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

void *print_message_function( void *ptr);

int main( )
{  pthread_t  thread1, thread2;
   char *msg1 = "Thread 1";
   char *msg2 = "Thread 2";
   int  iret1, iret2;

  iret1 = pthread_create( &thread1, NULL, print_message_function, (void *) msg1);
 iret2 = pthread_create( &thread2, NULL, print_message_function, (void *) msg2);
 /* wait till 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);
 }
```

```
gcc  -lpthread  pthreadprog.c
```

Posix Threads

Pthreads

```
#include <iostream>
#include <cstdlib>
#include <pthread.h>
using namespace std;
#define NTHREADS 8
void *helloWorld(~~void~~ * threadid)
{   long tid;
    tid = (long) threadid;
    cout << "Hello World! My thread id is" << tid << endl;
   pthread_exit(NULL);
}

int main( )
{     pthread_t   threads[NTHREADS];
      int rc;
      int i;
     for(i = 0;  i < NTHREADS; i++)
     {    cout << "main: creating thread 00" << i << endl;
          error = pthread create( &threads[i], NULL, helloWord, (void *) i);
          if (error)
          {  cout << "Error: unable to create thread" << error << endl;
             exit(-1);
          }
       pthread_exit(NULL);
     }
```

int pthread_create((returns zero when the call completes successfully.)

pthread_t *thread,

thread – returns the thread id (unsigned long int)

const pthread_attr_t *attr,

(attr – Set to NULL if default thread attributes are used

example PTHREAD_CREATE_JOINABLE
PTHREAD_CREATE_DETACHED } detached State

scheduling policy, scheduling parameters

scope : Kernel thread, User thread)

void * (*start_routine) – pointer to the function to be threaded.
Function has a single argument pointer to void.

void *arg);

pointer to argument of function. To pass multiple arguments, send a pointer to structure

---

void pthread_exit (void * retval)

retval – return value of thread

This routine kills the thread.

mutexes

(pthread_mutex_t
mutex1 = PTHREAD_MUTEX_INITIALIZER)

pthread_mutex_lock(& mutex1)

counter++

pthread_mutex_unlock(& mutex1);

Thread Synchronization

Thread library provides three synchronization mechanisms.

- mutexes : Mutual exclusion lock – Block access to variables by other threads. This enforce exclusive access by a thread to a variable or set of variables
- Joins : Make a thread wait till others are complete (terminated)
- condition variables. data type pthread_cond_t

Joiny :- A join is performed when one wants to wait for a thread to finish.

A thread calling routine may launch multiple threads then wait for them to finish to get the results.

One wait for the completion of the threads with a join.

```
  #include <stdio.h>
 #include <pthread.h>
#define NTHREADS  10
void *thread_function(void *);
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
 int counter = 0;
int main()
{  pthread_t  thread_id[NTHREADS];
   int i, j;
   for(i=0; i<NTHREADS; i++)
   {
      pthread_create( &thread_id[i], NULL, thread_function, NULL);
   }
   for( j=0;  j < NTHREAD; j++)
   {  pthread_join( thread_id[j], NULL);
   }
/* Now that all threads are complete, I can print the final result.
 Without the join, I could be printing a value before all the threads
 have been completed. */
 printf("Final Counter value: %d\n", counter);
}
```

```
void *thread_function( void *dummyptr)
{  printf("Thread number %ld\n", pthread_self());
   pthread_mutex_lock( &mutex1);
   counter++;
   pthread_mutex_unlock( &mutex1);
}
```

```
int  pthread_mutex_destroy( pthread_mutex_t *mutex)


int  pthread_detach( pthread_t thread, void **value_ptr)
```
