|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>    #define NUM\_THREADS 5    /\* create thread argument struct for thr\_func() \*/  typedef struct \_thread\_data\_t {    int tid;    double stuff;  } thread\_data\_t;    /\* shared data between threads \*/  double shared\_x;  pthread\_mutex\_t lock\_x;    void \*thr\_func(void \*arg) {    thread\_data\_t \*data = (thread\_data\_t \*)arg;      printf("hello from thr\_func, thread id: %d\n", data->tid);    /\* get mutex before modifying and printing shared\_x \*/    pthread\_mutex\_lock(&lock\_x);      shared\_x += data->stuff;      printf("x = %f\n", shared\_x);    pthread\_mutex\_unlock(&lock\_x);      pthread\_exit(NULL);  }    int main(int argc, char \*\*argv) {    pthread\_t thr[NUM\_THREADS];    int i, rc;    /\* create a thread\_data\_t argument array \*/    thread\_data\_t thr\_data[NUM\_THREADS];      /\* initialize shared data \*/    shared\_x = 0;      /\* initialize pthread mutex protecting "shared\_x" \*/    pthread\_mutex\_init(&lock\_x, NULL);      /\* create threads \*/    for (i = 0; i < NUM\_THREADS; ++i) {      thr\_data[i].tid = i;      thr\_data[i].stuff = (i + 1) \* NUM\_THREADS;      if ((rc = pthread\_create(&thr[i], NULL, thr\_func, &thr\_data[i]))) {        fprintf(stderr, "error: pthread\_create, rc: %d\n", rc);        return EXIT\_FAILURE;      }    }    /\* block until all threads complete \*/    for (i = 0; i < NUM\_THREADS; ++i) {      pthread\_join(thr[i], NULL);    }      return EXIT\_SUCCESS;  } |

In the above example code, we add some shared data called shared\_x and ensure serialized access to this variable through a mutex named lock\_x. Within the thr\_func() we call pthread\_mutex\_lock() before reading or modifying the shared data. Note that we continue to maintain the lock even through the printf() function call as releasing the lock before this and printing can lead to inconsistent results in the output. Recall that the code in-between the lock and unlock calls is called a critical section. Critical sections should be minimized for increased concurrency.