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

Note: sleep(1) statement is explained as a ‘1 second’ delay in the below explanations.

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 increases 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!

Since we comment out the 'pthread_join' section, the main thread will not wait until all the threads 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 completes.

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 uncomment the line 12:

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

Each thread will take 3 seconds of time to complete the execution and increase the *'count'* variable in the main thread. But the main thread will create new threads before the previous thread increases the 'count' value; concurrently. So all the created threads print the initial value of the 'count' variable; 1 until the main thread ends and terminate the process.

The result after uncomment the line 35:

Thread 1:1 says hi! (thread: 1)
Thread 1:2 says hi! (thread: 1)
Thread 1:3 says hi! (thread: 1)
Thread 2:1 says hi! (thread: 2)
Thread 2:2 says hi! (thread: 2)
Thread 2:3 says hi! (thread: 2)
Thread 3:1 says hi! (thread: 3)
Thread 3:2 says hi! (thread: 3)
Thread 3:3 says hi! (thread: 3)
Thread 4:1 says hi! (thread: 4)
Thread 4:2 says hi! (thread: 4)
Thread 4:3 says hi! (thread: 4)
Thread 5:1 says hi! (thread: 5)
Thread 5:2 says hi! (thread: 5)
Thread 5:3 says hi! (thread: 5)

Main thread says hi!

Now, the main loop has a 1-second gap between each thread, so this time is enough to complete the execution of each thread, print, and increase the ‘count’ variable.

The result after uncomment the line 37:

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!

Here, no gap between the creation of each thread, but the main thread will wait for 5 seconds and that time will be enough to complete the execution of every thread created.

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

i. lines 35 & 37

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!

Each thread will get enough time to complete the execution, so the output is equal to output given by the original code with which had *pthread_join* function.

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

Result:

Thread 1:1 says hi! [thread number 1]
Thread 1:2 says hi! [thread number 1]
Thread 1:1 says hi! [thread number 2]
Thread 1:3 says hi! [thread number 2]
Thread 1:1 says hi! [thread number 3]
Thread 1:2 says hi! [thread number 3]
Thread 1:2 says hi! [thread number 3]
Thread 2:1 says hi! [thread number 4]
Thread 2:3 says hi! [thread number 4]
Thread 2:3 says hi! [thread number 4]
Thread 3:2 says hi! [thread number 4]
Thread 3:1 says hi! [thread number 5]
Thread 4:3 says hi! [thread number 5]
Thread 4:2 says hi! [thread number 5]
Main thread says hi!

Here, there is a delay between each thread of 1 second. But each thread will take around 3 seconds to finish its execution and increase the '*count*' variable value by once. So the increment is done by the first thread is appear on the 4th thread and so on.

But the delay added in *sleep(1)* can be varied and the numbers printed in the STDOUT can be changed in different executions.

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

Result:

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

Here, no any time delay between all 5 threads, so the main loop will create 5 threads one after another, and none of the threads will be able to increase the '*count*' variable by one, before the end of the *for-loop*. Anyway, the *sleep()* in line 37 allows to complete the execution of all threads.

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 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 schedulings, 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.