Posix Threads

Læringsmål for øvelsen:

1. Oprette en ny thread
2. Oprette flere threads som deler en variabel
3. Teste fejl/opførsel når n threads deler et Vector object med x elementer
4. Teste efter den samme fejl/opførsel på Rpi(target)

Exercise 1 - Creating Posix Threads

Indledning

"This exercise consists of two different parts. The first being answering different questions that will aid in part two, in which a simple thread is to be created."

Exercise 1.1 Fundamentals

Questions

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

```
#include <pthread.h>
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
void *(*start)(void *), void *arg);
Returns 0 on success, or a positive error number on error
```

source: M. Kerrisk, The Linux Programming Interface, s.622

- **Which arguments does it take and what do they represent?**

  ```
  void *(*start)(void *),
  ```

  Den nye thread kalder start funktionen med void* som argument. Den originale thread fortsætter.
  (void*) gør det muligt at benytte en object pointer som argument i start funktionen.
  Ønskes der flere argumenter til start funktionen kan der benyttes en pointer til en struct, med de ønskede argumenter.
  pthread_t *thread, peger(points) til en buffer hvor identificering af denne thread kopieres til.
  pthread_attr_t *attr, hvor attr peger(point) på et pthread_attr_t object hvor atributter for den nye thread specificeres. Er attr istedet NULL, vil den nye thread have en række default attributes.

- **What happens when a thread is created?**
  Når der er lavet en ny thread, er det uvist hvilken thread som er den næste til at benytte CPU. Er det nødvendigt at håndhæve rækkefølgen skal der benyttes synchronization techniques.

- **What is a function pointer?**
  som med int* eller char*, kan en funktion have et argument med en pointer til en anden funktion. Som *start ovenfor beskrevet i pthread_create().

- **A function is to be supplied to the aforementioned function (thread creating function):**
  - ***Which argument(s) does it take?***
  som beskrevet ovenfor vil start funktionen have void* som argument, som kan være en pointer til et object eller hvis nødvendigt en struct.
  - ***What is the return type/value?*** _
  start funktionen returnere en void, som ligeledes kan pege på et object eller struct.*
  - **What can they be used for and how?** _
  Retur værdien kan benyttes til pthread_exit og pthread_join():

  ```
  include <pthread.h>
  void pthread_exit(void *retval);
  ```

```
    include <pthread.h>
    int pthread_join(pthread_t thread, void **retval);
```

Exercise 1.2 First threading program

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

Solution

Der udarbejdes et simpelt program hvor pthread_create() kaldes med argumentet threadFunc() som udskriver "Hello World".

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

static void *threadFunc(void *arg)
{
    printf("Hello World\n");
    return NULL;
}

int main()
{
    pthread_t t1;
    int s;

    s = pthread_create(&t1, NULL, threadFunc,NULL ); //default attributes NULL
    if (s!=0){ //return 0 on succes
        printf("pthread_create failed\n");
    }

    pthread_join(t1, NULL);

    return 0;

}
```

Result

Programmet bygges og eksekveres:

```
stud@stud-virtual-machine:~/isu_work/exercise3_posix_threads$ gcc -o Pthread_Hello_world hello_wor
ld.c -lpthread
stud@stud-virtual-machine:~/isu_work/exercise3_posix_threads$ ./Pthread_Hello_world
Hello World
```

Exercise 2 - Two threads

Indledning

"Write a program that creates two threads. When created, the threads must be passed an ID which they will print to stdout every second along with the number of times the thread has printed to stdout. When the threads have written to stdout 10 times each, they shall terminate. The main() function must wait for the two threads to terminate before continuing."

Solution

Løsningen implementeres ved at modificere det første program, så der nu oprettes 2 threads, hvor *threadFunc() får et unikt id som argument for de 2 threads.

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

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

static void *threadFunc(void *arg)
{
    for (int i = 1; i <= 10; i++)
    {
        printf("Thread: %ld  - Thread ID %i  -  Print Nr: %i\n",pthread_self(), *(int*)arg ,i);
        sleep(1);
    }
    return arg;
}

int main()
{
    pthread_t t1, t2;
    int thread1, thread2;

    int ID1 = 1;
    int ID2 = 2;

    thread1 = pthread_create(&t1, NULL, threadFunc, &ID1); //default attributes NULL
    if (thread1!=0){ //return 0 on succes
        printf("pthread_create failed\n");
    }

    thread2 = pthread_create(&t2, NULL, threadFunc, &ID2); //default attributes NULL
    if (thread2!=0){ //return 0 on succes
        printf("pthread_create failed\n");
    }

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    return 0;

}
```

Result

Og resultatet efter at bygge og eksekvere programmet:

```
stud@stud-virtual-machine:~/isu_work/exercise3_posix_threads$ ./pthread_two_threads
Thread: 140582442292800  - Thread ID 1  -  Print Nr: 1
Thread: 140582433900096  - Thread ID 2  -  Print Nr: 1
Thread: 140582433900096  - Thread ID 2  -  Print Nr: 2
Thread: 140582442292800  - Thread ID 1  -  Print Nr: 2
Thread: 140582433900096  - Thread ID 2  -  Print Nr: 3
Thread: 140582442292800  - Thread ID 1  -  Print Nr: 3
Thread: 140582433900096  - Thread ID 2  -  Print Nr: 4
Thread: 140582442292800  - Thread ID 1  -  Print Nr: 4
Thread: 140582442292800  - Thread ID 1  -  Print Nr: 5
Thread: 140582433900096  - Thread ID 2  -  Print Nr: 5
Thread: 140582433900096  - Thread ID 2  -  Print Nr: 6
Thread: 140582442292800  - Thread ID 1  -  Print Nr: 6
Thread: 140582433900096  - Thread ID 2  -  Print Nr: 7
Thread: 140582442292800  - Thread ID 1  -  Print Nr: 7
Thread: 140582442292800  - Thread ID 1  -  Print Nr: 8
Thread: 140582433900096  - Thread ID 2  -  Print Nr: 8
Thread: 140582442292800  - Thread ID 1  -  Print Nr: 9
Thread: 140582433900096  - Thread ID 2  -  Print Nr: 9
Thread: 140582442292800  - Thread ID 1  -  Print Nr: 10
Thread: 140582433900096  - Thread ID 2  -  Print Nr: 10
stud@stud-virtual-machine:~/isu_work/exercise3_posix_threads$
```

Der noteres ud fra ovenstående at den samme thread enkle gange printer to gange i træk, hvorefter den anden thread er sheduled.
Questions

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

Så bliver der ikke udskrevet og de 2 threads bliver nedlagt igen og programmet afsluttes. I implementeringen benyttes sleep(1) hvilket medfører at de 2 threads tilnærmelsesvis følges ad og nedlægges sammen til sidst.

- **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?**

Der er i implementeringen benyttet en pointer.

Exercise 3 - Sharing data between threads

Indledning

"Create a program that creates two threads, incrementer and reader. The two threads share an unsigned integer variable named shared which is initially 0. incrementer increments shared every second while reader reads it every second and outputs it to stdout."

Solution
Programmet modificeres nu således at der er forskellige start funktioner, når de 2 threads bliver oprettet. Istedet for et ID er der nu en pointer til "shared" - mens den ene thread incrementere vil den anden udskrive.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

static void *threadFunc1(void *arg)
{
    for (int i = 1; i <= 10; i++)
    {
        *(unsigned int*)arg = i;
        sleep(1);
    }
    return arg;
}

static void *threadFunc2(void *arg)
{
    for (int i = 1; i <= 10; i++)
    {
        printf("Thread Reader - Current value of shared:  %i\n", *(unsigned int*)arg);
        sleep(1);
    }
    return arg;
}

int main()
{
    pthread_t incrementer, reader;
    int thread1, thread2;

    unsigned int shared = 0;

    thread1 = pthread_create(&incrementer, NULL, threadFunc1, &shared); //default attributes NULL
    if (thread1!=0){ //return 0 on succes
        printf("pthread_create failed\n");
    }

    thread2 = pthread_create(&reader, NULL, threadFunc2, &shared); //default attributes NULL
    if (thread2!=0){ //return 0 on succes
        printf("pthread_create failed\n");
    }

    pthread_join(incrementer, NULL);
    pthread_join(reader, NULL);

    return 0;

}
```

Result

```
stud@stud-virtual-machine:~/isu_work/exercise3_posix_threads$ ./pthread_shared
Thread Reader - Current value of shared:  1
Thread Reader - Current value of shared:  1
Thread Reader - Current value of shared:  3
Thread Reader - Current value of shared:  4
Thread Reader - Current value of shared:  5
Thread Reader - Current value of shared:  6
Thread Reader - Current value of shared:  7
Thread Reader - Current value of shared:  8
Thread Reader - Current value of shared:  9
Thread Reader - Current value of shared:  10
stud@stud-virtual-machine:~/isu_work/exercise3_posix_threads$ ./pthread_shared
Thread Reader - Current value of shared:  1
Thread Reader - Current value of shared:  2
Thread Reader - Current value of shared:  2
Thread Reader - Current value of shared:  3
Thread Reader - Current value of shared:  4
Thread Reader - Current value of shared:  5
Thread Reader - Current value of shared:  6
Thread Reader - Current value of shared:  8
Thread Reader - Current value of shared:  9
Thread Reader - Current value of shared:  10
stud@stud-virtual-machine:~/isu_work/exercise3_posix_threads$ ./pthread_shared
Thread Reader - Current value of shared:  1
Thread Reader - Current value of shared:  2
Thread Reader - Current value of shared:  3
Thread Reader - Current value of shared:  4
Thread Reader - Current value of shared:  5
Thread Reader - Current value of shared:  6
Thread Reader - Current value of shared:  7
Thread Reader - Current value of shared:  8
Thread Reader - Current value of shared:  9
Thread Reader - Current value of shared:  10
stud@stud-virtual-machine:~/isu_work/exercise3_posix_threads$
```

Questions

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

Som der vises ovenfor er resultatet varierende. Dette skyldes at der arbejdes med den samme variabel to steder. Ligesom i tidligere opgave kan den ene thread shedules to gange i træk. I dette program vil det medføre at der bliver incrementeret eller udskrevet 2 gange i træk. I første eksempel kommer det til udtryk ved at 2 aldrig bliver udskrevet.

Exercise 4 - Sharing a Vector class between threads

Indledning

"Create a thread function writer that uses Vector::setAndTest() to set and test the value of a shared Vector object. Then create a main() function that creates a user-defined number of writer threads (between 1 and 100), each with their own unique ID. Let each writer set and test the shared Vector object to its ID every second. If a writer detects an inconsistency in the shared Vector object (i.e. setAndTest() returns false), it should write an error message."

Solution

Der laves et program hvor et loop benyttes til at lave N_THREADS. Hver thread får et unikt ID, som bruges i writer funktionen hvor et fælles Vector object benytter setAndTest(ID) indtil der opstår fejl.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <iostream>
#include "/home/stud/isu_work/exercise3_posix_threads/Vector.hpp"

#define N_THREAD 20
using namespace std;

Vector shared;

void *writer(void *arg)
{

    while(shared.setAndTest(*(int*)arg)){
        sleep(1);
    }

        printf("FAILED test - Writer ID %i\n",*(int*)arg);

    return arg;
}

int main()
{
    printf("Enter number of new Threads (0-100)\n");
    int n;
    std::cin >> n;

    pthread_t thread[n];
    int t;

    int err;

    for(int i = 1; i<= n; i++)
    {
        printf("New thread: %i\n", i);
        t = pthread_create(&thread[i], NULL, writer, &i);
        if (t!=0){ //return 0 on succes
        printf("pthread_create failed\n");
        }
        sleep(2);

    }

    //sleep(5);

    printf("Terminate thread: \n");
    for(int i = 1; i<= n; i++)
    {
        err = pthread_join(thread[i], NULL);
        if (err == 0){ //return 0 on succes
        printf("pthread %i - Terminates\n",i);
        }
    }

    return 0;

}
```

Result

```
stud@stud-virtual-machine:~/isu_work/exercise3_posix_threads$ ./pthread_shared_vect
Enter number of new Threads (0-100)
5
```

Benyttes der kun 5 threads opstår der ikke problemer:

```
stud@stud-virtual-machine:~/isu_work/exercise3_posix_threads$ ./pthread_shared_vect
New thread: 1
New thread: 2
New thread: 3
New thread: 4
New thread: 5
```

Men som N_THREADS øges, begynder der at opstå fejl:

```
New thread: 1
New thread: 2
New thread: 3
New thread: 4
New thread: 5
New thread: 6
FAILED test - Writer ID 6
New thread: 7
FAILED test - Writer ID 7
New thread: 8
New thread: 9
FAILED test - Writer ID 9
New thread: 10
New thread: 11
FAILED test - Writer ID 11
New thread: 12
New thread: 13
FAILED test - Writer ID 13
FAILED test - Writer ID 12
New thread: 14
New thread: 15
New thread: 16
New thread: 17
New thread: 18
New thread: 19
New thread: 20
FAILED test - Writer ID 21
FAILED test - Writer ID 21
Terminate thread: 
```

Questions

- **Do your writers detect any problems?**

Når antallet af N_THREADS øges begynder der at opstå problemer. Dette skyldes de mange threads der kalder setAndTest på et fælles object.

- **Are there any problems in this program? Do you see them? Why do you (not) see them?**

Indsættes der en sleep() i loopet hvor der laves writer threads, synes problemet at blive mindre.

```
stud@stud-virtual-machine:~/isu_work/exercise3_posix_threads$ ./pthread_shared_vect
New thread: 1
New thread: 2
New thread: 3
New thread: 4
New thread: 5
New thread: 6
New thread: 7
New thread: 8
New thread: 9
New thread: 10
New thread: 11
New thread: 12
New thread: 13
New thread: 14
FAILED test - Writer ID 14
New thread: 15
New thread: 16
New thread: 17
New thread: 18
New thread: 19
New thread: 20
Terminate thread:
```

Exercise 5 - Tweaking parameters

Indledning

"Modify your program from exercise 4 so that the writers loop time is no longer one second but a user-defined number of microseconds. Experiment with the number of writers created and shorter loop time as well as the number of elements in the Vector."

Solution

Programmet fra opgave 4, modificeres således at loop time i writer funktionen vælges med user input.

```
printf("Enter number of ms between setAndTest\n");

    std::cin >> wait;

 while(shared.setAndTest(*(int*)arg)){
        usleep(wait);
.
.
.
```

Result

```
stud@stud-virtual-machine:~/isu_work/exercise3_posix_threads$ ./Pthread_shared_vect_ext
Enter number of new Threads (0-100)
10
Enter number of ms between setAndTest
200
```

Questions

- **Determine when they occur when altering values - if you did or didn't see them in exercise 4, then why do you think so?**

Ændres number of elements i Vector objektet til eksempelvis 500 fremfor 10.000 opleves færre fejl.
-Dette antages at skyldes at ved setAndTest med de 10.000 elementer, tager det længere tid end ved de 500 og en ny thread med setAndTest kan nå at skabe komplikationer og derved fejl.

Ved at loop time i writer funktionen sættes mindre opstår der ligeledes flere fejl.

Exercise 6 - Testing on target

Indledning

"Recompile the solution from exercise 4 and test it on target following the same line of thinking as in exercise 5. Compare your findings with those in that of exercise 5."

Solution
Programmet bygges nu med arm-rpizw-g++ compileren og kopieres til target med scp.

Result

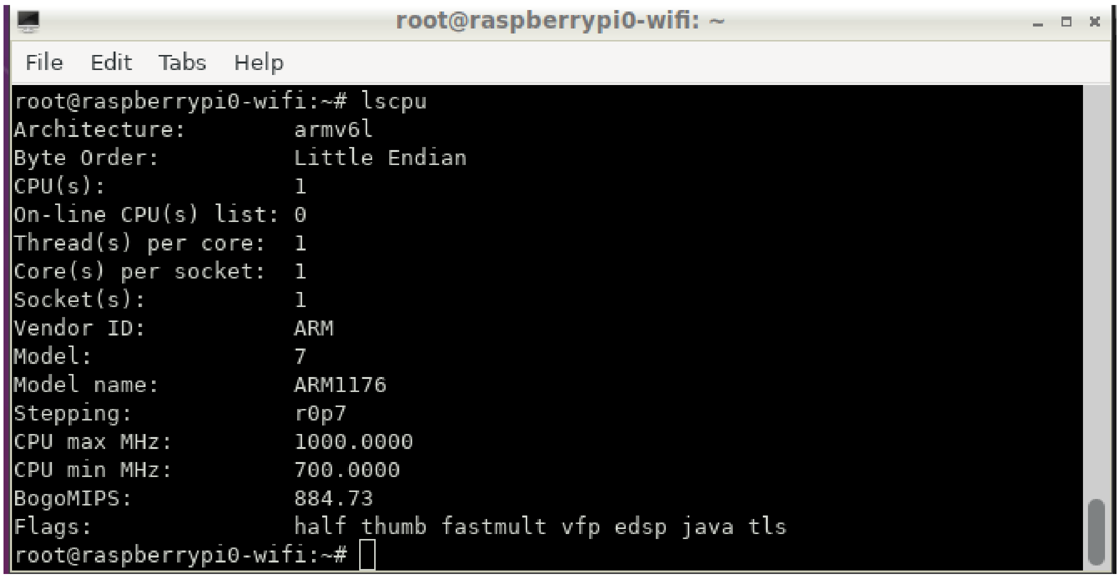
```
stud@stud-virtual-machine:~/isu_work/exercise3_posix_threads$ arm-rpizw-g++ -o Pthread_test_on_tar
get  share_vector.cpp -lpthread
stud@stud-virtual-machine:~/isu_work/exercise3_posix_threads$ scp Pthread_test_on_target root@Rpi:
Pthread_test_on_target                          100%   17KB   2.2MB/s   00:00
```

Questions

- **Are the parameters that you found to present problems on the host the same that yield problems on the target? Why do you experience what you do?**

På target opnåes det bedste resultat ved få threads og stor wait time i writer funktionen.
Det forekommer at der på applikation sker flere fejl, på trods af der benyttes samme parametre.

-Forskellen antages at skyldes at CPU og sheduler er forskellige, pc/Rpi, og derfor afvikles applikationen forskelligt.
Nedenfor vises Rpi CPU informationer:



```
root@raspberrypi0-wifi:~# lscpu
Architecture:        armv6l
Byte Order:          Little Endian
CPU(s):              1
On-line CPU(s) list: 0
Thread(s) per core:  1
Core(s) per socket:  1
Socket(s):           1
Vendor ID:           ARM
Model:               7
Model name:          ARM1176
Stepping:            r0p7
CPU max MHz:         1000.0000
CPU min MHz:         700.0000
BogoMIPS:            884.73
Flags:               half thumb fastmult vfp edsp java tls
root@raspberrypi0-wifi:~#
```

```
root@raspberrypi0-wifi:~# lscpu
Architecture:        armv6l
Byte Order:          Little Endian
CPU(s):              1
On-line CPU(s) list: 0
Thread(s) per core:  1
Core(s) per socket:  1
Socket(s):           1
Vendor ID:           ARM
Model:               7
Model name:          ARM1176
Stepping:            r0p7
CPU max MHz:         1000.0000
CPU min MHz:         700.0000
BogoMIPS:            884.73
Flags:               half thumb fastmult vfp edsp java tls
```

Filer

proc.PNG	30,4 KB	30.09.2021	Rasmus Baunvig Aagaard