Program 1:

```
/******************************************************************************
 * * FILE: hello.c
 * * DESCRIPTION:
 * *   A "hello world" Pthreads program.  Demonstrates thread creation and
 * *   termination.
 * * AUTHOR: Blaise Barney
 * * LAST REVISED: 08/09/11
******************************************************************************/
#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);
      }
    }
  /* Last thing that main() should do */
  pthread_exit(NULL);
}
```

Program 2

```
/******************************************************************************
 * * FILE: hello_arg2.c
 * * DESCRIPTION:
 * *   A "hello world" Pthreads program which demonstrates another safe way
 * *   to pass arguments to threads during thread creation.  In this case,
 * *   a structure is used to pass multiple arguments.
 * * AUTHOR: Blaise Barney
 * * LAST REVISED: 01/29/09
******************************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     8

char *messages[NUM_THREADS];
struct thread_data
{
  int  thread_id;
  int  sum;
```

```
   char *message;
};
struct thread_data thread_data_array[NUM_THREADS];

void *PrintHello(void *threadarg)
{
   int taskid, sum;
   char *hello_msg;
   struct thread_data *my_data;

   sleep(1);
   my_data = (struct thread_data *) threadarg;
   taskid = my_data->thread_id;
   sum = my_data->sum;
   hello_msg = my_data->message;
   printf("Thread %d: %s  Sum=%d\n", taskid, hello_msg, sum);
   pthread_exit(NULL);
}
int main(int argc, char *argv[])
{
pthread_t threads[NUM_THREADS];
int *taskids[NUM_THREADS];
int rc, t, sum;

sum=0;
messages[0] = "English: Hello World!";
messages[1] = "French: Bonjour, le monde!";
messages[2] = "Spanish: Hola al mundo";
messages[3] = "Klingon: Nuq neH!";
messages[4] = "German: Guten Tag, Welt!";
messages[5] = "Russian: Zdravstvytye, mir!";
messages[6] = "Japan: Sekai e konnichiwa!";
messages[7] = "Latin: Orbis, te saluto!";

for(t=0;t<NUM_THREADS;t++) {
```

```
  sum = sum + t;
  thread_data_array[t].thread_id = t;
  thread_data_array[t].sum = sum;
  thread_data_array[t].message = messages[t];
  printf("Creating thread %d\n", t);
  rc = pthread_create(&threads[t], NULL, PrintHello, (void *)
       &thread_data_array[t]);
  if (rc) {
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
    }
  }
pthread_exit(NULL);
}
```

Program 3:

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

pthread_t tid[2];
int counter;
pthread_mutex_t lock;

void* trythis(void* arg)
{
    pthread_mutex_lock(&lock);

    unsigned long i = 0;
    counter += 1;
    printf("\n Job %d has started\n", counter);
        /* 4294967295 hexadecimal value*/
```

```
    for (i = 0; i < (0xFFFFFFFF); i++) ;

    printf("\n Job %d has finished\n", counter);
    pthread_mutex_unlock(&lock);
    return NULL;
}

int main(void)
{
    int i = 0;
    int error;

    if (pthread_mutex_init(&lock, NULL) != 0) {
        printf("\n mutex init has failed\n");
        return 1;
    }

    while (i < 2) {
        error = pthread_create(&(tid[i]),NULL,trythis,NULL);
        if (error != 0)
            printf("\nThread can't be created :[%s]",
                strerror(error));
        i++;
    }

    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    pthread_mutex_destroy(&lock);

    return 0;
}
```

Program 4:

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

#define NUM_THREADS  3
#define TCOUNT 10
#define COUNT_LIMIT 12

int     count = 0;
pthread_mutex_t count_mutex;
pthread_cond_t count_threshold_cv;

void *inc_count(void *t)
{
  int i;
  long my_id = (long)t;

  for (i=0; i < TCOUNT; i++) {
    pthread_mutex_lock(&count_mutex);
    count++;

    /* *   Check the value of count and signal waiting thread when condition is
*       reached.  Note that this occurs while mutex is locked.**/

    if (count == COUNT_LIMIT) {
      printf("inc_count(): thread %ld, count = %d  Threshold reached. ",
            my_id, count);
      pthread_cond_signal(&count_threshold_cv);
      printf("Just sent signal.\n");
      }
    printf("inc_count(): thread %ld, count = %d, unlocking mutex\n",
          my_id, count);
    pthread_mutex_unlock(&count_mutex);
```

```
    /* Do some work so threads can alternate on mutex lock */
    sleep(1);
    }
  pthread_exit(NULL);
}

void *watch_count(void *t)
{
  long my_id = (long)t;
  printf("Starting watch_count(): thread %ld\n", my_id);

  /**   Lock mutex and wait for signal.  Note that the pthread_cond_wait routine
  *     will automatically and atomically unlock mutex while it waits.
  *       Also, note that if COUNT_LIMIT is reached before this routine is run by
  *         the waiting thread, the loop will be skipped to prevent pthread_cond_wait
  *           from never returning.* */

  pthread_mutex_lock(&count_mutex);
  while (count < COUNT_LIMIT) {
    printf("watch_count(): thread %ld Count= %d. Going into wait...\n",
my_id,count);
    pthread_cond_wait(&count_threshold_cv, &count_mutex);
    printf("watch_count(): thread %ld Condition signal received. Count= %d\n",
my_id,count);
    printf("watch_count(): thread %ld Updating the value of count...\n",
my_id,count);
    count += 125;
    printf("watch_count(): thread %ld count now = %d.\n", my_id, count);
    }
  printf("watch_count(): thread %ld Unlocking mutex.\n", my_id);
  pthread_mutex_unlock(&count_mutex);
  pthread_exit(NULL);
}
```

```
int main(int argc, char *argv[])
{
  int i, rc;
  long t1=1, t2=2, t3=3;
  pthread_t threads[3];
  pthread_attr_t attr;

  /* Initialize mutex and condition variable objects */
  pthread_mutex_init(&count_mutex, NULL);
  pthread_cond_init (&count_threshold_cv, NULL);

  /* For portability, explicitly create threads in a joinable state */
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
  pthread_create(&threads[0], &attr, watch_count, (void *)t1);
  pthread_create(&threads[1], &attr, inc_count, (void *)t2);
  pthread_create(&threads[2], &attr, inc_count, (void *)t3);

  /* Wait for all threads to complete */
  for (i = 0; i < NUM_THREADS; i++) {
    pthread_join(threads[i], NULL);
  }
  printf ("Main(): Waited and joined with %d threads. Final value of count = %d.
Done.\n",NUM_THREADS, count);

  /* Clean up and exit */
  pthread_attr_destroy(&attr);
  pthread_mutex_destroy(&count_mutex);
  pthread_cond_destroy(&count_threshold_cv);
  pthread_exit (NULL);

}
```