Threads: Passing Arguments to Threads

LAB # 09

Spring 2023

CSE-204L Operating Systems Lab

Submitted by: **Ali Asghar**

Registration No.: **21PWCSE2059**

Class Section: **C**

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

Submitted to:

Engr. Madiha Sher

Date:

26th June 2023

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Objectives:

This lab examines aspects of **threads** and **multiprocessing** (and **multithreading**) and how the arguments may be passed to the child threads.
****

Passing Arguments to Threads:

The **pthread_create( )** routine permits the programmer to pass one argument to the thread start routine. For cases where multiple arguments must be passed, this limitation is easily overcome by creating a structure which contains all of the arguments, and then passing a pointer to that structure in the **pthread_create( )** routine.

All arguments must be passed by reference and cast to **(void *)**. Important: threads initially access their data structures in the parent thread's memory space. That data structure must not be corrupted/modified until the thread has finished accessing it.

The following example pass a simple integer to each thread.

Example: pthread_create( ) argument passing :

Lab6_5.c

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 7

char *messages[NUM_THREADS];

void *PrintHello(void *threadid)
{
   int *id_ptr, taskid;
   sleep(1);
   id_ptr = (int *) threadid;
   taskid = *id_ptr;
   printf("\n %s from thread %d \n\n", messages[taskid], taskid);
   pthread_exit(NULL);
}

int main( )
{
   pthread_t   threads[NUM_THREADS];
   int *taskids[NUM_THREADS];
   int rc, t;
   messages[0] = "English: Hello World!";
   messages[1] = "French: Bonjour, le monde!";
   messages[2] = "Spanish: Hola al mundo";
   messages[3] = "Klingon: Nuq neH!";
   messages[4] = "German: Guten Tag, Welt!";
   messages[5] = "Russian: Zdravstvytye, mir!";
   messages[6] = "Japan: Sekai e konnichiwa!";
   messages[7] = "Latin: Orbis, te saluto!";

   for(t=0;t<NUM_THREADS;t++)
   {
      taskids[t] = (int *) malloc(sizeof(int));
      *taskids[t] = t;
      printf("Creating thread %d\n", t);
      rc = pthread_create(&threads[t], NULL, PrintHello, (void *) taskids[t]);
      if (rc)
      {
         printf("ERROR; return code from pthread_create() is %d\n", rc);
         exit(-1);
      }
   }

pthread_exit(NULL);
}
```

Sample output

```
Creating thread 0
Creating thread 1
Creating thread 2
Creating thread 3
Creating thread 4
Creating thread 5
Creating thread 6

English: Hello World! from thread 0
French: Bonjour, le monde! from thread 1
Spanish: Hola al mundo from thread 2
Klingon: Nuq neH! from thread 3
German: Guten Tag, Welt! from thread 4
Japan: Sekai e konnichiwa! from thread 6
Russian: Zdravstvytye, mir! from thread 5
```

Assignments:

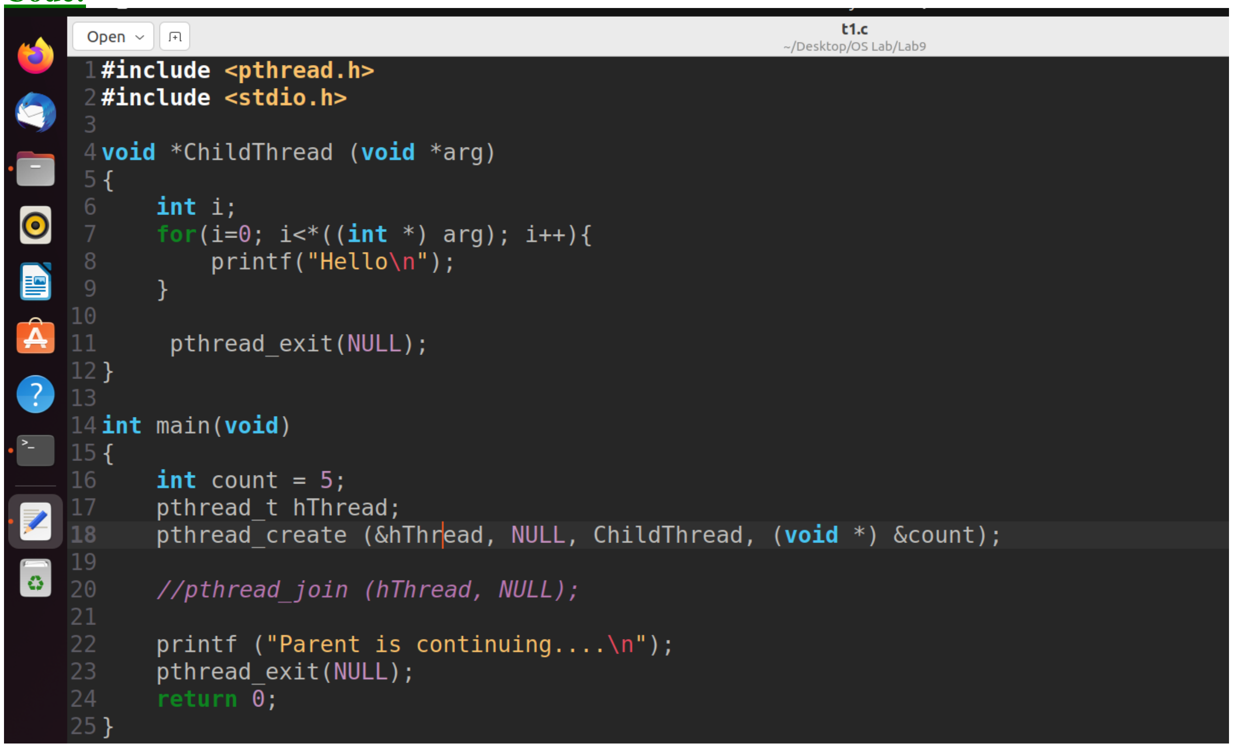
Problem # 1:

Modify the above **Box #1** program such that the main program passes the **count** as argument to the child thread function and the child thread function prints that many **count** print statements.

```
/* Box #1 : Passing Thread Arguments */

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

void *ChildThread (int  argument)
{
   int   i;

     .......................

     pthread_exit(NULL);
}

int main(void)
{
  pthread_t   hThread;
  pthread_create (............................);

  pthread_join (hThread, NULL);

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

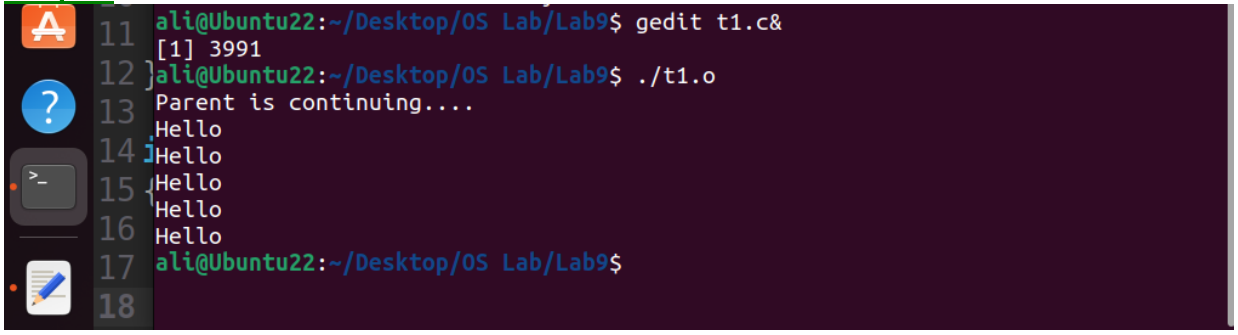
Compile and Execute the Box #2 program and show the output and explain why is the output so?

Code:



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

void *ChildThread (void *arg)
{
    int i;
    for(i=0; i<*((int *) arg); i++){
        printf("Hello\n");
    }

     pthread_exit(NULL);
}

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

    //pthread_join (hThread, NULL);

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

Output:



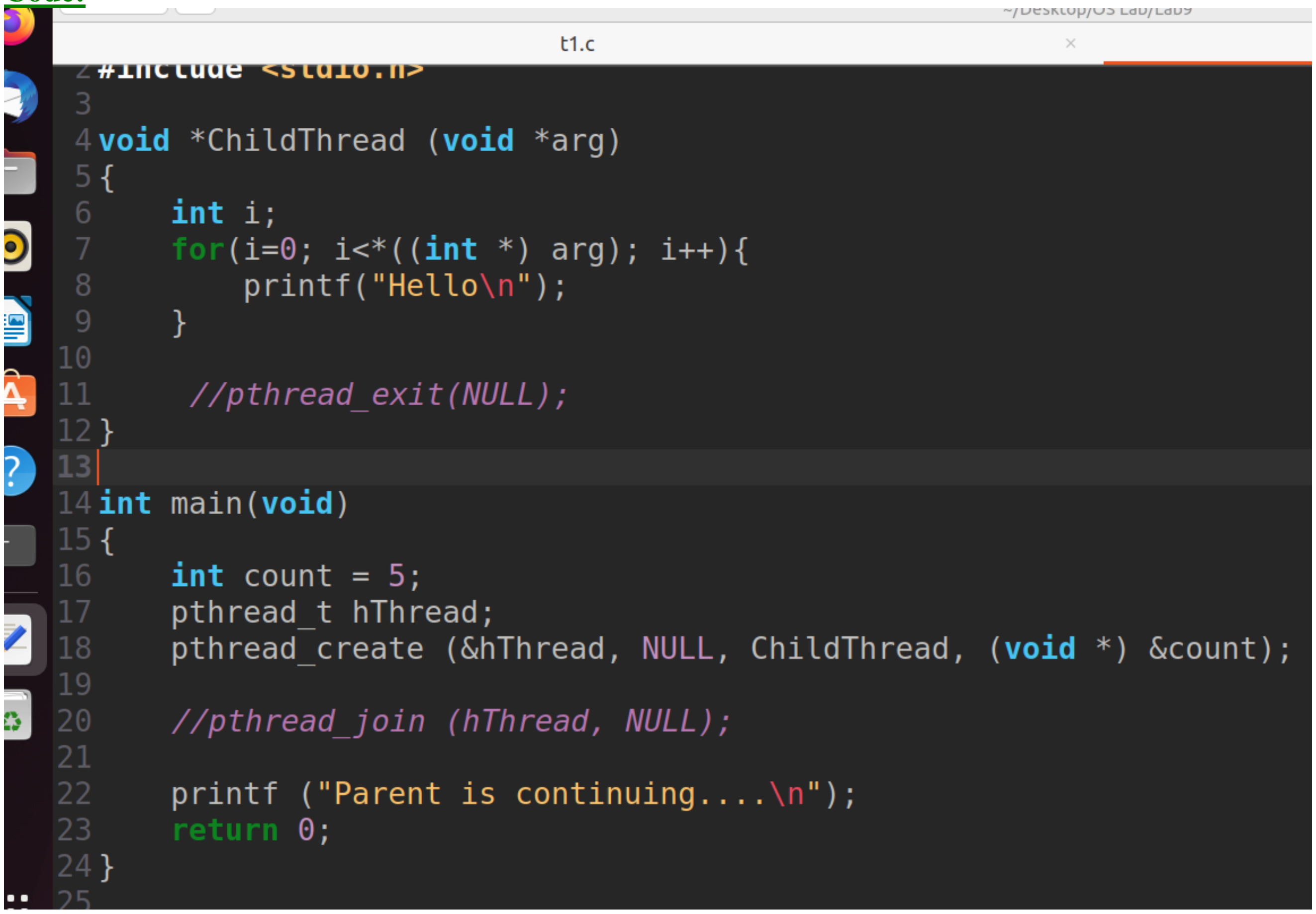
```
ali@Ubuntu22:~/Desktop/OS Lab/Lab9$ gedit t1.c&
[1] 3991
ali@Ubuntu22:~/Desktop/OS Lab/Lab9$ ./t1.o
Parent is continuing....
Hello
Hello
Hello
Hello
Hello
ali@Ubuntu22:~/Desktop/OS Lab/Lab9$
```

Problem # 2:

Write a program Box # 2 by removing pthread_exit function from child thread function and check the output? Is it the same as output of Box # 1? If so Why? Explain?

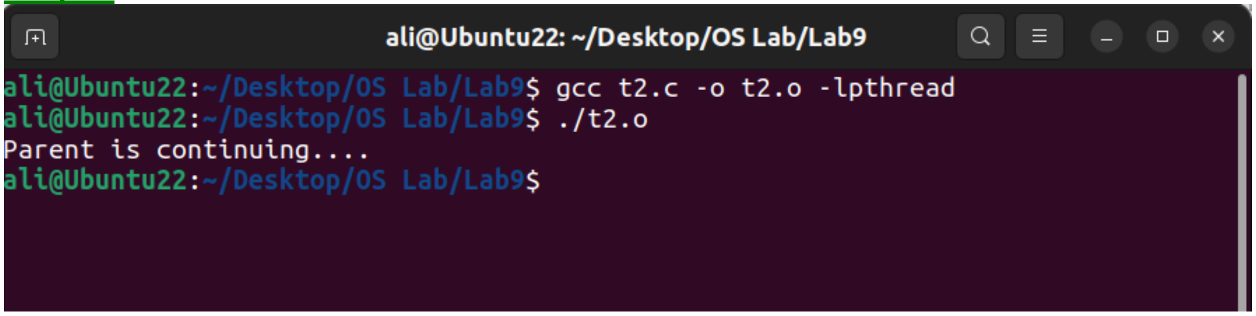
```
/* Box #3: Implicit Thread Exit  */

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

void ChildThread (int argument)
{
     int i;

     ...............................
     /* No pthread_exit function */
}

int main(void)
{
  pthread_t   hThread;

  pthread_create (..........................................);

  pthread_join (hThread, NULL);
  printf ("Parent is continuing....\n");        return 0;
}
```

Code:



```
t1.c
#include <stdio.h>

void *ChildThread (void *arg)
{
    int i;
    for(i=0; i<*((int *) arg); i++){
        printf("Hello\n");
    }

     //pthread_exit(NULL);
}

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

    //pthread_join (hThread, NULL);

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

Output:



```
ali@Ubuntu22: ~/Desktop/OS Lab/Lab9
ali@Ubuntu22:~/Desktop/OS Lab/Lab9$ gcc t2.c -o t2.o -lpthread
ali@Ubuntu22:~/Desktop/OS Lab/Lab9$ ./t2.o
Parent is continuing....
ali@Ubuntu22:~/Desktop/OS Lab/Lab9$
```

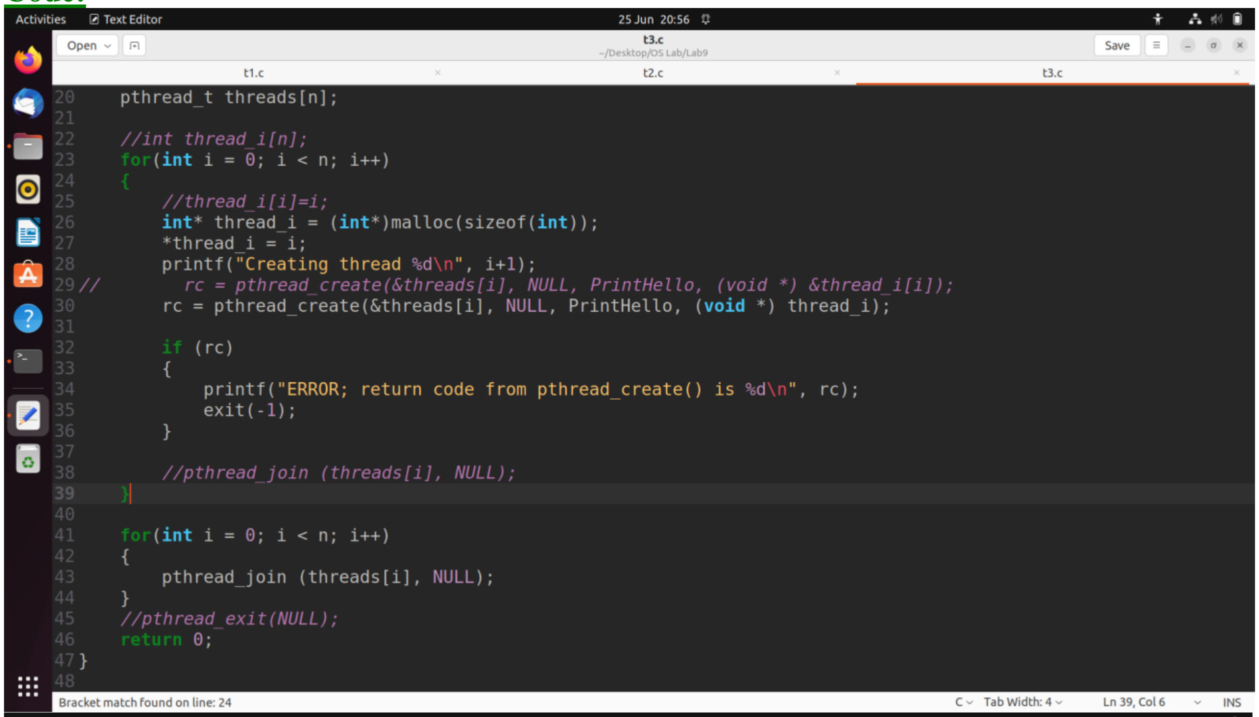
Output Explanation:

No, the output is not the same as Box # 1 because main thread do not wait for child thread in this case. However, if we put pthread_join statement in main function then the output will be same as Box # 1. This is because main thread will wait for child threads to finish.

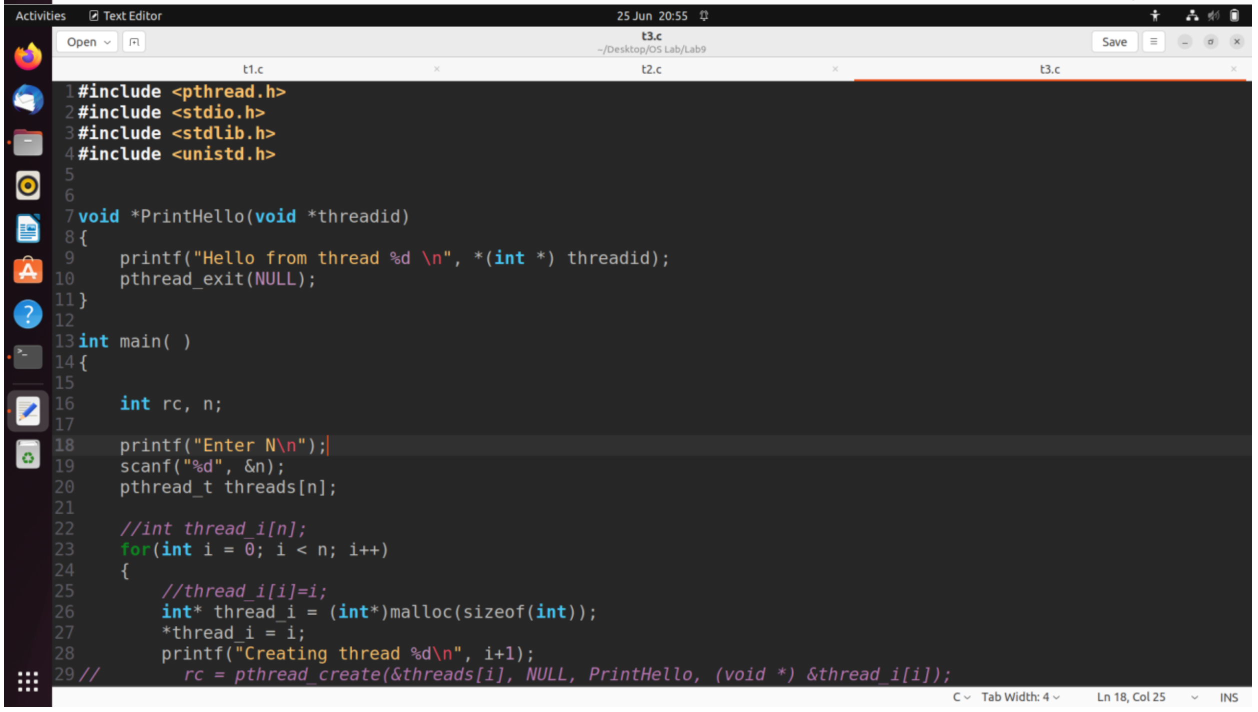
Problem # 3:

Write a program that creates N child threads in for loop. Pass the value of loop variable as argument to thread function. Analyze the output and give reasons for the output you observe.

Code:

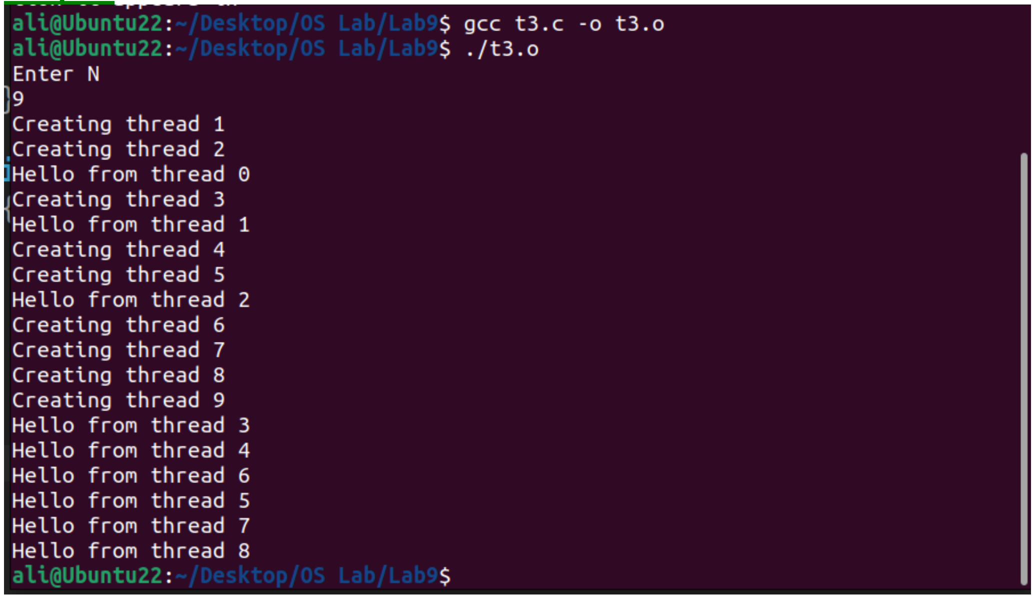


```
    pthread_t threads[n];

    //int thread_i[n];
    for(int i = 0; i < n; i++)
    {
        //thread_i[i]=i;
        int* thread_i = (int*)malloc(sizeof(int));
        *thread_i = i;
        printf("Creating thread %d\n", i+1);
//        rc = pthread_create(&threads[i], NULL, PrintHello, (void *) &thread_i[i]);
        rc = pthread_create(&threads[i], NULL, PrintHello, (void *) thread_i);

        if (rc)
        {
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }

        //pthread_join (threads[i], NULL);
    }

    for(int i = 0; i < n; i++)
    {
        pthread_join (threads[i], NULL);
    }
    //pthread_exit(NULL);
    return 0;
}
```



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


void *PrintHello(void *threadid)
{
    printf("Hello from thread %d \n", *(int *) threadid);
    pthread_exit(NULL);
}

int main( )
{

    int rc, n;

    printf("Enter N\n");
    scanf("%d", &n);
    pthread_t threads[n];

    //int thread_i[n];
    for(int i = 0; i < n; i++)
    {
        //thread_i[i]=i;
        int* thread_i = (int*)malloc(sizeof(int));
        *thread_i = i;
        printf("Creating thread %d\n", i+1);
//        rc = pthread_create(&threads[i], NULL, PrintHello, (void *) &thread_i[i]);
```

Output:



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab9$ gcc t3.c -o t3.o
ali@Ubuntu22:~/Desktop/OS Lab/Lab9$ ./t3.o
Enter N
9
Creating thread 1
Creating thread 2
Hello from thread 0
Creating thread 3
Hello from thread 1
Creating thread 4
Creating thread 5
Hello from thread 2
Creating thread 6
Creating thread 7
Creating thread 8
Creating thread 9
Hello from thread 3
Hello from thread 4
Hello from thread 6
Hello from thread 5
Hello from thread 7
Hello from thread 8
ali@Ubuntu22:~/Desktop/OS Lab/Lab9$
```

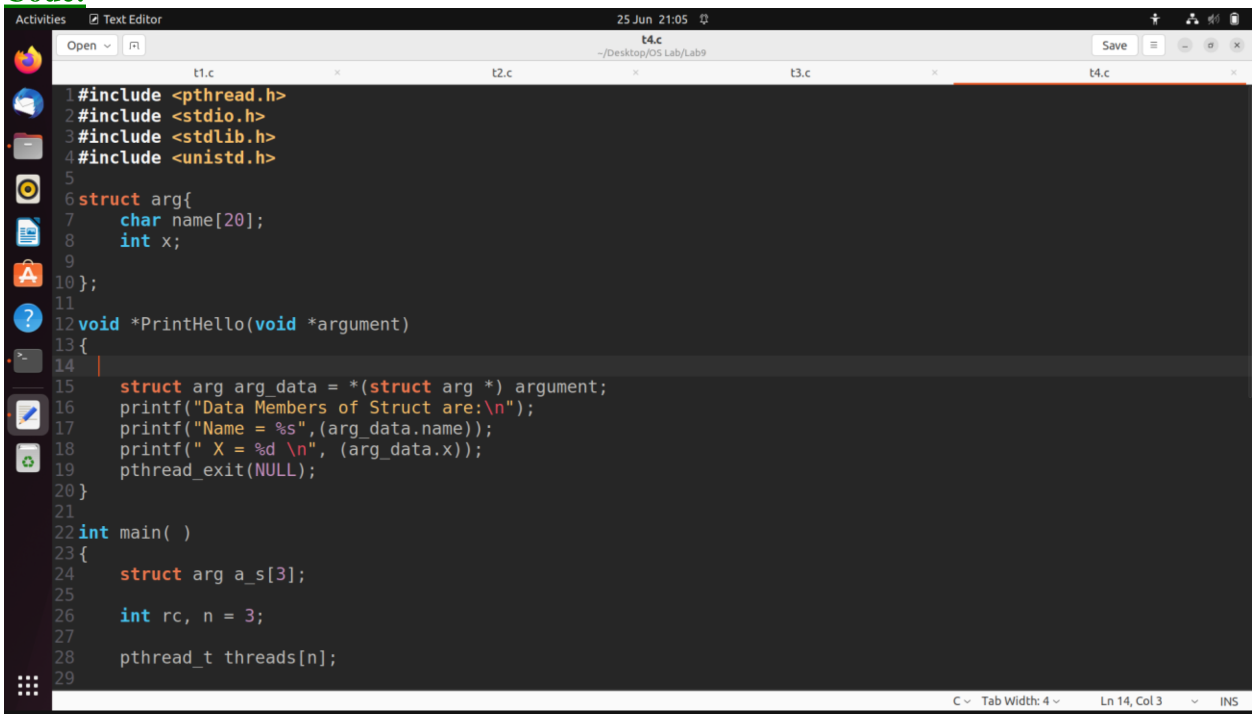
Output Explanation:

As expected, the output shows us that 9 threads have been created and each thread prints the sentence hello from [thread #]. In order to safely pass the thread number, we allocate new memory address in each iteration. If we pass the iterating variable directly, then the output will not be as expected. Output will be undesired because we'll be passing the variable by reference.

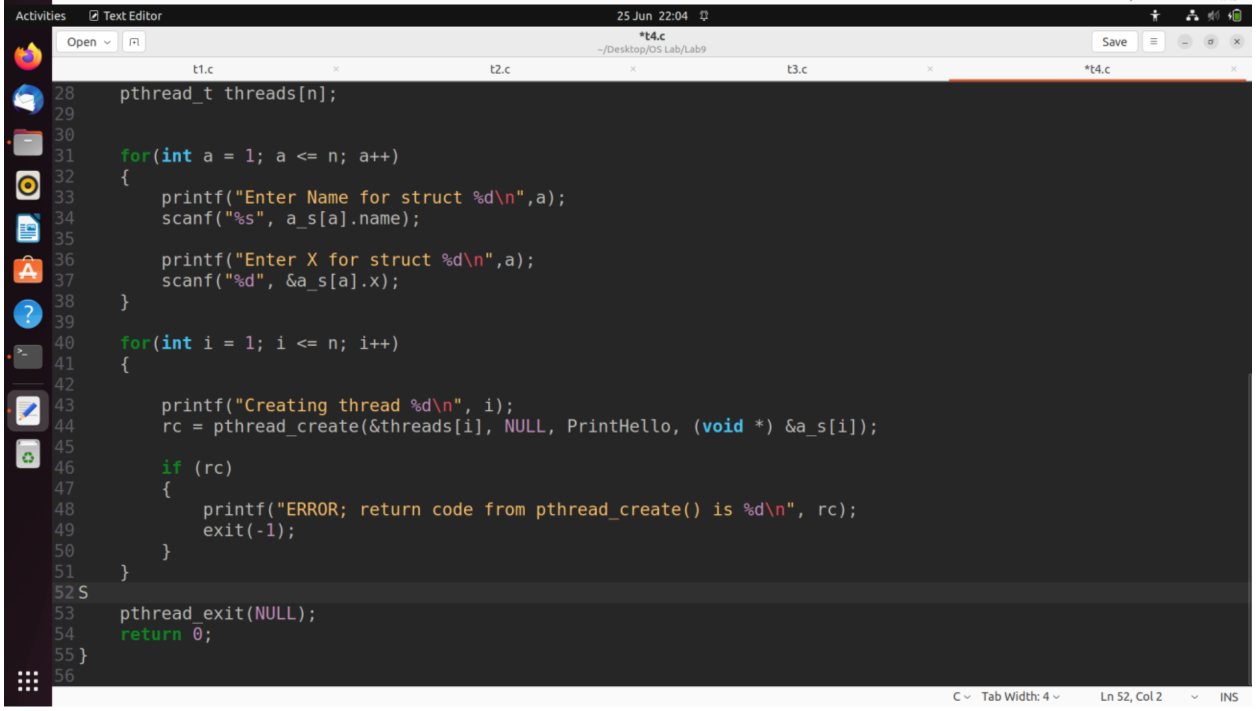
Problem # 4:

Write a program that passes a structure with at least two data members as an argument to the thread function. Take the values as input in main function and display the data members in thread function. Repeat for at least 3 child threads.

Code:

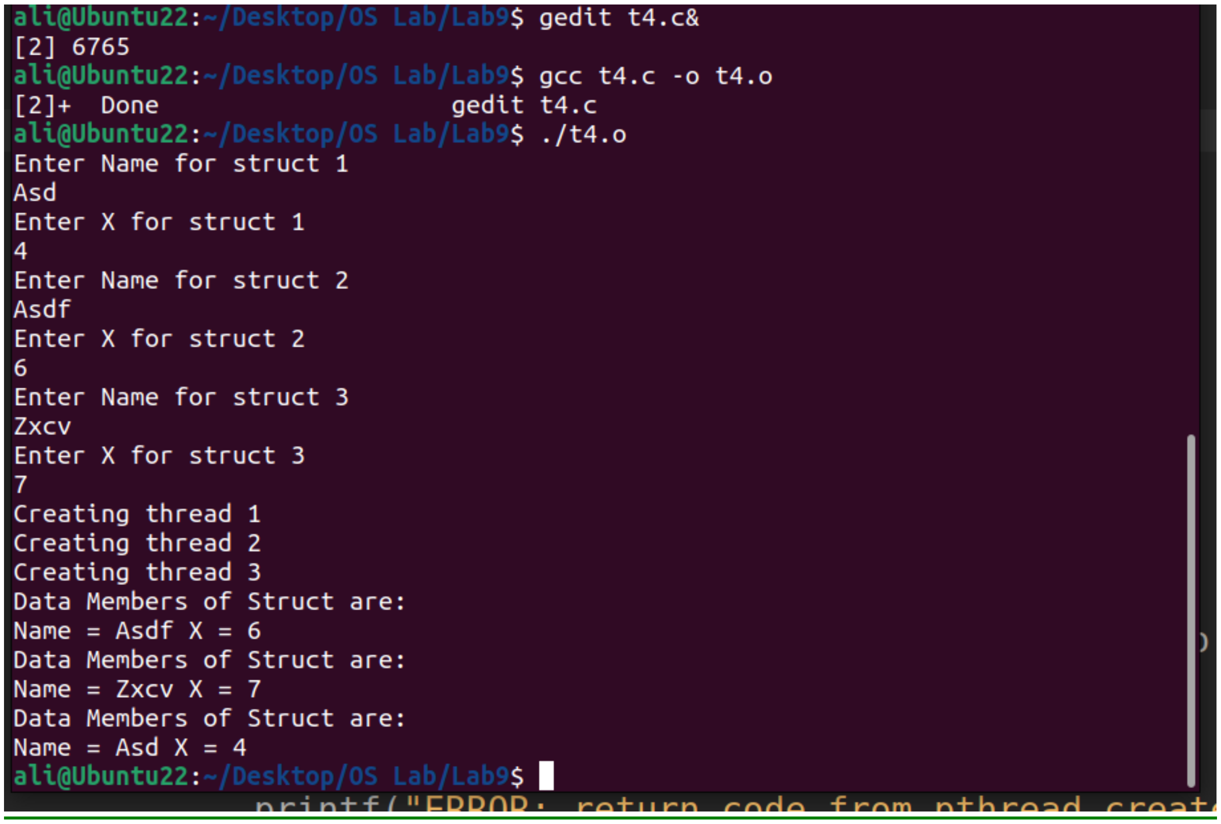


```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

struct arg{
    char name[20];
    int x;

};

void *PrintHello(void *argument)
{

    struct arg arg_data = *(struct arg *) argument;
    printf("Data Members of Struct are:\n");
    printf("Name = %s",(arg_data.name));
    printf(" X = %d \n", (arg_data.x));
    pthread_exit(NULL);
}

int main( )
{
    struct arg a_s[3];

    int rc, n = 3;

    pthread_t threads[n];

```



```c
    pthread_t threads[n];


    for(int a = 1; a <= n; a++)
    {
        printf("Enter Name for struct %d\n",a);
        scanf("%s", a_s[a].name);

        printf("Enter X for struct %d\n",a);
        scanf("%d", &a_s[a].x);
    }

    for(int i = 1; i <= n; i++)
    {

        printf("Creating thread %d\n", i);
        rc = pthread_create(&threads[i], NULL, PrintHello, (void *) &a_s[i]);

        if (rc)
        {
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
S
    pthread_exit(NULL);
    return 0;
}

```

Output:



```
ali@Ubuntu22:~/Desktop/OS Lab/Lab9$ gedit t4.c&
[2] 6765
ali@Ubuntu22:~/Desktop/OS Lab/Lab9$ gcc t4.c -o t4.o
[2]+  Done                    gedit t4.c
ali@Ubuntu22:~/Desktop/OS Lab/Lab9$ ./t4.o
Enter Name for struct 1
Asd
Enter X for struct 1
4
Enter Name for struct 2
Asdf
Enter X for struct 2
6
Enter Name for struct 3
Zxcv
Enter X for struct 3
7
Creating thread 1
Creating thread 2
Creating thread 3
Data Members of Struct are:
Name = Asdf X = 6
Data Members of Struct are:
Name = Zxcv X = 7
Data Members of Struct are:
Name = Asd X = 4
ali@Ubuntu22:~/Desktop/OS Lab/Lab9$ 
         printf("ERROR: return code from pthread_creat
```