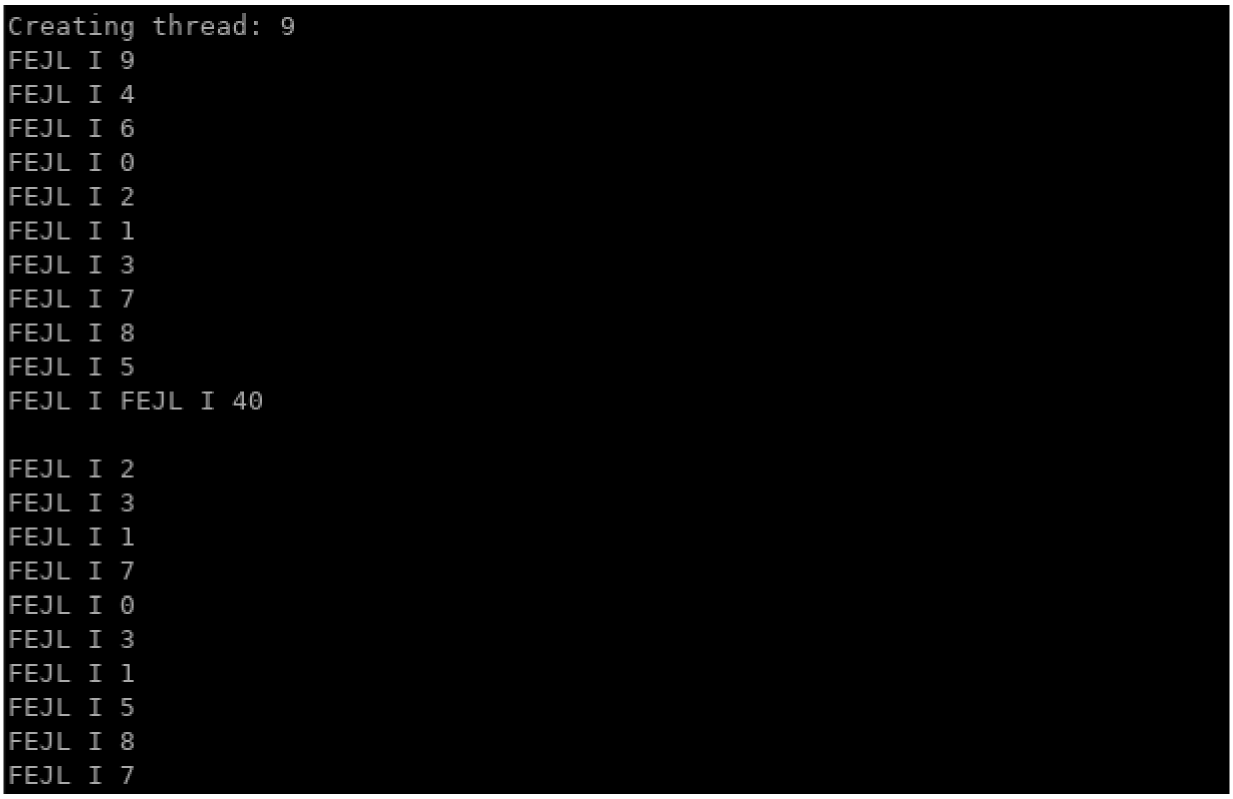
Are there any problems in this program? Do you see any? Why (not)?

When viewing the output to the terminal, we can see that it repeats a number once in a while, and in these cases it skips the following number. This is because sometimes the reader reads 2 times before the incrementer can increment.

Exercise 4 - Sharing a Vector class between threads

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

#define ThrNum 10

using namespace std;

Vector v1;
long delayt;

void * writer(void *ptr)
{
    bool err;
    while(1)
    {
        err = v1.setAndTest((long)ptr);
        if (err == false)
        {
            cout << "FEJL I " << (long)ptr << endl;
        }
        usleep(delayt);
    }
   return nullptr;
}

int main(int argc, char* argv[])
{
    cout << "Skriv delay i mikrosekunder" << endl;
    cin >> delayt;

    pthread_t wr[ThrNum];

    for(int i=0;i<ThrNum;i++)
    {
        pthread_create(&wr[i], nullptr, writer, (void*)(intptr_t)i);
        cout << "Creating thread: " << i << endl;
    }
    for(int i=0;i<ThrNum;i++)
    {
       pthread_join(wr[i], nullptr);
    }
    return 0;
}
```

Nedenfor ses koden eksekveret:



```
Creating thread: 9
FEJL I 9
FEJL I 4
FEJL I 6
FEJL I 0
FEJL I 2
FEJL I 1
FEJL I 3
FEJL I 7
FEJL I 8
FEJL I 5
FEJL I FEJL I 40

FEJL I 2
FEJL I 3
FEJL I 1
FEJL I 7
FEJL I 0
FEJL I 3
FEJL I 1
FEJL I 5
FEJL I 8
FEJL I 7
```

Do your writers detect any problems?

The writer detects more problems the more threads we initialize.

Are there any problems in this program?

The program has a problem when the threads are working at the vector at the same time.

Do you see them?

Yes

Why do you (not) see them?

Because we use multible threads at the same data.

Exercise 5 -Tweaking parameters

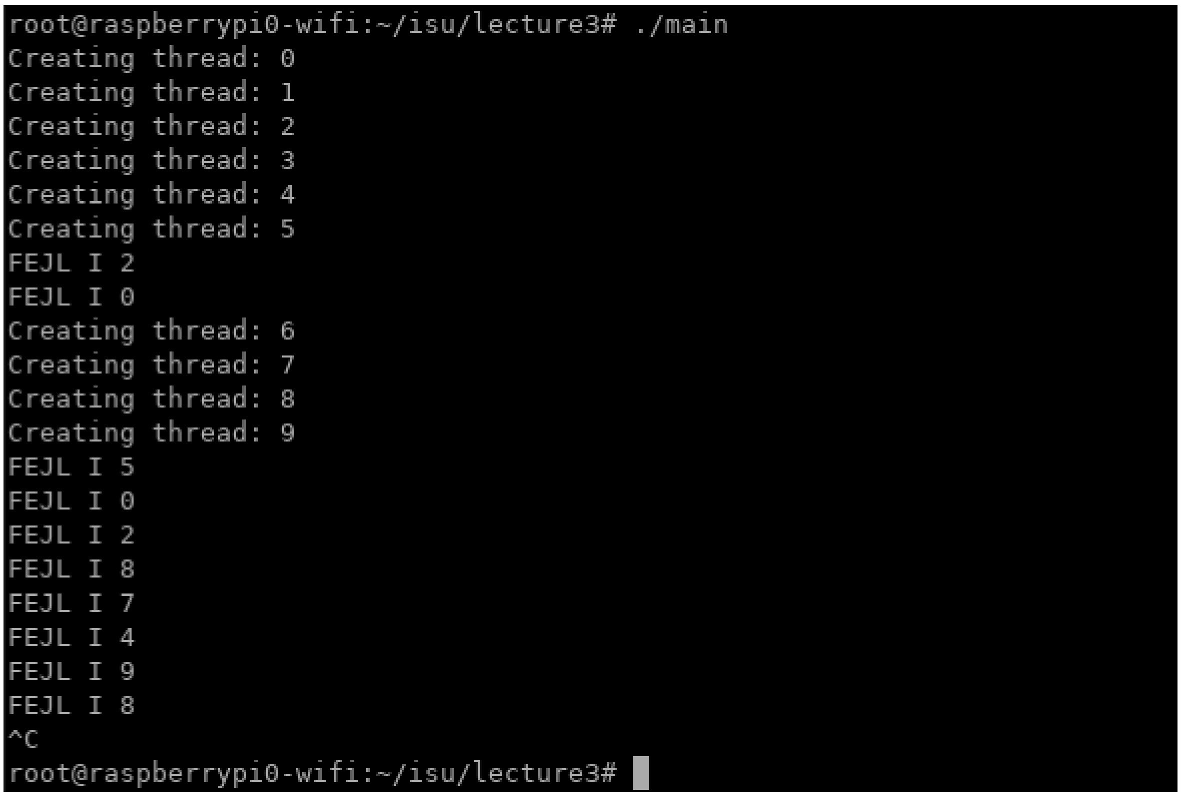
```
long delayt;                            // Variabel delayt initieres

void * writer(void *ptr)
{
    bool err;
    while(1)
    {
        err = v1.setAndTest((long)ptr);
        if (err == false)
        {
            cout << "FEJL I " << (long)ptr << endl;          // pthread_self()
        }
        usleep(delayt);         // Ændring til usleep(delayt)
    }
   return nullptr;
}
```

Do you see any problems?
Explain when and why they start to occur, and why you did not see them in exercise 4

When we use delays of micro seconds instead of 1 second as we did in exercise 4, we experience a higher quantity of errors because the probability of two threads changing and reading at the same time rises.

Exercise 6 - Testing on target

Nedenfor vises eksekvering af Exercise 4 på target:



```
root@raspberrypi0-wifi:~/isu/lecture3# ./main
Creating thread: 0
Creating thread: 1
Creating thread: 2
Creating thread: 3
Creating thread: 4
Creating thread: 5
FEJL I 2
FEJL I 0
Creating thread: 6
Creating thread: 7
Creating thread: 8
Creating thread: 9
FEJL I 5
FEJL I 0
FEJL I 2
FEJL I 8
FEJL I 7
FEJL I 4
FEJL I 9
FEJL I 8
^C
root@raspberrypi0-wifi:~/isu/lecture3#
```

Compare your findings with those in that of exercise 5.
Are the parameters that you found to present problems on the host the same that yield problems on the target?

Yes, the amount of problems on target are also altered by the amount of Threads. We did however experience fewer problems on the target than the host.

Why do you experience what you do?

- **Tilføjet af Sahand Matten for cirka en måned siden**

Da der ikke var givet learning goals har vi valgt at reviewe udfra vores egne learning goals.

Learning goals.

1 Understanding the basics of threads.

Vi kan se en forståelse for Threads under exercise 1, som de har besvaret.

2 Learning to create threads and utilize them to execute functions.

Vi kan se at der er lavet programmer der bruger threads til at udfører en opgave. Der er dog ikke nogle forklaringer til hvad de har gjort. Kunne godt bruge en lille forklaringen i tekst eller nogle kommentar under koden.

3 Sharing data between several threads.

Vi kan se billeder af hvad vi formøder er output og noget kode. Der er ikke nogen forklarende tekst eller kommentar under kode.

4 Understanding what affects thread performance and their ability to execute properly.

De fleste spørgsmål der omfatter Thread performance er blevet besvaret.

Must Haves

relevant files in repository ✗

Makefile in repository ✓

Konklusion.

Der er nogle mangler i relevant kode for de sidste exercises, derudover er der ikke besvarende tekst eller kommentar i kode. Dette gøre det svære at vurder jeres forståelse. Mangler billed beskrivelse. Til næste gang kunne der skrives lidt tekst til billeder og kode sådan man kan se jeres forståelse.

Vi føler ikke vi er i stand til at vurder ordenligt, og hvad vi kunne vurder er der betydelige mangler.

pthread1_hello.png (10,8 KB) Andreas Ventzel, 2019-02-26 13:12
thread2_executed.png (24,8 KB) Andreas Ventzel, 2019-02-26 13:20
pthread3_executed.png (7,96 KB) Andreas Ventzel, 2019-02-26 13:33
pthread4_target.png (19,4 KB) Andreas Ventzel, 2019-02-26 15:58
pthread4_executed.png (9,31 KB) Andreas Ventzel, 2019-02-26 16:23