Pthread programs

1.Hello program using Pthreads.

```
#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);
        }// end of if
    }// end of for
    pthread_exit(NULL);
}
```

2. 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.

```
#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);
}
```

3. Example code demonstrating decomposition of array processing by distributing loop iterations. A global sum is maintained by a mutex variable.

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

#define NTHREADS      4
#define ARRAYSIZE   1000000
#define ITERATIONS   ARRAYSIZE / NTHREADS

double  sum=0.0, a[ARRAYSIZE];
pthread_mutex_t sum_mutex;


void *do_work(void *tid)
{
     int i, start, *mytid, end;
       double mysum=0.0;

     /* Initialize my part of the global array and keep local sum */
       mytid = (int *) tid;
       start = (*mytid * ITERATIONS);
       end = start + ITERATIONS;
       printf ("Thread %d doing iterations %d to %d\n",*mytid,start,end-1);
       for (i=start; i < end ; i++) {
          a[i] = i * 1.0;
          mysum = mysum + a[i];
       }

     /* Lock the mutex and update the global sum, then exit */
       pthread_mutex_lock (&sum_mutex);

           sum = sum + mysum;
       pthread_mutex_unlock (&sum_mutex);
       pthread_exit(NULL);
 }

int main(int argc, char *argv[])
{
     int i, start, tids[NTHREADS];
     pthread_t threads[NTHREADS];
     pthread_attr_t attr;

   /* Pthreads setup: initialize mutex and explicitly create threads in a
    * joinable state (for portability).  Pass each thread its loop offset */

     pthread_mutex_init(&sum_mutex, NULL);
     pthread_attr_init(&attr);
     pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
```

```
    for (i=0; i<NTHREADS; i++) {
      tids[i] = i;
      pthread_create(&threads[i], &attr, do_work, (void *) &tids[i]);
    }

  /* Wait for all threads to complete then print global sum */
    for (i=0; i<NTHREADS; i++) {
      pthread_join(threads[i], NULL);
    }
    printf ("Done. Sum= %e \n", sum);

    sum=0.0;
    for (i=0;i<ARRAYSIZE;i++){
      a[i] = i*1.0;
      sum = sum + a[i];
    }
      printf("Check Sum= %e\n",sum);

      /* Clean up and exit */
      pthread_attr_destroy(&attr);
      pthread_mutex_destroy(&sum_mutex);
      pthread_exit (NULL);
}
```

4. This example demonstrates how to explicitly create a thread in a detached state. This might be done to conserve some system resources if the thread never needs to join later. Compare with the join.c program where the threads are created joinable.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NUM_THREADS     4

void *BusyWork(void *t)
{
  long i, tid;
  double result=0.0;
  tid = (long)t;
  printf("Thread %ld starting...\n",tid);
  for (i=0; i<1000000; i++) {
      result = result + sin(i) * tan(i);
  }
  printf("Thread %ld done. Result = %e\n",tid, result);
}

int main(int argc, char *argv[])
{
   pthread_t thread[NUM_THREADS];
```

```
    pthread_attr_t attr;
    int rc;
    long t;

    /* Initialize and set thread detached attribute */
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

    for(t=0;t<NUM_THREADS;t++) {
          printf("Main: creating thread %ld\n", t);
         rc = pthread_create(&thread[t], &attr, BusyWork, (void *)t);
        if (rc) {
            printf("ERROR; return code from pthread_create() is %d\n", rc);
             exit(-1);
        }
    }

    /* We're done with the attribute object, so we can destroy it */
    pthread_attr_destroy(&attr);

  /* The main thread is done, so we need to call pthread_exit explicitly to
   * *  permit the working threads to continue even after main completes.*/

    printf("Main: program completed. Exiting.\n");
    pthread_exit(NULL);
}
```