Tasks

Task 1. Fork example

Process B is the parent process, Process A is the child process because process A gets a process ID of zero.

No it is not affected, both iterates between 0-100. The variable is not affected because the different processes don't share the same source of memory.

The pids ran last time were 89 and 90.

Task 2. Shared memory buffer

See source code

Task 3. Shared memory buffer with semaphores

In task 2 synchronization problems appeared because there was no mutual exclusion between the producer and consumer and so they accessed the same resource at the same time (mainly the amount variable) causing race conditions.

In task 3 the producer and consumer simply used mutual exclusion methods to access the buffer and so once the producer had added a number and was not locking the shared memory the consumer could access the buffer safely with data loss prevention.

Task 4. Message queue modification

See source code

Task 5. pthread_create() example

```
lukase-l@LAPTOP-325PO44P:/mnt/c/Us
./app
This is the parent (main) thread.
This is the child thread.
```

Task 6. pthread_create() example 2

The input to a thread is a void pointer and then casted into the struct type, this needs to happen every time a new thread is used.

on line 31-34:
```
pthread_create(&(children[id]), // our handle for the child
               NULL, // attributes of the child
               child, // the function it should run
               (void*)args); // args to that function
```
void pointer —>

on line 11, casted back to struct: struct threadArgs *args = (struct threadArgs*) params;

Task 7. pthread_create(), how to handle return values

See source code

Task 8. Bank account example

```
lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ ./app 32

The final account balance with 32 threads is $0.00.

lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ ./app 64

The final account balance with 64 threads is $0.00.

lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ ./app 128

The final account balance with 128 threads is $0.00.

lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ ./app 256

The final account balance with 256 threads is $0.00.

lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ ./app 512

The final account balance with 512 threads is $0.00.

lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ ./app 1024

The final account balance with 1024 threads is $0.00.

lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ ./app 2048

The final account balance with 2048 threads is $0.00.
```

We got no bugs but there is no mutual exclusion implemented meaning there can appear race conditions and two or more threads can update the variable at the same time, causing a bug and the final balance might not be 0$ for even number of threads.

Task 9. Bank account, correction

see source code

```
lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ ./app 32

The final account balance with 32 threads is $0.00.

lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ ./app 64

The final account balance with 64 threads is $0.00.

lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ ./app 128

The final account balance with 128 threads is $0.00.

lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ ./app 256

The final account balance with 256 threads is $0.00.

lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$
lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ ./app 512

The final account balance with 512 threads is $0.00.

lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ ./app 1024

The final account balance with 1024 threads is $0.00.

lukase-1@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ ./app 2048

The final account balance with 2048 threads is $0.00.
```

Task 10. Dining Professors, implementation

Four conditions:

1. Circular set:
   There are only 5 chopsticks.
2. Hold and wait:

Each philosopher that takes a chopstick don't give it back once its picked up.

3. Mutual exclusion:
   Each chopstick can only be held by one of the philosophers.
4. No preemption:
   Doesn't give back chopstick even though it cannot get the other to continue to eat.

Task 11. Dining Professors, deadlock-free implementation

We adjusted the hold-and-wait condition. By using pthread's trylock function, to see if a resource was lockable or not, we could solve the previous task's deadlock. If a professor couldn't lock both chopstick then the professor give back the left chopstick already taken and doesn't just hold and wait.

Task 12. Sequential matrix multiplication

```
lukase-l@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ make matmulseq
gcc -O2 -o app matmulseq.c -lpthread
lukase-l@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ time ./app

real    0m9.662s
user    0m9.641s
sys     0m0.000s
```

It took 9.662 seconds

Task 13. Parallel matrix multiplication

```
lukase-l@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ time ./app

real    0m6.012s
user    0m45.953s
sys     0m0.406s
```

The execution time was 6.012 seconds

$$Speedup = 9.662/6.012 = 1.60711909514 \cong 160\% faster$$

Task 14. Parallelization of the initialization

```
lukase-l@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ make matmulseq_init
gcc -O2 -o app matmulseq_init_parallell.c -lpthread
lukase-l@LAPTOP-325PO44P:/mnt/c/Users/lukas/Downloads/Assignments/VT22/OS/DV1629/lab1_code_students_v1.3$ time ./app

real    0m3.049s
user    0m22.469s
sys     0m0.328s
```

It went even faster now that we programmed threads for both multiplication and initialization of the matrices. The speed compared to the original program was

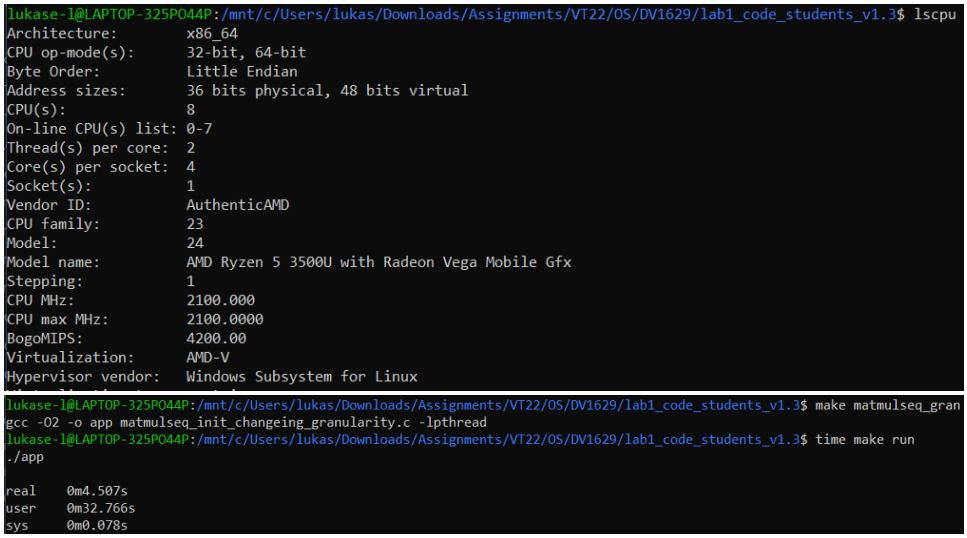
$$Speedup = 9.662/3.049 = 3.16890783864 \cong 317\% faster$$

The program speedup compared to task 13:

$$Speedup = 6.012/3.049 = 1.97179403083 \cong 197\% faster$$

The speedup was greater because the overhead of creating the threads, scheduling them and let them work in parallell was less time consuming than creating the matrices sequentially.

Task 15. Changing granularity



The execution time was 4.507 seconds, The exec time was faster than task 13. 4.507 < 6.012 seconds.

The reason might be because the amount of threads related to the amount of processing cores creates an even balance between the amount of work each thread does and the overhead and scheduling work that it adds in time complexity.