CO327: Lab 03

E/15/140 (Jaliyagoda A.J.N.M.)

Exercise 1

1. Save the above program as thread.c and compile it as:

gcc -pthread -o thread thread.c -Wall

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

-pthread is a flag in GCC to inform the compiler to link the POSIX Threads library while compiling.

Exercise 2

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

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

how many new processes did it create?

= 1 + 2 + 4
= 7 child processes

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?

Only 3 threads created since the original process isn't duplicated or cloned in this case.
Each thread executes the code in the given function, not continue the code from the point it created like in fork.

Exercise 3

Assumptions:

- sleep(1) statement is explained as a '1 second' delay in the below explanations.
- 'comment out' means comment the line by adding // into the beginning of the line (please find the answers_e15140_old file for answering to the questions taken '*comment out*' as *'uncomment'* instruction)

Compile and run the above program as is.

a) Can you explain the results?

The result is as below:

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!

The for loop in the main function will call the thread_function as separate threads by passing the 'count' variable as a reference. Each thread prints a specific sentence with the count value 3 times and increases the count value by one at the end of each thread.

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

increase the value in arg[0] (in the main thread, the variable named 'count') by one.

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

(int *)arg - This is the value of the first arg, which is passed to the function as a pointer/ arg[0]
*(int *)arg) - The value in the pointer, (int *)arg
(*(int *)arg)++ - increase the value in the memory location, '(int *)arg' by one.

Summary:
Code in the line 15, (**(*(int *)arg)++;**) reading the pointer value in its provided arguments list (pointer) and increasing the value stored in that pointer's memory location by one.

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

Result:

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!
Main thread says hi!
Thread 4:1 says hi!
Thread 4:2 says hi!
Thread 4:3 says hi!

Since we comment out the 'pthread_join' section, the main thread loop will not wait until all the threads are completed and return. It will create all 5 threads and execute the print statement in the line 38, and exit from the main() and terminate the process.

Meanwhile, 3 or 4 threads can complete the execution and print some results into the std out, before the main thread terminates the process. (The result may depend on the processor specifications, process scheduling algorithm of the operating system and other processes running at that time)

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

The result after comment the line 12 (line 35, 37 are not commented):

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!

Now there are no delay statements inside the thread function. So each thread executes as soon as it is scheduled and prints three lines to StdOut and exits from the thread. The main thread waits 1 second after creating a thread, so the wait is enough to keep the functionality the same as the use of thread_join function.

The result after comment the line 35 (line 12, 37 are not commented):

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!
Main thread says hi!
Thread 1:2 says hi!

Now each thread function is taking 3 seconds to print all 3 lines into StdOut and increase the count variable on the main thread by one. But since line 35 is commented, there is no significant time gap between the creation of each thread. (we can say that all threads execute nearly parallely)

So some of the threads are given only a few times (nearly one second) to complete the execution before the main thread (and the process) terminate. And there is very less possibility to complete the execution of a thread function and increase the 'count' variable in the main thread by one.

The result after comment the line 37 (line 12, 35 are not commented):

Thread 1:1 says hi!
Thread 1:2 says hi!
Thread 1:1 says hi!
Thread 1:3 says hi!
Thread 1:2 says hi!
Thread 1:1 says hi!
Thread 2:3 says hi!
Thread 2:2 says hi!
Thread 2:1 says hi!
Thread 3:3 says hi!
Thread 3:2 says hi!
Thread 3:1 says hi!
Main thread says hi!

Now there is a time gap between each thread creation, and each thread has a 1 second delay between each print statement. So at least a few threads can complete the execution and increase the 'count' variable by one. Since line 37 is commented, the main thread will terminate the process as soon as it prints the statement, "*Main thread says hi!*"

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

i. lines 35 & 37 (Line 12 isn't commented)

Result:

Thread 1:1 says hi!
Thread 1:1 says hi!
Thread 1:1 says hi!
Main thread says hi!

Now there is no time delay between each thread creation as well as no wait time after the loop. So only the first few threads can print their first statement, before the terminate of the process

ii. lines 12 & 35 (run the program multiple times, and explain changes in the results, if any)
(Line 37 isn't commented)

Result:

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!*	*Thread 1:1 says hi!* *Thread 1:1 says hi!* *Thread 1:2 says hi!* *Thread 1:3 says hi!* *Thread 1:2 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!*	*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!*

There are no changes observed. Since there is no delay between the thread creation and no delays inside the thread functions, it is possible to execute all threads in nearly parallel. But changes in the print order can happen due to thread scheduling order on different processors, different operating systems.

iii. lines 12 & 37 (increase the sleep time of line 37 gradually to 5)
(Line 35 isn't commented)

Result:

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!

There is a time delay between each thread creation, so each thread will get enough time to complete it and increase the count variable by one at the end in a perfect order.

Since line 37 is commented, no change happens with the change on the 37th line ???

c) Now, uncomment all sleep() statements.

i. Can you explain the results?

Result:

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 2:3 says hi!

Thread 2:2 says hi!
Thread 2:1 says hi!
Thread 2:3 says hi!

Thread 3:2 says hi!
Thread 3:1 says hi!
Thread 4:2 says hi!
Thread 4:3 says hi!

Thread 4:3 says hi!
Main thread says hi!

Now there is a gap between every newly created thread of 1 second, and each thread will take 3 seconds for execution (indicated by the empty line in the above result) So the count value increased after 3 seconds and we can see the result of that increment in the 7th line of the result so on.

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

Result:

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 1:3 says hi!
Thread 2:2 says hi!
Thread 2:1 says hi!
Thread 3:3 says hi!
Thread 3:1 says hi!
Thread 3:2 says hi!
Thread 4:3 says hi!
Thread 4:2 says hi!
Main thread says hi!

Yes. Now each thread will take 3 seconds to complete the execution, and the gap between each thread is 1 second. At the end of the loop, the main thread is waiting for only 1 second, and the last thread(s) will not be able to complete its execution because of the end of the main thread.

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

No, can't universally agree on the above statement. In some programs, we can estimate the time taken by each thread and can define enough time of *sleep()* in the main thread. But it is not possible to estimate the thread execution time on each and every program since the effects of CPU process scheduling, the run time complexity of the programs (in threads), and many other reasons. So it is better to use join calls with signals rather than *sleep()* functions in practice.

Exercise 4

1. Use the following skeleton code to implement a multi-threaded server.

[check the *ex4.c* file for answers]

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?

Usually, the local variables in a program are stored in the stack. Here, if we created a variable (or pointer) in the usual way, it will be created in the stack of the main thread. But we had to pass this pointer value into the thread function and the stack implementation does not guarantee the availability of the pointer location until the end of the thread execution. So it is recommended to use a memory space (pointer) in the heap section (by using malloc function) as the argument of the pThread function.