

LAB #08

Threads Creation and Execution



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

(20 May 2023)

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

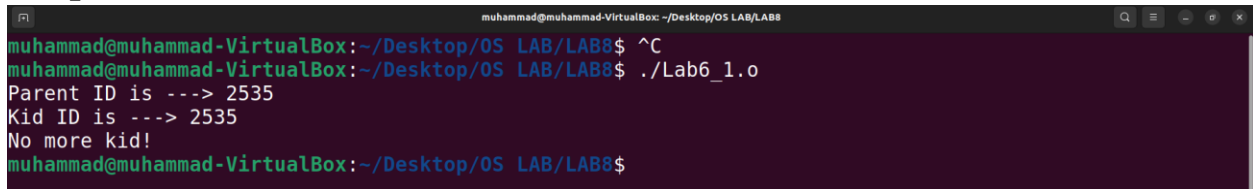
Example Lab6_1

CODE:



```
1#include <stdio.h>
2#include <pthread.h>
3#include <unistd.h>
4#include <stdlib.h>
5
6void *kidfunc(void *p)
7{
8    printf ("Kid ID is ---> %d\n", getpid( ));
9}
10
11void main ( )
12{
13    pthread_t kid ;
14    pthread_create (&kid, NULL, kidfunc, NULL);
15    printf ("Parent ID is ---> %d\n", getpid( ) ) ;
16    pthread_join (kid, NULL) ;
17    printf ("No more kid!\n") ;
18}
```

Output:



```
muhammad@muhammad-VirtualBox: ~/Desktop/OS LAB/LAB8$ ^C
muhammad@muhammad-VirtualBox: ~/Desktop/OS LAB/LAB8$ ./Lab6_1.o
Parent ID is ---> 2535
Kid ID is ---> 2535
No more kid!
muhammad@muhammad-VirtualBox: ~/Desktop/OS LAB/LAB8$
```

Question: Are the process id numbers of parent and child thread the same or different? Give reason(s) for your answer.

Answer: The process id numbers of parent and child thread are the same. This is because threads share the same address space and resources of the process that created them. They are different from processes, which have their own address space and resources and get different process id numbers when created by fork().

Example Lab6_2

CODE:

```

1#include <stdio.h>
2#include <pthread.h>
3#include <unistd.h>
4#include <stdlib.h>
5
6int glob_data = 5 ;
7
8void *kidfunc(void *p)
9{
10    printf ("Kid here. Global data was %d.\n", glob_data) ;
11    glob_data = 15 ;
12    printf ("Kid Again. Global data was now %d.\n", glob_data) ;
13}
14
15void main ( )
16{
17    pthread_t kid ;
18
19    pthread_create (&kid, NULL, kidfunc, NULL) ;
20    printf ("Master thread here. Global data = %d\n", glob_data) ;
21    glob_data = 10 ;
22    pthread_join (kid, NULL) ;
23    printf ("End of program. Global data = %d\n", glob_data) ;
24}

```

Output:

```

muhammad@muhammad-VirtualBox: ~/Desktop/OS LAB/LAB8
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB8$ ^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB8$ ./Lab6_2.o
Master thread here. Global data = 5
Kid here. Global data was 10.
Kid Again. Global data was now 15.
End of program. Global data = 15
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB8$

```

Question: Do the threads have separate copies of glob_data? Why?

Or why not?

Answer: No, the threads do not have separate copies of glob_data.

This is because threads share the same global variables and heap memory of the process that created them. They only have their own stack memory and registers. Therefore, any changes made by one thread to a global variable will be visible to other threads as well.

Problem#1:

CODE:

```

/* Box #1: Simple Child Thread */

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

void *ChildThread(void *argument)
{
    int i;

    for ( i = 1; i <= 20; ++i ){

```

```

        printf(" Child Count - %d\n", i);
    }
    pthread_exit(NULL);
}

int main(void)
{
    pthread_t  hThread;    int  ret;

    ret=pthread_create(&hThread, NULL, (void *)ChildThread, NULL); /* Create
Thread */

    if (ret < 0)
        printf("Thread Creation Failed\n");  return 1;

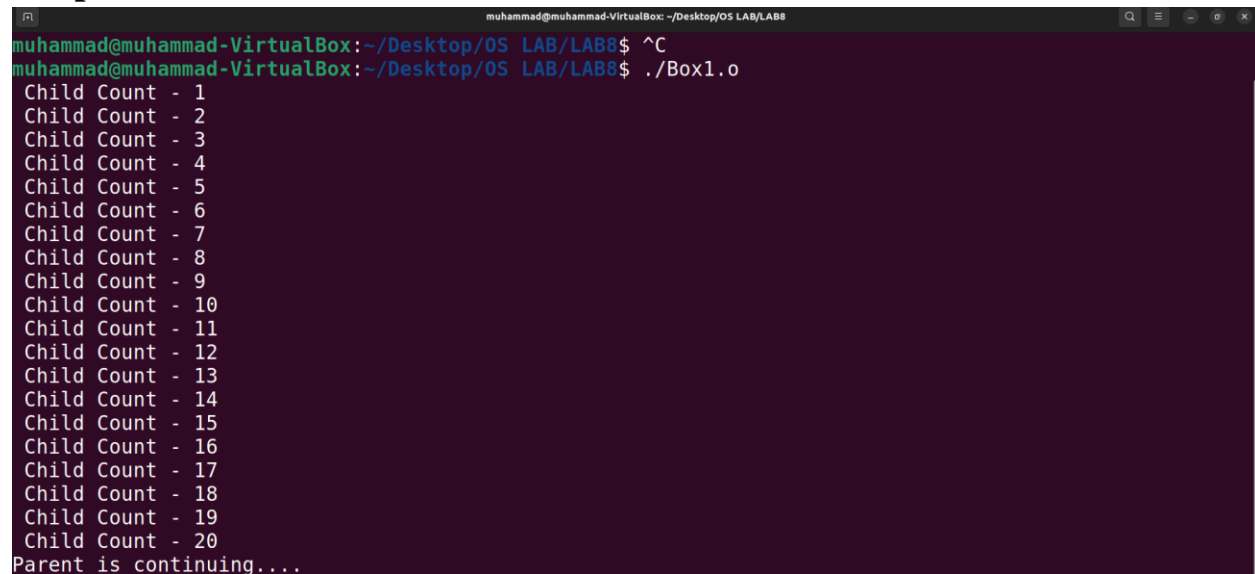
    pthread_join (hThread, NULL); /* Parent waits for */

    printf("Parent is continuing...\n");

return 0;
}

```

Output:



```

muhammad@muhammad-VirtualBox: ~/Desktop/OS LAB/LAB8
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB8$ ^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB8$ ./Box1.o
Child Count - 1
Child Count - 2
Child Count - 3
Child Count - 4
Child Count - 5
Child Count - 6
Child Count - 7
Child Count - 8
Child Count - 9
Child Count - 10
Child Count - 11
Child Count - 12
Child Count - 13
Child Count - 14
Child Count - 15
Child Count - 16
Child Count - 17
Child Count - 18
Child Count - 19
Child Count - 20
Parent is continuing....

```

Compile and execute the Box #1 program and show the output and explain why the output is so?

Answer: The **main** function creates a new thread using the **pthread_create** function and passes the **ChildThread** function as an argument. This function is then executed by the newly created thread and prints out the numbers from 1 to 20.

After creating the new thread, the **main** function waits for the child thread to finish using the **pthread_join** function. Once the child thread has finished executing, the main function continues and prints out “Parent is continuing...”.

Problem#2:

CODE:

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

```
1#include <pthread.h>
2#include <stdio.h>
3
4void ChildThread (int argument)
5{
6    int i;
7
8    for ( i = 1; i <= 20; ++i ){
9        printf(" Child Count - %d\n", i);
10    }
11/* No pthread_exit function */
12
13int main(void)
14{
15    pthread_t  hThread; int ret;
16
17    ret=pthread_create(&hThread, NULL, (void *)ChildThread, NULL); /* Create Thread */
18
19    if (ret < 0){printf("Thread Creation Failed\n"); return 1;}
20
21    pthread_join (hThread, NULL);
22    printf ("Master thread is continuing...\n");
23    return 0;}
```

Output:

```
muhammad@muhammad-VirtualBox: ~/Desktop/OS LAB/LAB8
muhammad@muhammad-VirtualBox: ~/Desktop/OS LAB/LAB8$ ^C
muhammad@muhammad-VirtualBox: ~/Desktop/OS LAB/LAB8$ ./box2.o
Child Count - 1
Child Count - 2
Child Count - 3
Child Count - 4
Child Count - 5
Child Count - 6
Child Count - 7
Child Count - 8
Child Count - 9
Child Count - 10
Child Count - 11
Child Count - 12
Child Count - 13
Child Count - 14
Child Count - 15
Child Count - 16
Child Count - 17
Child Count - 18
Child Count - 19
Child Count - 20
Master thread is continuing....
```

Answer: The **pthread_exit** function has been removed from the **ChildThread** function. When you compile and run this program, you should see the same output as in the previous version of the program.

The reason for this is that when a thread finishes executing its start routine (in this case, the **ChildThread** function), it implicitly calls **pthread_exit** and terminates. This means that even if you don't explicitly call **pthread_exit** in your thread function, it will still be called automatically when the thread finishes executing.

So in this case, removing the **pthread_exit** function from the **ChildThread** function does not change the behavior of the program.