GWDG
AG-C
Lars Quentin

Exercise 1 / April 07, 2026
Practical Course High-Performance Computing / SoSe 2026
60 Minutes Total

Exercise Introduction

This exercise accompanies the tutorial on pthreads. The slides and code snippets are found on the homepage. The exercises build on each other. The exercises are designed s.t. you can either use a given source code and complete the exercises with that, or you can implement it yourself.

Learning Objectives

After completing the exercise sheet, you should be able to:

- compile and run a pthread program
- know how to spawn and join threads with pthreads
- know what a critical section in a shared memory program is and how to handle it with mutexes and semaphores

Contents

Task 1: Shared Memory (10 min)

What needs to be kept in mind in shared memory programming?

Task 2: Hello World (10 min)

1. Take a look at `pth_hello.c`.
   a) Identify the thread function. Where does the function get the value of `thread_count` from?
   b) What is special about the variable `thread_count`?
2. Compile and run the program multiple times with different thread counts. What do you see? Compile it with `gcc -g -Wall -o pth_hello pth_hello.c -lpthread` and run it with `./pth_hello <num_threads>`.

Task 3: Estimation of π 1/2 (10 min)

```
int n = 100, i;
double factor = 1.0;
double sum = 0.0, pi;
for (i = 0; i < n; i++, factor = -factor) {
    sum += factor/(2*i+1);
}
pi = 4.0*sum;
```

1. What steps do you need to take to parallelize the above code snippet with pthreads?

Task 4: Estimation of π 2/2 (20 min)

1. Try to parallelize the code snippet above yourself. Use `pth_pi_skeleton.c` for some guidelines if you do not want to try it all by yourself. If you do create the source code from scratch, please consider the following:
   - It should take the number of threads and `n` as input.
   - Add print statements in the thread function, which print the thread rank, the current value of the thread private sum (`my_sum`) and the current value of the global sum (`sum`).
   - Add a print statement for the global sum after joining the threads.
   - Add a print statement for the estimation of π.

   Sample Pseudo Output

   ```
   > ./pth_pi 4 100
   [rank] my_sum: <value>
   [rank] sum: <value>
   [rank] my_sum: <value>
   [rank] sum: <value>
   [rank] my_sum: <value>
   [rank] sum: <value>
   [rank] my_sum: <value>
   [rank] sum: <value>
   Sum after join: <value>
   Estimation of PI: <value>
   ```

2. Compile and run `pth_pi.c` or your program. Run it with: `./pth_pi <num threads> <n>`
   a) What changes, when you change the number of threads? E.g., try 1, 2, 4, 8.
   b) What changes, when you change `n`? E.g., try 8, 100, 200, 1000
   c) Run the program multiple times with 4 threads and $n = 100$. Is the output always the same?

Task 5: Mutexify the estimation of π (10 min)

1. Take your previous code and make it safe with Mutexes.
2. Compile and run your program with different numbers of threads and different values of `n`. `./pth_pi_mutex <num threads> <n>`
   a) Run the program multiple times with 4 threads and $n = 100$. Is the output always the same? What differences do you see compared to the version without mutexes?

Optional Task 6: From mutex to semaphore (15 min)

This is a difficult **additional** task which will support your understanding in the topic.

Take the base code and make it safe with Semaphores.

1. run and compile it several times with four threads. Is the access ordered?
2. What steps do you need to take to order the access to `sum`?
3. Implement the ordered access.

Implementation tip:

- Just spin-lock / busy-wait until its your ranks turn

Further Reading

- `https://man7.org/linux/man-pages/man7/pthreads.7.html`
- `https://man7.org/linux/man-pages/man0/semaphore.h.0p.html`