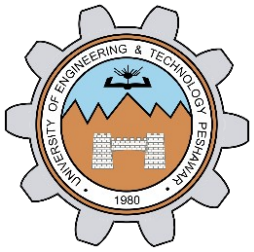
THREADS CREATION AND EXECUTION

LAB # 7



Spring 2020

CSE204L Operating Systems Lab

Submitted by: **SHAH RAZA**

Registration No. : **18PWCSE1658**

Class Section: **B**

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

Student Signature: _____

Submitted to:

Engr. Mian Ibad Ali Shah

Wednesday, 8 July, 2020

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Lab Objective(s):

This lab examines aspects of **threads** and **multiprocessing** (and **multithreading**). The primary objective of this lab is to implement the Thread Management.

Example # 01:

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

Code:

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

void *kidfunc(void *p)
{
    printf("Kid id is -----> %d\n",getpid());
}

int main()
{
    pthread_t kid;
    pthread_create(&kid,NULL,kidfunc,NULL);
    printf("Parent id is -----> %d\n",getpid());
    pthread_join(kid,NULL);
    printf("No more kid!\n");

}
```

Output :

```
Shah-Raza@ubuntu:~/Work/OS/Lab 7/Example 1$ ./example1
Parent id is -----> 1771
Kid id is -----> 1771
No more kid!
```

Observations:

Yes, the process id numbers for parent and child thread are the same because a thread is also a part of the process.

Example # 02:

Do the threads have separate copies of glob_data? Why? Or why not?

Code:

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

int glob_data=5;

void* kidfunc(void*p)
{
    printf("Kid here.Global data was %d.\n",glob_data);
    glob_data=15;
    printf("Kid Again.Global data was now %d.\n",glob_data);
}

int main()
{
    pthread_t kid;
    pthread_create(&kid,NULL,kidfunc,NULL);
    printf("Parent here.Global data = %d.\n",glob_data);
    glob_data=10;
    pthread_join(kid,NULL);
    printf("End of program.Global data = %d.\n",glob_data);
}
```

Output :

```
Shah-Raza@ubuntu:~/Work/OS/Lab 7/Example 2$ ./example2
Parent here.Global data = 5.
Kid here.Global data was 10.
Kid Again.Global data was now 15.
End of program.Global data = 15.
```

Observations:

No, threads do not have different copies of global data because they share the same memory. Although each thread has its own stack but their global data is same as global data is stored in memory and not in stack.

Example # 03:

Code:

```c
#include<pthread.h>
#include<stdio.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid)
{
    printf("\n%ld:Hello World!.\n",threadid);
    pthread_exit(NULL);
}

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


    }
    pthread_exit(NULL);

}
```

Output :

```
Shah-Raza@ubuntu:~/Work/OS/Lab 7/Example 3$ ./example3
Creating thread 0
Creating thread 1
Creating thread 2
Creating thread 3
Creating thread 4

4:Hello World!.

3:Hello World!.

2:Hello World!.

1:Hello World!.

0:Hello World!.
```

Example # 04:

Code:

```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
int this_is_global;
void thread_func( void *ptr );
int main() {
int local_main;
int pid,status;
pthread_t thread1,thread2;
printf("First, we create two threads to see better what context they share...\n");
this_is_global=1000;
printf("Set this_is_global=%d\n",this_is_global);
pthread_create(&thread1,NULL,(void*)&thread_func,(void*)NULL);
pthread_create(&thread2,NULL,(void*)&thread_func,(void*)NULL);
pthread_join(thread1,NULL);
pthread_join(thread2,NULL);
printf("After threads, this_is_global=%d\n",this_is_global);
printf("\n");
printf("Now that the threads are done, let's call fork..\n");
local_main=17; this_is_global=17;
printf("Before fork(), local_main=%d, this_is_global=%d\n",local_main,this_is_global);
pid=fork();
if (pid == 0) { /* this is the child */
    printf("In child, pid %d: &global: %X, &local: %X\n", getpid(), &this_is_global, &local_main);
    local_main=13; this_is_global=23;
    printf("Child set local main=%d, this_is_global=%d\n",local_main, this_is_global);
    exit(0);
}
else { /* this is parent */
    printf("In parent, pid %d: &global: %X, &local: %X\n", getpid(), &this_is_global, &local_main);
    wait(&status);
    printf("In parent, local_main=%d, this_is_global=%d\n",local_main,this_is_global);
}
exit(0);
}
void thread_func(void *dummy) {
int local_thread;
printf("Thread %d, pid %d, addresses: &global: %X, &local: %X\n",pthread_self(), getpid(), &this_is_global, &local_thread);
this_is_global++;
printf("In Thread %d, incremented this_is_global=%d\n", pthread_self(),this_is_global);
pthread_exit(0);
}
```

Output :

```
Shah-Raza@ubuntu:~/Work/OS/Lab 7/Example 4$ ./example4
First, we create two threads to see better what context they share...
Set this_is_global=1000
Thread 556537600, pid 2350, addresses: &global: 272E9014, &local: 212C0ED4
In Thread 556537600, incremented this_is_global=1001
Thread 564930304, pid 2350, addresses: &global: 272E9014, &local: 21AC1ED4
In Thread 564930304, incremented this_is_global=1002
After threads, this_is_global=1002

Now that the threads are done, let's call fork..
Before fork(), local_main=17, this_is_global=17
In parent, pid 2350: &global: 272E9014, &local: 2D8E2FCC
In child, pid 2353: &global: 272E9014, &local: 2D8E2FCC
Child set local main=13, this_is_global=23
In parent, local_main=17, this_is_global=17
```

Task # 01:

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

Code:

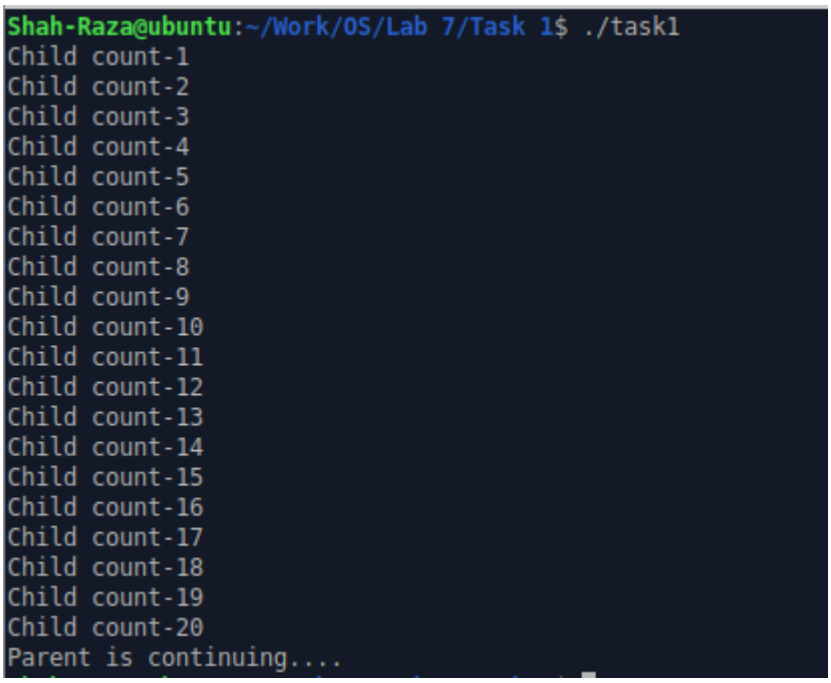
```c
#include <pthread.h>
#include <stdio.h>

void *ChildThread(void *argument)
{
    int i;
    for(i=1;i<=20;++i)
        printf("Child count-%d\n",i);
    pthread_exit(NULL);
}

int main()
{
    pthread_t hThread;
    int ret;
    ret=pthread_create(&hThread,NULL,(void *)ChildThread,NULL);
    if(ret<0)
    {
        printf("Thread creation Failed\n");
        return 1;
    }
    pthread_join(hThread,NULL);
    printf("Parent is continuing....\n");
    return 0;

}
```

Output :



Observations:

Inside main a thread is created and the ChildThread function is passed to it . Due to the pthread_join function the main thread waits for the child thread to terminate. A for loop is implemented inside the child thread which counts from 1 to 20.

Task # 02:

In the **Box #2** 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.

Code:

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

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

int main()
{
    pthread_t hThread;
    int ret,count;
    printf("Enter the Thread Count: ");
    scanf("%d",&count);
    ret=pthread_create(&hThread,NULL,(void *)ChildThread,count);
    if(ret<0)
    {
        printf("Thread creation Failed\n");
        return 1;
    }
    pthread_join(hThread,NULL);
    printf("Parent is continuing....\n");
    return 0;

}
```

Output :

```
Shah-Raza@ubuntu:~/Work/OS/Lab 7/Task 2$ ./task2
Enter the Thread Count: 15
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
Parent is continuing....
```

Observations:

In this program we are passing the count as an argument to the child thread. A for loop is implemented inside the child thread which counts from 1 to count.

Task # 03:

Write a program Box #3 by removing pthread_exit function from child thread function and check the output? Is it the same as output of Box #2? If so Why? Explain?

Code:

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

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

int main()
{
    pthread_t hThread;
    int ret,count;
    printf("Enter the Thread Count: ");
    scanf("%d",&count);
    ret=pthread_create(&hThread,NULL,(void *)ChildThread,count);
    if(ret<0)
    {
        printf("Thread creation Failed\n");
        return 1;
    }
    pthread_join(hThread,NULL);
    printf("Parent is continuing....\n");
    return 0;

}
```

Output :

```
Shah-Raza@ubuntu:~/Work/OS/Lab 7/Task 3$ ./task3
Enter the Thread Count: 15
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
Parent is continuing....
```

Observations:

After removing pthread_exit function from the given program the output is same as Task#2 because even if pthread_exit function is not called explicitly in the child thread, when the child thread is done with its task it calls pthread_exit implicitly.