National University of Computer and Emerging Sciences

Operating System Lab – 07

MULTITHREADING

Contents

Objective

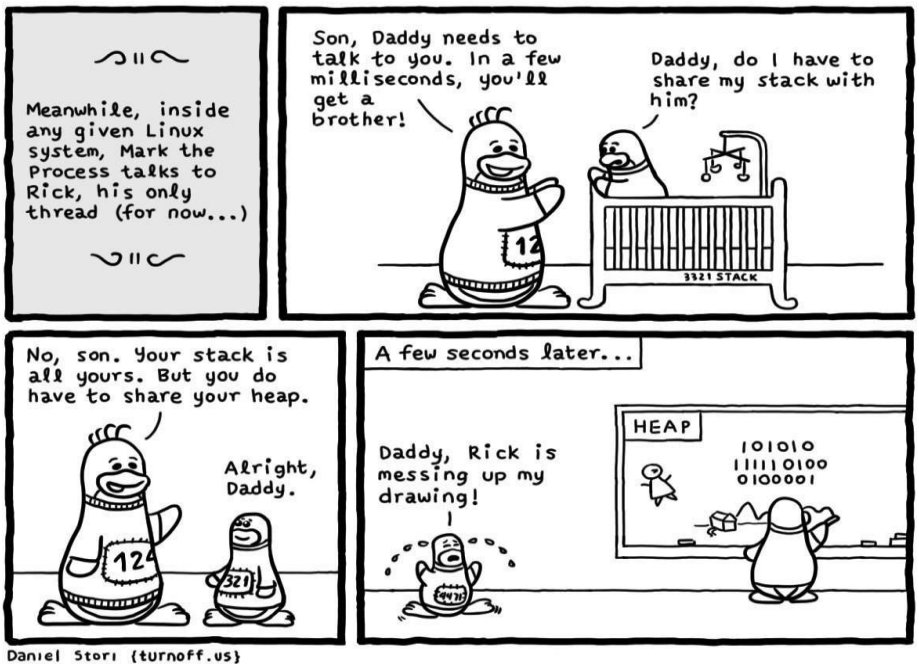
Introduction to multithreaded programming using **POSIX thread (pthread) libraries**.

Our main objectives are:

1. Thread creation in Linux
2. Joining of thread in Linux
3. Initializing thread attributes
4. Setting Attribute detach state.
5. Destroying attribute

Threads

Threads are often described as **lightweight processes**. They can work like two or more processes sharing the same address space i.e., *they will work independently like processes but can share the same global variables*. They are mostly used when two tasks can be done independently without depending much on each other.



Basic System Calls Related to Multithread Programming

The following are two basic system calls related to multithreaded programming however, there are many system calls available.

S.NO	System Call	Description
1	Pthread_create()	For creating threads
2	Pthread_join()	Wait of thread termination

pthread_create() - System Call

This system call is used to create new thread, a syntax is given below.

```
#include<pthread.h>
int pthread_create(

pthread_t *threaded,              //id of thread
const pthread_attr_t *attr,       //attributes of thread
void *(*start_routine) (void*),   //function that is to assign
void *arg                         //arguments that have to pass to thread function

);
```

```
Example: pthread_create(&id[0], NULL, printNumber, &arg);
```

Arguments	Syntax	Description
ID	`pthread_t *`	Reference (or pointer) tothe ID of the thread.
attr	`pthread_attr_t *`	Used to set the attributes of a thread (e.g., the stack size, scheduling policy, etc.) Passing NULL suffices for most applications.
Starting routine	`void *`	The name of the function that the thread starts to execute. If the function's return type is void *, then its name is simply written; otherwise, it must be type-cast to void *.
arg	`void *`	This is the argument that the *starting routine* takes. If it takes multiple arguments, a struct is used.

Return Values:

If successful it returns 0 otherwise it generates a nonzero number.

pthread_join() -System Call

This system call waits for the thread specified by thread to terminate. A syntax is shown below:

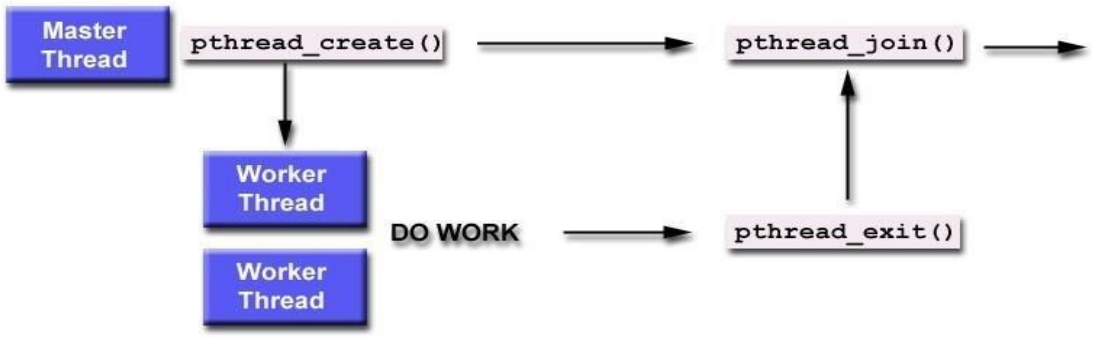
```
Int pthread_join (

Pthread_t threaded,     //id of thread which have to join
void **retval           //return status of thread

);
```

Return Values:

If successful it return 0 otherwise it generates a nonzero number.



Example 1: *Two Threads displaying two strings "Hello" and "How are you?"independent of each other*

1. Create a new file thread.c with .c extension using any editor
2. Type the following code.

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
void * thread1()
{
while(1){
printf("Hello!!\n");
}
}
void * thread2()
{
while(1){
printf("How are you?\n");
}
}
int main()
{
int status;
pthread_t tid1,tid2;
pthread_create(&tid1,NULL,thread1,NULL);
pthread_create(&tid2,NULL,thread2,NULL);
pthread_join(tid1,NULL);
pthread_join(tid2,NULL);
return 0;}
```

3. Save and exit.
4. To compile it type the following command on terminal.

```
gcc thread.c -o thread -lpthread
```

- **gcc** is the compiler command.
- **thread.c** is the name of c program source file.
- **-o** is option to make object file.
- **thread** is the name of object file.
- **-lpthread** is option to execute pthread.h library file.

5. Run it by using following command.

```
./thread
```

The –lpthread at the end to link the pthread library.

Example 2: *Create a function message() that takes threadid as argument andprints the message with thread id. There should be atleast five independent threads*

1. Create a new file msgthreads.c with .c extension using any editor
2. Type the following code.

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

void *PrintHello(void *threadid)
{
  long tid;
  tid = (long)threadid;
  printf("Hello World! It's me, thread #%ld!\n", tid);
  pthread_exit(NULL);
}

int main (int argc, char *argv[])
{
  pthread_t threads[NUM_THREADS];
  int rc;
  long t;
  for(t=0; t<NUM_THREADS; t++){
    printf("In main: creating thread %ld\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);
}
```

3. Save and exit.
4. To compile it type the following command on terminal.

```
gcc msgthreads.c –o msgthreads -lpthread
```

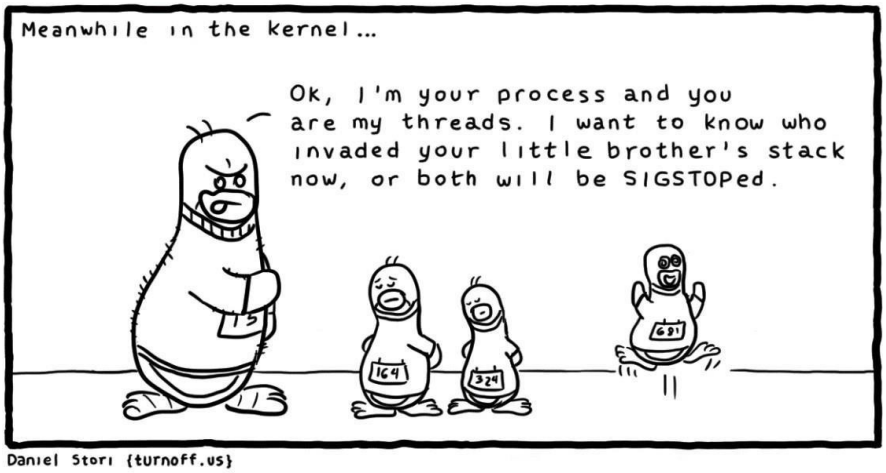
5. Run it by using following command.

```
./msgthreads
```

Note: remove pthread_join system call and then observe the changes

Attributes in Threads

Thread attributes are thread characteristics that affect the behavior of the thread. **NULL** is passed in above examples in place of thread attributes, now we start to place thread attributes that uses default attributes of threads. We may create and customize a thread attribute object to specify othervalues for the attribute.



System Calls related to Attributes of Threads

The following are the system calls related to threads' attribute.

S.NO	System Call	Description
1	pthread_attr_init()	Initializes a thread attributes object
2	pthread_attr_setdetachstate()	Controls detach state of a thread
3	pthread_attr_destroy()	Destroys attribute objects

1. **pthread_attr_init()** -System Call

This initializes a thread attributes object attr with the default value. The syntax is shown below:

```
int pthread_attr_init(pthread attr t *attr)
```

Return Values:

If successful completion, it will return a 0 otherwise, an error number is returned to indicate the error.

2. **pthread_attr_setdetachstate()** -System Call

The detachstate attribute controls whether the thread is created in a detached state.

```
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate)
```

PTHEAD_CREATE_DETACHED

Thread state is detached means it cannot be joined with other threads.

PTHREAD_CREATE_JOINABLE

Thread state is joinable means it can be joined with other threads

3. **pthread_attr_destroy()** -System Call

When a thread attributes objects is no longer required, it should be destroyed using this system call.

int pthread_attr_destroy(pthread_attr_t *attr)

Return Values:

If successful completion, it will return a 0 otherwise, an error number is returned to indicate the error.

`pthread_self()`

Syntax

```
pthread_t tid;
tid = pthread_self();
```

DESCRIPTION

The pthread_self() function shall return the thread ID of the calling thread.

RETURN VALUE

Refer to the DESCRIPTION.

ERRORS

No errors are defined. The pthread_self() function shall not return an error code.

Example 3: *Create a detached thread for a function infoThread()*

1. Create a new file detachthread.c with .c extension using any editor
2. Type the following code.

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

void *theThread(void *parm) {
        printf("Entered the thread\n");
        return NULL;
}

int main(int argc, char **argv) {
        pthread_attr_t attr;
        pthread_t thread;

        printf("Create a default thread attributes object\n");
        pthread_attr_init(&attr);
        printf("Set the detach state thread attribute\n");
        pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);

        printf("Create a thread using the new attributes\n");
        pthread_create(&thread, &attr, theThread, NULL);
        printf("Destroy thread attributes object\n");
        pthread_attr_destroy(&attr);
int rc;
rc = pthread_join(thread, NULL);

        printf("Join now fails because the detach state attribute was changed\n pthread_join returns non zero
value %d",rc);

printf("Main completed\n");
return 0;
}
```

3. Save and exit.
4. To compile it type the following command on terminal.

```
gcc detachthread.c  –o detachthread –lpthread
```

5. Run it by using following command.

```
./detachthread
```

You can get much information about these attributes and more information about system calls related to thread attributes: follow the links below

6. https://docs.oracle.com/cd/E19455-01/806-5257/6je9h032j/index.html
7. http://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html
8. https://vcansimplify.wordpress.com/2013/03/08/pthread-tutorial-simplified/

Lab Activity

1. Write a program that create 3 threads
   a) On successful creation, print “Thread #” in its starting routine and terminate themselves by showing their return value.
   b) On unsuccessful creation, Print “Error”.

```
Thread 1
Thread 2
Thread 3
Thread 1 returns: 0
Thread 2 returns: 0
Thread 3 returns: 0
```

2. Write a program, which make 4 threads. Each thread will print one table out of [5678] up to 1000.

3. Write a program that creates N number of threads specified in the command line, each prints “hello message and its own thread ID”. Sleep the main thread for 1 second and create every 4 or 5 threads. The output of your code should alike:

```
I am thread 1. Created new thread (4) in iteration 0...
Hello from thread 4 - I was created in iteration 0
I am thread 1. Created new thread (6) in iteration 1...
I am thread 1. Created new thread (7) in iteration 2...
I am thread 1. Created new thread (8) in iteration 3...
I am thread 1. Created new thread (9) in iteration 4...
I am thread 1. Created new thread (10) in iteration 5...
Hello from thread 6 - I was created in iteration 1
Hello from thread 7 - I was created in iteration 2
Hello from thread 8 - I was created in iteration 3
Hello from thread 9 - I was created in iteration 4
Hello from thread 10 - I was created in iteration 5
I am thread 1. Created new thread (11) in iteration 6...
I am thread 1. Created new thread (12) in iteration 7...
Hello from thread 11 - I was created in iteration 6
Hello from thread 12 - I was created in iteration 7
```

4. Write a program to sum 10 elements of an array by multithreading.
5. Procom has 4 volunteers on their front desk.
   - Volunteer 1 manages On day registration
   - Volunteer 2 handles announcements
   - Volunteer 3 handles sponsors
   - Volunteer 4 resolve queries of participants

   Implement this system using pthread for 100 participants.