E/15/156
Jayathilaka H.A.D.T.T.

CO327
Operating Systems
Lab 04 : Introduction to Multithreading

Exercise1

1. **Save the above program as thread.c and compile it as: gcc -pthread -o thread thread.c -Wall**

```
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ gcc -pthread -o thread thread.c -Wall
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ ./thread
Thread says hi!
Thread says hi!
Thread says hi!
Thread says hi!
Thread says hi!
Thread says hi!
Thread says hi!
Thread says hi!
Thread says hi!
Thread says hi!
Main thread says hi!
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
```

a) Why do you need '-pthread' flag when you compile?

'-pthread' flag function is to allow one thread to wait for the end of another. The Pthreads standard does not describe what happens when multiple threads call pthread join at the same time. So if we remove the flag we can get errors when compiling.

Exercise2

1. **Consider the following piece of code from multiprocessing lab session:**

```
int i;
for (i = 0; i < 3; i++)
        fork();
```

a) how many new processes did it create?

It created total 8 processes including the parent process

Now consider this piece of code:

```
int i;
for(i = 0; i < 3; i++)
        pthread_create(&myThread,NULL,function,NULL);
```

b) how many threads will it create? Compare this with a) above. Why is there a difference?

This will create total 4 processes including the main thread in the process

Exercise 3

1. **Compile and run the above program as is.**

```
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ gcc -pthread -o thread exercise3.c -Wall
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ ./thread
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:3 says hi!
Thread 2:1 says hi!
Thread 2:2 says hi!
Thread 2:3 says hi!
Thread 3:1 says hi!
Thread 3:2 says hi!
Thread 3:3 says hi!
Thread 4:1 says hi!
Thread 4:2 says hi!
Thread 4:3 says hi!
Thread 5:1 says hi!
Thread 5:2 says hi!
Thread 5:3 says hi!
Main thread says hi!
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
```

a) Can you explain the results?

This process generates 5 threads in a sequential manner. When the value is 1, the first thread receives the reference to the thread number. The value is increased by 1 for each thread. This is how various threads figure out what number they belong to. Individual threads iterate from 0 to increase the sequence number within the thread.

b) What is the objective of the code in line 15?

It's combining pointer dereferencing *some_pointer with casting (some_type)some_value_with_some_other_type. The threads' running function "thread function()" is handed a reference to the memory address with the thread number, which is cast to a void pointer. On the void data type, the increment operation is not possible.

c) Deconstruct this statement, and explain how the objective you mentioned is achieved.

First void* is casted to int*. Then int* is dereferenced to obtain the value. After that int value is incremented.

2. **Comment out the code segment for the join call (lines 30 – 34). Can you explain the result?**

```
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ gcc -pthread -o thread exercise3.c -Wall
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ ./thread
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:3 says hi!
Thread 1:1 says hi!
Thread 2:2 says hi!
Thread 2:3 says hi!
Thread 1:1 says hi!
Thread 3:2 says hi!
Thread 3:3 says hi!
Thread 1:1 says hi!
Thread 4:2 says hi!
Thread 4:3 says hi!
Thread 1:1 says hi!
Thread 5:2 says hi!
Thread 5:3 says hi!
Main thread says hi!
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
```

The number of threads created is still 5. But now the threads work asynchronously. So the threads run in parallel.

a) Now, comment out the sleep() statements at lines 12, 35 and 37 (only one at a time) and explain each result.

Commenting out line 12

```
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ gcc -pthread -o thread exercise3.c -Wall
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ ./thread
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:3 says hi!
Thread 2:1 says hi!
Thread 2:2 says hi!
Thread 2:3 says hi!
Thread 3:1 says hi!
Thread 3:2 says hi!
Thread 3:3 says hi!
Thread 4:1 says hi!
Thread 4:2 says hi!
Thread 4:3 says hi!
Thread 5:1 says hi!
Thread 5:2 says hi!
Thread 5:3 says hi!
Main thread says hi!
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
```

The program behaves as the original. Also there is no delay between all the lines printed for a single thread.

Commenting out line 35

```
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ gcc -pthread -o thread exercise3.c -Wall
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ ./thread
Thread 1:1 says hi!
Thread 1:1 says hi!
Thread 1:1 says hi!
Thread 1:1 says hi!
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:2 says hi!
Thread 1:2 says hi!
Thread 1:2 says hi!
Thread 1:2 says hi!
Main thread says hi!
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
```

There is no time between thread creations. As a result, before at least one thread prints a single line, all threads are initialized with thread count 1. The internal iterations of all five threads will be printed as 1,2,3. Because all threads have the same thread number, it is impossible to tell which one prints first.

Commenting out line 37

```
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ gcc -pthread -o thread exercise3.c -Wall
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ ./thread
Thread 1:1 says hi!
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:3 says hi!
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:3 says hi!
Thread 2:1 says hi!
Thread 2:2 says hi!
Thread 2:3 says hi!
Main thread says hi!
Thread 3:2 says hi!
Thread 3:3 says hi!
Thread 3:3 says hi!
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
```

In here the main thread's message is printed before all the other threads finish printing. That happened because there is no sleep between starting the last thread and the server thread printing it's message.

b) Now, comment out pairs of sleep() statements as follows, and explain the results.

i. lines 35 & 37

```
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ gcc -pthread -o thread exercise3.c -Wall
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ ./thread
Thread 1:1 says hi!
Thread 1:1 says hi!
Thread 1:1 says hi!
Thread 1:1 says hi!
Main thread says hi!
Thread 1:1 says hi!
Thread 1:1 says hi!
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
```

Between the end of thread creation and the message of the main thread, there is no delay.As a result, it is printed asynchronously. There is no time between thread creations. As a result, all 5 threads have the same id 1. The first Hello message is printed on each thread. The exit() function is called without delay in the main thread. Before that, none of the other threads make it to the second iteration. As a result, just 5 1.1 messages are shown.

ii. lines 12 & 35 (run the program multiple times, and explain changes in the results, if any)

```
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ gcc -pthread -o thread exercise3.c -Wall
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ ./thread
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:3 says hi!
Thread 1:1 says hi!
Thread 2:2 says hi!
Thread 2:3 says hi!
Thread 1:1 says hi!
Thread 3:2 says hi!
Thread 3:3 says hi!
Thread 1:1 says hi!
Thread 4:2 says hi!
Thread 4:3 says hi!
Thread 1:1 says hi!
Thread 5:2 says hi!
Thread 5:3 says hi!
Main thread says hi!
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
```

The last line is always the same but thread result is changed because individual threads print asynchronously.

iii. lines 12 & 37 (increase the sleep time of line 37 gradually to 5)

```
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ gcc -pthread -o thread exercise3.c -Wall
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ ./thread
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:3 says hi!
Thread 2:1 says hi!
Thread 2:2 says hi!
Thread 2:3 says hi!
Thread 3:1 says hi!
Thread 3:2 says hi!
Thread 3:3 says hi!
Thread 4:1 says hi!
Thread 4:2 says hi!
Thread 4:3 says hi!
Thread 5:1 says hi!
Thread 5:2 says hi!
Thread 5:3 says hi!
Main thread says hi!
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
```

Because 37 has already been commented on, we are unable to increase it. While changing line 35 from sleep(1) to sleep(2), the code was tested (5). The output is identical in both cases. Only if the time it takes to create and execute a single thread is longer than 1 second will the output alter. However, the computer takes a fraction of the time.

b) Now, uncomment all sleep() statements.

```
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ gcc -pthread -o thread exercise3.c -Wall
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$ ./thread
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:3 says hi!
Thread 1:1 says hi!
Thread 1:1 says hi!
Thread 1:3 says hi!
Thread 2:2 says hi!
Thread 2:1 says hi!
Thread 2:2 says hi!
Thread 3:3 says hi!
Thread 3:2 says hi!
Thread 3:3 says hi!
Thread 4:3 says hi!
Main thread says hi!
thiwanka@DESKTOP-UTHF9FV:/mnt/f/Engineering/Third year/Sixth semester/CO327 Operating Systems/Lab/Lab4$
```

i. Can you explain the results?

Because there is just a 1 second wait between thread creation and main thread writing, the main thread may finish printing its message before the other threads. Because join() blocking does not connect the threads, they are asynchronous. The thread count variable can be used by several threads.

ii. Does the output change if you revert the sleep value in line 37 back to 1? If so, why?

Yes. Now consistently the main thread's message comes in the last line.

3. Consider the following statement: "you use sleep() statements instead of join calls to get the desired output of a multithreaded program."

a) Write a short critique of this statement expressing your views and preferences, if any.

Thread.sleep() method can be used to pause the execution of current thread for specified time in milliseconds. Thread.sleep() communicates with the thread scheduler to place the current thread in a waiting state for a set amount of time. When the wait time is up, the thread state is changed to runnable, and it waits for the CPU to finish the job. As a result, the length of time that the current thread sleeps is determined by the operating system's thread scheduler.

If we have an estimate for the pieces of code to run, we can use sleep to imitate practically anything that can be done with join. It's impossible to know for sure how long these times will last. In order to reduce uncertainty, longer sleep() periods must be introduced. However, this is a complete waste of time. With join, you can get the same result in a short amount of time.

Exercise 4

1. **Use the following skeleton code to implement a multi-threaded server.**
   Code : exercise4.c

2. **Why do you need to declare connfd as a variable in the heap? What problem might occur if it was declared as a local variable?**
   The connection file descriptor is represented by confd. The thread should make a copy of the file descriptor value before the following server loop iteration overwrites the stack memory address when confd points to a position in stack.
   This can lead to racial tensions.