/* Thread creation and termination */
/* Compile using gcc -lpthread program_name.c */

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

void *print_message_function( void *ptr );

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 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);
}

OUTPUT:
Thread 1
Thread 2
Thread 1 returns: 0
Thread 2 returns: 0
/* Thread synchronization using MUTEX */

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

void *functionC();
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int counter = 0;

main()
{
    int rc1, rc2;
    pthread_t thread1, thread2;

    /* Create independent threads each of which will execute functionC */
    if( (rc1=pthread_create( &thread1, NULL, &functionC, NULL)) )
    {
        printf("Thread creation failed: %d\n", rc1);
    }
    if( (rc2=pthread_create( &thread2, NULL, &functionC, NULL)) )
    {
        printf("Thread creation failed: %d\n", rc2);
    }

    /* 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);
    exit(0);
}

void *functionC()
{
    pthread_mutex_lock( &mutex1 );
    counter++;
    printf("Counter value: %d\n",counter);
    pthread_mutex_unlock( &mutex1 );
}

OUTPUT

    Counter value: 1
    Counter value: 2

When a mutex lock is attempted against a mutex which is held by another thread, the thread is
blocked until the mutex is unlocked. When a thread terminates, the mutex does not unless explicitly
unlocked. Nothing happens by default.