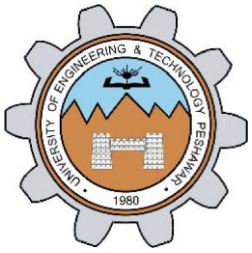
LAB #09

Threads: Passing Arguments to Threads



Spring 2023

CSE-204L Operating Systems Lab

Submitted by: MUHAMMAD SADEEQ

Registration No.: 21PWCSE2028

Section: C

"On my honor, as a student of the University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work"

Submitted to:

Engr. Madiha Sher

(27 May 2023)

Department of Computer systems engineering
University of Engineering and Technology,
Peshawar

TASK 1

CODE:

```c
/* Box #1 : Passing Thread Arguments */
#include <pthread.h>
#include <stdio.h>

void *ChildThread (void*  argument)
{
    int   i ;
    for(i=1;i<= *((int*)argument);i++){
    printf("Count :%d\n",i);
    }
      pthread_exit(NULL);
}

int main(void)
{
   pthread_t   hThread;
   int count=5;
   pthread_create (&hThread,NULL,ChildThread,(void*)&count);

   pthread_join (hThread, NULL);

       printf ("Parent is continuing....\n");
       return 0;
}
```

Output:

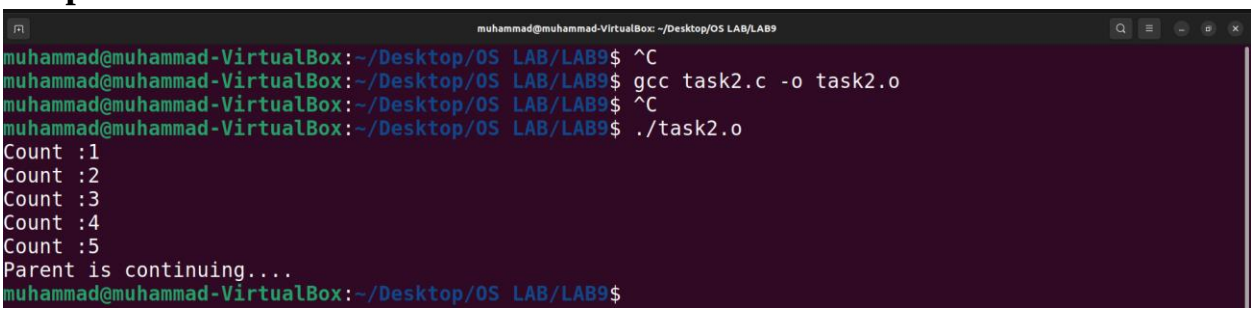
```
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB9$ ^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB9$ ./task1.o
Count :1
Count :2
Count :3
Count :4
Count :5
Parent is continuing....
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB9$
```

TASK 2

CODE:

```c
#include <pthread.h>
#include <stdio.h>

void *ChildThread (void*  argument){
    int   i ;
    for(i=1;i<= *((int*)argument);i++){
    printf("Count :%d\n",i);
    }

}

int main(void)
{
   pthread_t   hThread;
   int count=5;
   pthread_create (&hThread,NULL,ChildThread,(void*)&count);

   pthread_join (hThread, NULL);

       printf ("Parent is continuing....\n");
       return 0;
}
```

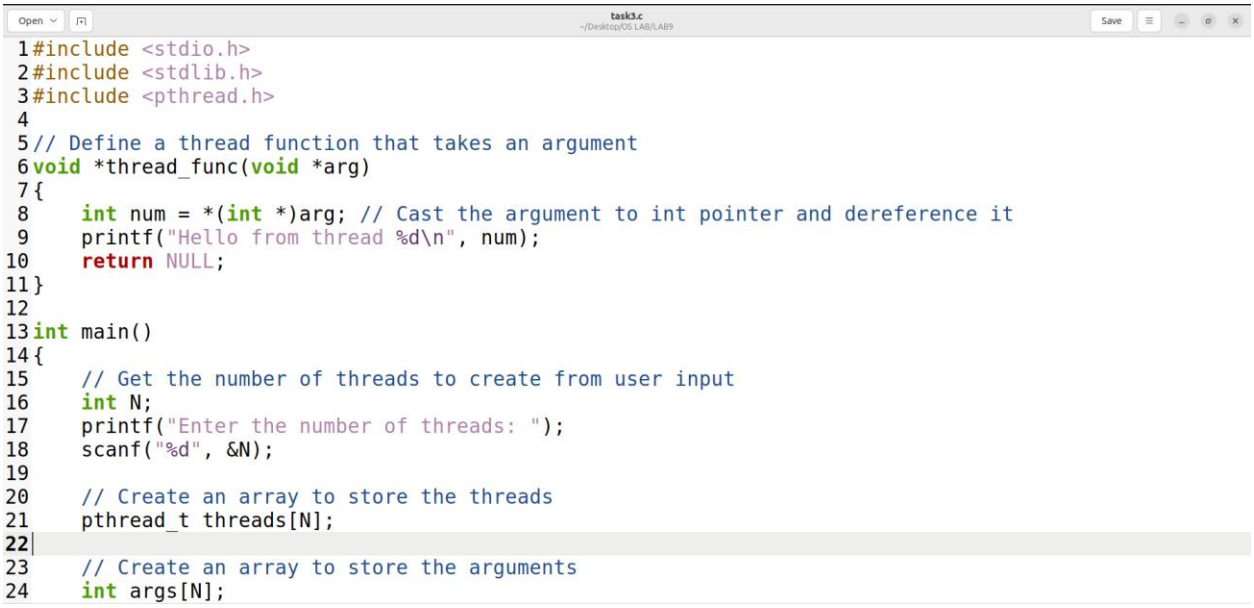
Output:



The output is the same as Box #1 because performing a return from the start function of any thread other than the main thread results in an implicit call to pthread_exit(), using the function's return value as the thread's exit status1. Therefore, calling pthread_exit explicitly or returning implicitly has the same effect on terminating the thread and making its exit status available to other threads.

TASK 3

CODE:




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

// Define a thread function that takes an argument
void *thread_func(void *arg)
{
    int num = *(int *)arg; // Cast the argument to int pointer and dereference it
    printf("Hello from thread %d\n", num);
    return NULL;
}

int main()
{
    // Get the number of threads to create from user input
    int N;
    printf("Enter the number of threads: ");
    scanf("%d", &N);

    // Create an array to store the threads
    pthread_t threads[N];

    // Create an array to store the arguments
    int args[N];
```

```
25
26      // Create N threads in a for loop
27      for (int i = 0; i < N; i++)
28      {
29          // Assign the loop variable to the argument array
30          args[i] = i;
31          // Create a new thread with the thread function and the argument array element
32          if (pthread_create(&threads[i], NULL, thread_func, &args[i]) != 0)
33          {
34              fprintf(stderr, "Failed to create thread %d\n", i);
35              exit(1);
36          }
37      }
38
39      // Wait for all threads to finish
40      for (int i = 0; i < N; i++)
41      {
42          if (pthread_join(threads[i], NULL) != 0)
43          {
44              fprintf(stderr, "Failed to join thread %d\n", i);
45              exit(1);
46          }
47      }
48
49      // Print a message when done
50      printf("All threads are done\n");
51      return 0;
52  }
```

Output:

```
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB9$ ^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB9$ ./task3.o
Enter the number of threads: 5
Hello from thread 3
Hello from thread 2
Hello from thread 1
Hello from thread 0
Hello from thread 4
All threads are done
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB9$ ^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB9$ ./task3.o
Enter the number of threads: 5
Hello from thread 4
Hello from thread 0
Hello from thread 3
Hello from thread 1
Hello from thread 2
All threads are done
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB9$
```

The reason for this variation is that the order of execution of the threads is not guaranteed by the pthread library. The threads may run concurrently or interleaved, depending on how the operating system allocates CPU time to them. Therefore, we cannot predict or control the exact output of this program.

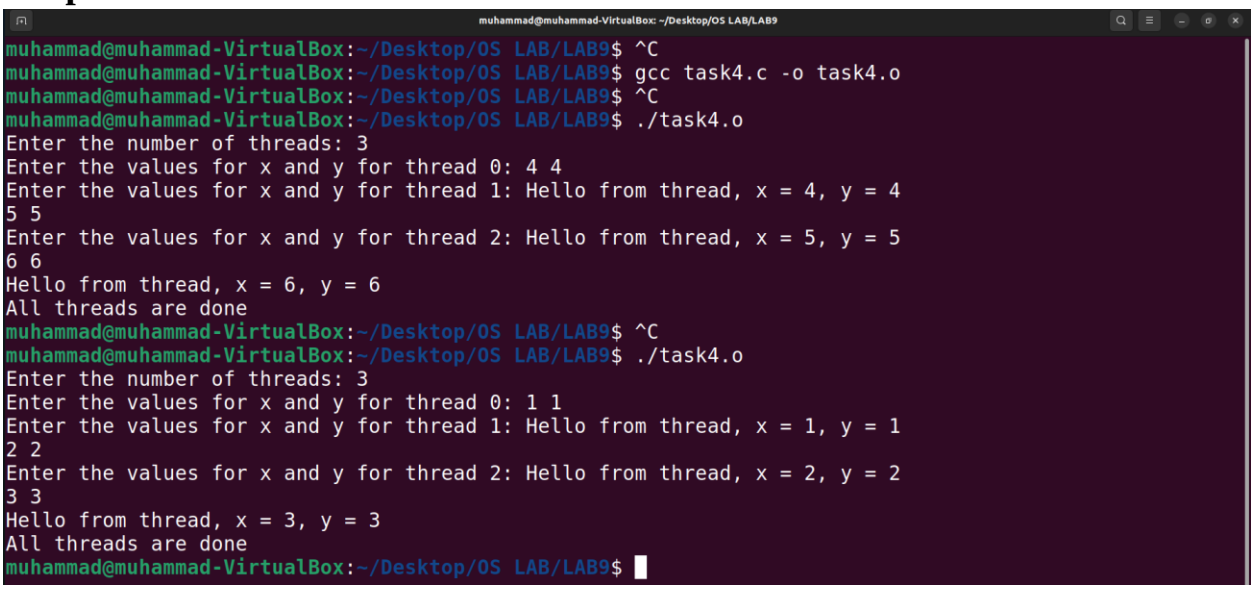
TASK 4

CODE:

task4.c
~/Desktop/OS LAB/LAB9

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

// Define a structure with two data members
struct Data {
    int x;
    int y;
};

// Define a thread function that takes a structure pointer as an argument
void *thread_func(void *arg)
{
    // Cast the argument to a structure pointer and dereference it
    struct Data data = *(struct Data *)arg;
    // Display the data members
    printf("Hello from thread, x = %d, y = %d\n", data.x, data.y);
    // Free the allocated memory for the structure
    free(arg);
    return NULL;
}

int main()
{
    // Get the number of threads to create from user input
    int N;
    printf("Enter the number of threads: ");
    scanf("%d", &N);

    // Create an array to store the threads
    pthread_t threads[N];

    // Create N threads in a for loop
    for (int i = 0; i < N; i++)
    {
        // Allocate memory for a new structure
        struct Data *data = malloc(sizeof(struct Data));
        // Take the values for the data members as input
        printf("Enter the values for x and y for thread %d: ", i);
        scanf("%d%d", &data->x, &data->y);
        // Create a new thread with the thread function and the structure pointer as argument
        if (pthread_create(&threads[i], NULL, thread_func, data) != 0)
        {
            fprintf(stderr, "Failed to create thread %d\n", i);
            exit(1);
        }
    }

    // Wait for all threads to finish
    for (int i = 0; i < N; i++)
    {
        if (pthread_join(threads[i], NULL) != 0)
        {
            fprintf(stderr, "Failed to join thread %d\n", i);
            exit(1);
        }
    }

    // Print a message when done
    printf("All threads are done\n");
    return 0;
}
```

Output:



The reason for this variation is that the order of execution of the threads is not guaranteed by the pthread library. The threads may run concurrently or interleaved, depending on how the operating system allocates CPU time to them. Therefore, we cannot predict or control the exact output of this program.