Exp 7: Threads

1. *Objectives:*

- Introducing the concept of threads.
- Introducing PTHREAD library .
- Tracing and writing multi-threading programs.

2. *What is Thread?*

A thread is a semi-process that has its own stack, and executes a given piece of code. Unlike a real process, the thread normally shares its memory with other threads. A Thread Group is a set of threads all executing inside the same process. They all share the same memory, and thus can access the same global variables, same heap memory, same set of file descriptors, etc. All these threads execute in parallel (i.e. using time slices, or if the system has several processors, then really in parallel).

3. *Thread Management Functions*

Function	Information
`int pthread_create (pthread_t * threadhandle, /* Thread handle returned by reference */ pthread_attr_t *attribute /* Special Attribute for starting thread, may be NULL */, void *(*start_routine)(void *) /* Main Function which thread executes */, void *arg /* An extra argument passed as a pointer */);`	Request the PThread library for creation of a new thread. The return value is 0 on success. The return value is negative on failure. The pthread_t is an abstract data type that is used as a handle to reference the thread.
`int pthread_join ( pthread_t threadhandle /* Pass threadhandle */, void **returnvalue /* Return value is returned by ref. */ );`	Blocks the calling thread until a thread terminates. Return 0 on success, and negative on failure. The returned value is a pointer returned by reference. If you do not care about the return value, you can pass NULL for the second argument.
`void pthread_exit ( void *retval /* return value passed as a pointer */ );`	This Function is used by a thread to terminate. The return value is passed as a pointer. This pointer value can be anything so long as it does not exceed the size of (void *).

Example

In this part, you will learn about concurrent programming using pthreads. Below, is a simple example that shows the steps needed to create a multithreaded program. Compile the code (using -lpthread option) Then, run the code and observe the output.

```
/***************************************
Thread creation and termination
compile use: gcc -o t1 t1.c –lpthread
File Name: Thread.c
****************************************/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *print_message_function( void *ptr );
void main()
{
pthread_t thread1, thread2;
char *message1 = "Thread 1";
char *message2 = "Thread 2";
int iret1, iret2;
/* Create independent threads each of which will execute function*/
iret1 = pthread_create( &thread1, NULL, print_message_function,
(void*) message1);
iret2 = pthread_create( &thread2, NULL, print_message_function,
(void*) message2);
/* Wait until threads are complete before main continues. Unless we
wait, we run the risk of executing an exit which will terminate the
process and all threads before the threads have completed.*/
pthread_join( thread1, NULL);
pthread_join( thread2, NULL);
printf("Thread 1 returns: %d\n",iret1);
printf("Thread 2 returns: %d\n",iret2);
exit(0);
}
void *print_message_function( void *ptr )
{
char *message;
message = (char *) ptr;
printf("%s \n", message);
}
```

4. *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 *).The following example passes a simple integer to each thread.

```
/****************************************
Passing argument to Thread
compile using: -lpthread
File Name: ThreadArg.c
****************************************/
#include <stdio.h>
#include <pthread.h>
#define NUM_THREADS 5
void *Print(void *threadid)
{
int ThreadID;
threadid = (int)threadid;
printf("Hello World! Thread ID,%d \n", ThreadID);
pthread_exit(NULL);
}
int main ()
{
pthread_t threads[NUM_THREADS];
int rc;
int i;
for( i=0; i < NUM_THREADS; i++ ){
printf("main() : creating thread,%d \n" , i);
rc = pthread_create(&threads[i], NULL,Print, (void *)i);
if (rc){
printf("Error:unable to create thread,%d \n",rc);
exit(-1);
}
}
pthread_exit(NULL);
}
```

5. Exercise:

Trace the following program and write down your expected output .

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

void *function1( );  //functions prototype
void *function2( );

int pid;
void main()
{
// so for there is single process with single thread
pthread_t  thr1, thr2; // prepare two threads two threads

pid = fork(); // a new process is created....

if(pid ==0)
{
pthread_create(&thr1,NULL,* function1, NULL);
pthread_join(thr1,NULL);
}
else
{
wait(NULL);//force the parent process to wait the child

pthread_create(&thr2,NULL,* function2, NULL);
pthread_join(thr2,NULL);
}

exit(0);
}

void *function1(  )
{
printf("The PID of this process is %d \n",pid);
int i=0;
for(;i<3;i++)
printf("B\n");
}

void *function2(  )
{
printf("The PID of this process is %d \n",pid);
printf("A \n");    }
```