CO327 LAB 03 :INTRODUCTION TO MULTITHREADING

E/13/058
De Silva M.D.R.A.M
Semester 06

Using POXIS pthread library,

Exercise 1

1.Save example1.c as thread.c and compile it as, gcc -pthread -o thread thread.c -Wall

```
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ gcc -pthread -o thread thread.c -Wall
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ ./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!
```

2.Why do you need '-pthread' flag when you compile?

Without -pthread ,

```
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ gcc -o threadnop thread.c -Wall
/tmp/ccbniGi6.o: In function `main':
thread.c:(.text+0x68): undefined reference to `pthread_create'
thread.c:(.text+0x91): undefined reference to `pthread_join'
collect2: error: ld returned 1 exit status
```

We need '-pthread' flag in order to tell the compiler to link in the pthread library as well as configure the compilation for threads.

Exercise 02

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

```c
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void){

    int i;

    for(i = 0;i<3;i++){
        fork();
        printf("pid = %d,ppid = %d,i = %d\n",getpid(),getppid()

    wait(NULL);



    }



}
```

```
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ gcc -Wall forloop_pcount.c -o forlooppcount
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ ./forlooppcount
pid = 3527,ppid = 3340,i = 0
pid = 3528,ppid = 3527,i = 0
pid = 3528,ppid = 3527,i = 1
pid = 3529,ppid = 3528,i = 1
pid = 3529,ppid = 3528,i = 2
pid = 3530,ppid = 3529,i = 2
pid = 3528,ppid = 3527,i = 2
pid = 3531,ppid = 3528,i = 2
pid = 3527,ppid = 3340,i = 1
pid = 3532,ppid = 3527,i = 1
pid = 3532,ppid = 3527,i = 2
pid = 3533,ppid = 3532,i = 2
pid = 3527,ppid = 3340,i = 2
pid = 3534,ppid = 3527,i = 2
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$
```

a.) How many new processes did it create?

Number of new processes created = 8.

b.) Consider this piece of code,

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <pthread.h>
#include <sys/wait.h>

void *function(void *arg)
{

printf("Thread says hi!\n");
sleep(2);

return NULL;
}

int main(void){
    pthread_t myThread;
    int i;

    for(i = 0;i<3;i++){

        pthread_create(&myThread,NULL,function,NULL);
        sleep(1);

    }

}
```

output:

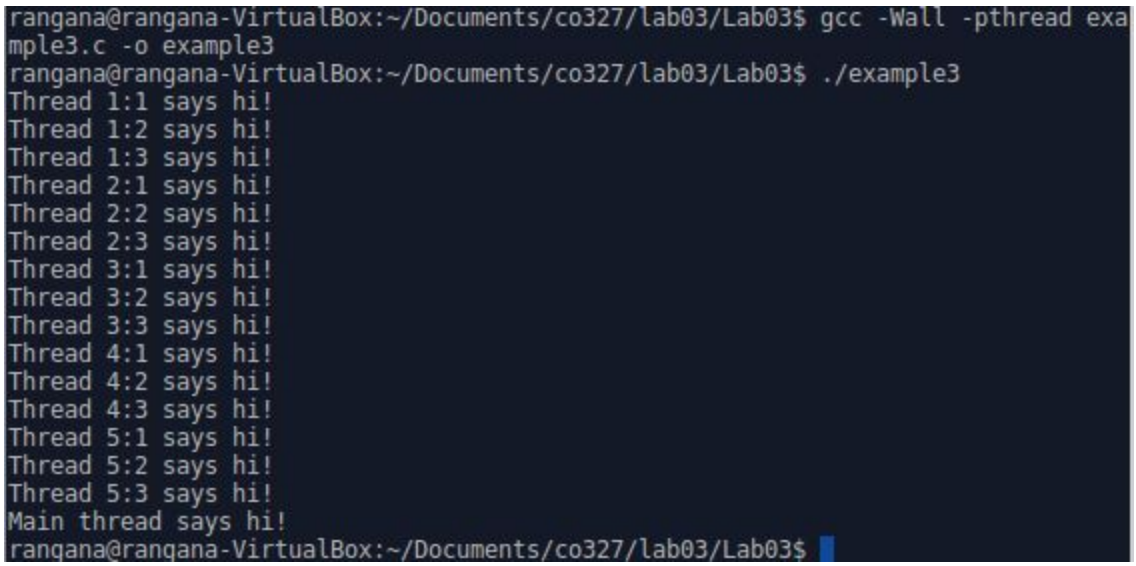
```
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ gcc -Wall -pthread exe
rcise01_b.c -o exercise01_b
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ ./exercise01_b
Thread says hi!
Thread says hi!
Thread says hi!
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$
```

It creates three threads.

In comparison with part a we see that in part a by using fork(), 8 processes were created for a 3 time loop. Here, only three threads are created. The reason is that fork() command creates a new child process every time it is executed and hence create many processes but in part b no new duplicated processes are created and only three threads for three loop calls.

Exercise 03

1.



a.) Explain the result.

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

void *thread_function(void *arg)
{
        int a;
        for(a = 1; a <= 3; a++)
        {
                printf("Thread %d:%d says hi!\n", *(int *)arg, a);
//              sleep(1);
        }

        (*(int *)arg)++;
        return NULL;
}

int main(void)
{
        pthread_t mythread;
        int i, count = 1;

        for (i = 0; i < 5; i++){
                if ( pthread_create( &mythread, NULL, thread_function, &count)
                {
                        printf("error creating thread.");
                        abort();
                }
                if ( pthread_join ( mythread, NULL ) )
                {
                        printf("error joining thread.");
                        abort();
                }
//              sleep(1);
        }
//      sleep(5);
        printf("Main thread says hi!\n");
        exit(0);
}
```

When you compile this code with -pthread and run it first the main method gets invoked and in that there is 5 round loop. In that loop pthread_create function is called which starts a new thread.it starts execution by invoking “thread_function” . In that function a 3 time loop is called where and printf command is executed. At completion of pthread_create function it returns o, and hence the if(pthread_create()) condition in the main loop is not true when it returns zero, main loop will proceed with the second loop call and so on till 5 loops calls are finished. In every loop call thread function is invoked and there by 5 threads are created. And in each thread “thread_function” is invoked to print three stdout lines.

b.) What is the objective of the code in line 15? Deconstruct this statement, and explain how the objective you mentioned is achieved

‘(*(int *)arg)++;’ →
It increments the counter variable . thread_function gets the input argument count when called in the main method and this address variable is casted into an int and dereferenced to get the value of count and incremented. The objective is to increment the count value.

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

When proper joining is not done, the main thread will no longer wait for the end of newly created threads.Hence, it will print the main thread stdout line only as shown below.

```
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ gcc -Wall -pthread exa
mple3.c -o withoutjoin
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ ./withoutjoin
Main thread says hi!
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$
```

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

Comment out line 12

```
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ gcc -Wall -pthread exa
mple3.c -o commentline12
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ ./commentline12
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!
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$
```

At this moment the joint statement is commented out as well as line 12 sleep() statements.
We see the same result we get with the joint statement appearing.
This happens as the other two sleep () statements in the main method gives some time to the thread to execute its print statement. sleep() in line 37 gives some time for the thread to execute itself.After that only the next loop call occur and after each and every loop call only the main thread print statement is printed.

Comment out line 35

```
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ gcc -Wall -pthread exa
mple3.c -o commentline35
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ ./commentline35
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 1:3 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!
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$
```

At this moment the joint statement is commented out as well as line 35 sleep() statements. We see that the counter value is not updated and as before it has created five threads which has executed thread_function and finally has printed the main thread statement.The counter value has not updated as we've not given some time for a thread_function to update its count value.

Comment out line 37

```
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ gcc -Wall -pthread exa
mple3.c  -o commentline37
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ ./commentline37
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 1:1 says hi!
Thread 2:3 says hi!
Thread 2:2 says hi!
Thread 2:3 says hi!
Thread 3:2 says hi!
Thread 3:1 says hi!
Main thread says hi!
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$
```

At this moment the joint statement is commented out as well as line 37 sleep() statements.

By commenting out this line we see that different outputs tend to show in the terminal when the same program is called again and again. The order shown above changes when executed again.

Main thread calls pthread_create function and creates a thread and will wait for 1 second before going to the next loop call. And within this time period pthread_function executes and will execute the print statement once in the pthread_create loop.Then main thread creates the second thread.By then pthread_create executes the second print line of the loop,"**Thread 1:2 says hi**!" .Then the second thread created starts executing and in that thread, the loop is called and it will execute the print statement for the 1st loop call of the pthread_create function. Thus we see, "**Thread 1:1 says hi**!' again in the terminal as the third line. Then the main thread creates the third thread and at this time the third loop call of the first thread created happens executing "**Thread 1:3 says hi!".** And by this time third thread is in the first loop call and the second thread is in the 1st lop call.Then since, the first thread is finished executing it wil update the counter variable which is shared by all threads and within next couple of seconds will be created going with the same procedure.

After five seconds where all the threads have being created , the statement "**Main thread says hi"** gets printed on the terminal.

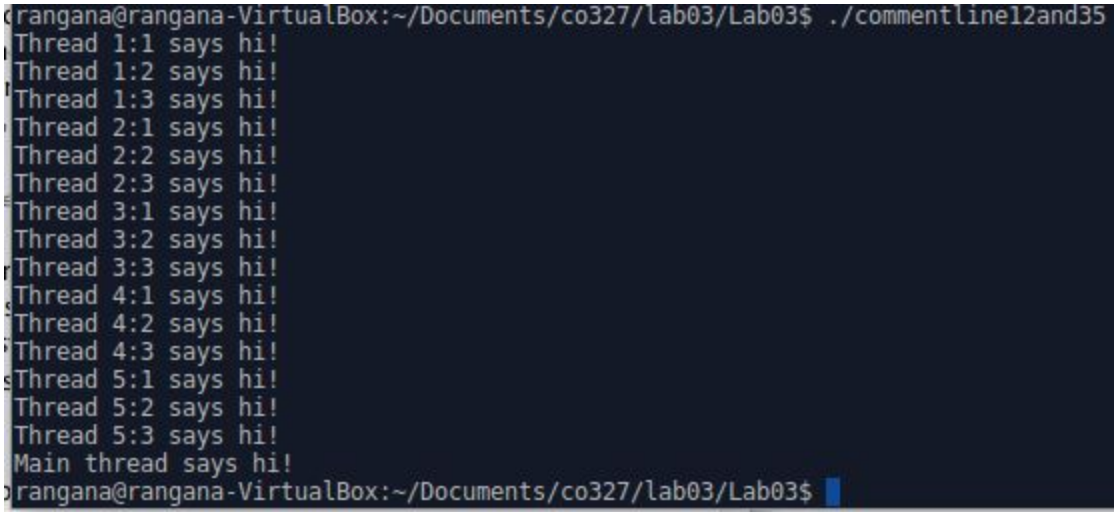
b) Now, comment out pairs of sleep() statements as follows, and explain the results.

i. **lines 35 & 37**

```
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ gcc -Wall -pthread ex
mple3.c  -o commentline35and37
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ ./commentline35and37
Main thread says hi!
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$
```

Main thread exists after creating the five threads and it will not wait for the results of the created threads.

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



Only sleep() statement in the code occurs after creating all the five threads. Ad at each thread the count variable is also updated without a sleep statement and we see that the count variable has been updated correctly in the output as well. Since, there's no waiting happening when threads are executed when run several times there can be a jumble up in the print statements , even though here we see them in order.

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

c) Now, uncomment all sleep() statements.

```
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ gcc -Wall -pthread exa
mple3.c  -o uncommentall
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$ ./uncommentall
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!
Thread 4:3 says hi!
Thread 4:2 says hi!
Thread 5:3 says hi!
Main thread says hi!
rangana@rangana-VirtualBox:~/Documents/co327/lab03/Lab03$
```

We see that the count variable has updated up to 5 and this is because of the wait we provide by sleep(5). If sleep(1) was used instead of sleep(5) then it will not return the same output and will stop at the update point where count is 4. This is because one second is not enough.

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

Pthread_join() waits until the thread finish and sleep() will wait for a given time period. Pthread_join() will ensure that it waits for the exact time which the threads take to execute and that guarantee is not there with sleep()

Exercise 4

1.Implement a multi-threaded server.

In the submission folder named **multi_threaded_server.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?

If it was declared as a as a local variable then, each client will have to update the connection again and again as it uses the same memory space. Instead of that by using as a variable in the heap it will create different memory locations for each client.
.