Mutex (MUTual EXclusion)

Concurrent Programming

Introduction

- What is Mutex?
- Pthreads Mutex API
- Examples

What is Mutex?

- Synchronization mechanism for enforcing limits on access to a resource in an environment where there are many threads of execution

CRITICAL SECTION
MUTEX

Photo reference: http://www.rudyhuyn.com/blog/2015/12/31/synchroniser-ses-agents-avec-lapplication/mutex/

Pthreads Mutex API

- pthread_mutex_init
  - More APIs related to mutex attribute
- pthread_mutex_lock
- pthread_mutex_trylock
- pthread_mutex_unlock

Pthread Mutex API – pthread_mutex_init

```
int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t *mutexattr);
```

- Initialize the mutex object

```
@param[in]  mutex          Mutex to be initialized
@param[in]  mutexattr      Used for setting attributes of a mutex.(e.g.,Deadlock Checking)
                           Default 0
@return                    0 on success
```

Pthread Mutex API – pthread_mutexattr

```
int pthread_mutexattr_init(
                          pthread_mutexattr_t *mattr);
int pthread_mutexattr_destroy(
                          pthread_mutexattr_t *mattr);
```

- Initialize the mutex object

```
@param[in]  mutexattr      Mutex attributes object
@return                    0 on success
```

Pthread Mutex API – pthread_mutexattr

```
int pthread_mutexattr_settype(
                              pthread_mutexattr_t *mattr,
                              int type);
```

- set type of mutex attributes object

```
@param[in]  mutexattr     Mutex attributes object
@param[in]  type          Mutex type attributes value
                          PTHREAD_MUTEX_NORMAL, PTHREAD_MUTEX_ERRORCHECK,
                          PTHREAD_MUTEX_RECURSIVE, PTHREAD_MUTEX_DEFAULT

@return                    0 on success
```

Pthread Mutex API – pthread_mutex_lock

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

- Lock the mutex object. If the mutex is already locked, the calling thread shall block until the mutex becomes available.

```
@param[in]  mutex          Mutex to be locked
@return                    0 if acquired. Error number related to the mutexattr if failed.
```

Pthread Mutex API – pthread_mutex_trylock

```
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```

- Lock the mutex object. If the mutex is already locked, return immediately.

```
@param[in]  mutex          Mutex to be locked
@return                    0 if acquired. Error number related to the mutexattr if failed.
```

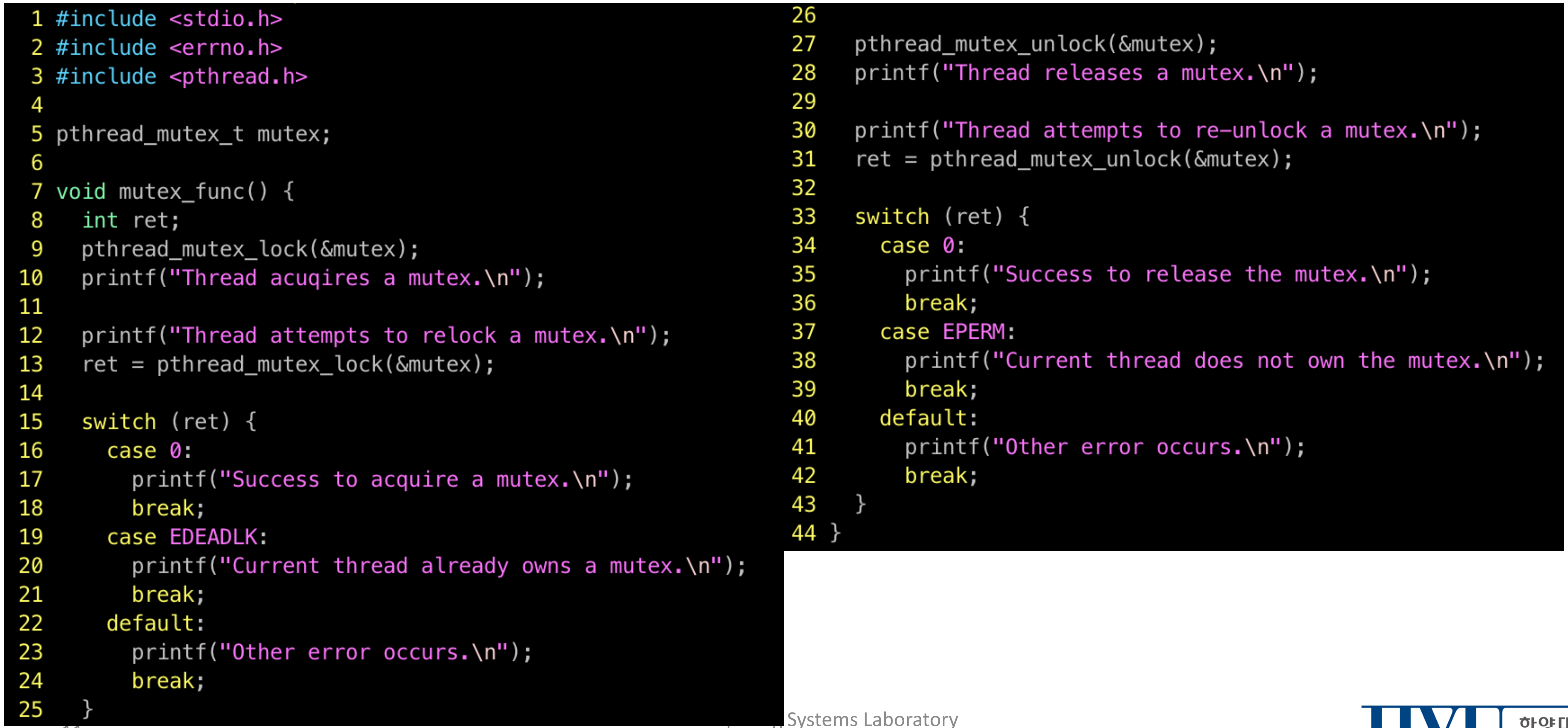
Pthread Mutex API – pthread_mutex_unlock

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- Release the mutex object.

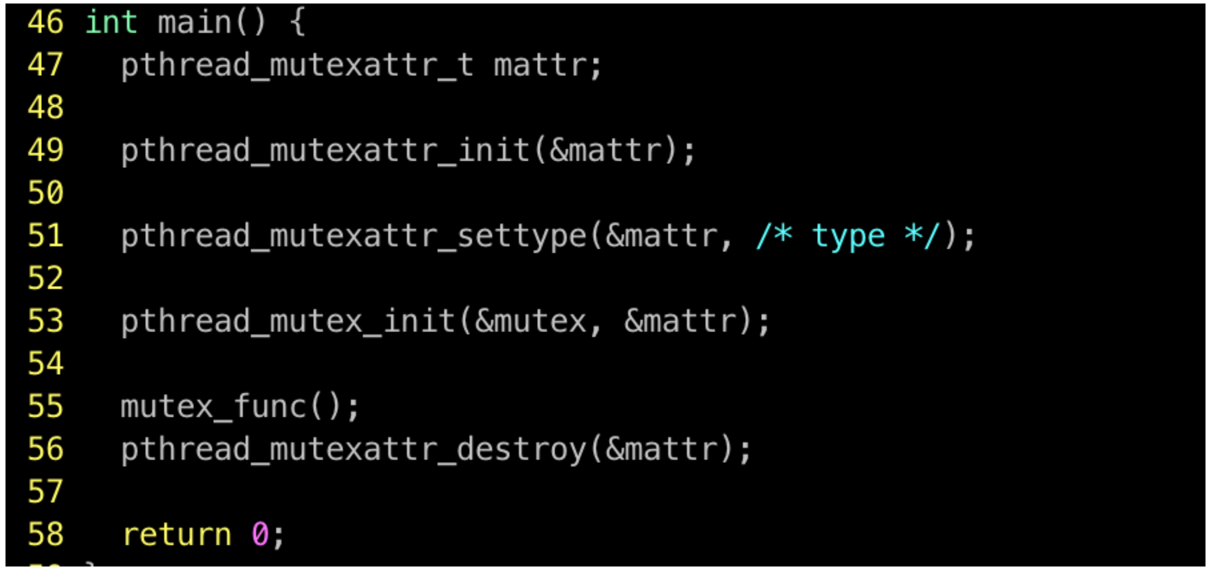
```
@param[in]  mutex          Mutex to be released
@return                    0 if released. Error number related to the mutexattr if failed.
```

Example1



```c
#include <stdio.h>
#include <errno.h>
#include <pthread.h>

pthread_mutex_t mutex;

void mutex_func() {
  int ret;
  pthread_mutex_lock(&mutex);
  printf("Thread acuqires a mutex.\n");

  printf("Thread attempts to relock a mutex.\n");
  ret = pthread_mutex_lock(&mutex);

  switch (ret) {
    case 0:
      printf("Success to acquire a mutex.\n");
      break;
    case EDEADLK:
      printf("Current thread already owns a mutex.\n");
      break;
    default:
      printf("Other error occurs.\n");
      break;
  }

  pthread_mutex_unlock(&mutex);
  printf("Thread releases a mutex.\n");

  printf("Thread attempts to re-unlock a mutex.\n");
  ret = pthread_mutex_unlock(&mutex);

  switch (ret) {
    case 0:
      printf("Success to release the mutex.\n");
      break;
    case EPERM:
      printf("Current thread does not own the mutex.\n");
      break;
    default:
      printf("Other error occurs.\n");
      break;
  }
}
```

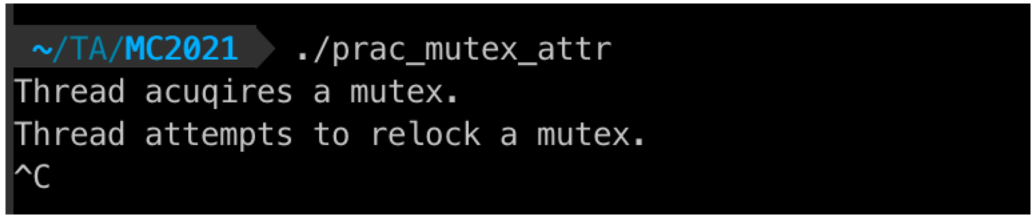
Example1 (cont.)



```c
int main() {
  pthread_mutexattr_t mattr;

  pthread_mutexattr_init(&mattr);

  pthread_mutexattr_settype(&mattr, /* type */);

  pthread_mutex_init(&mutex, &mattr);

  mutex_func();
  pthread_mutexattr_destroy(&mattr);

  return 0;
```

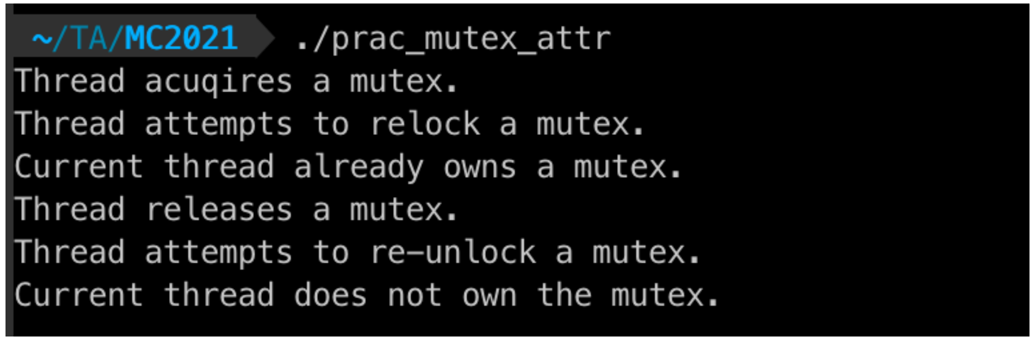
Example1 (cont.)

< PTHREAD_MUTEX_DEFAULT >



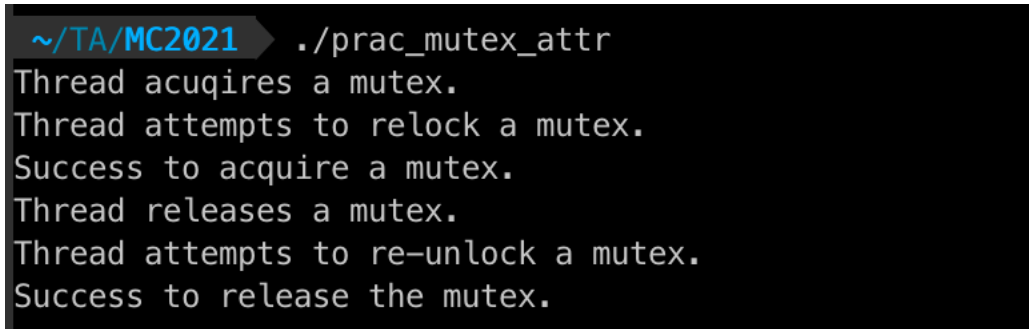
```
~/TA/MC2021  ./prac_mutex_attr
Thread acuqires a mutex.
Thread attempts to relock a mutex.
^C
```

< PTHREAD_MUTEX_ERRORCHECK >



```
~/TA/MC2021  ./prac_mutex_attr
Thread acuqires a mutex.
Thread attempts to relock a mutex.
Current thread already owns a mutex.
Thread releases a mutex.
Thread attempts to re-unlock a mutex.
Current thread does not own the mutex.
```

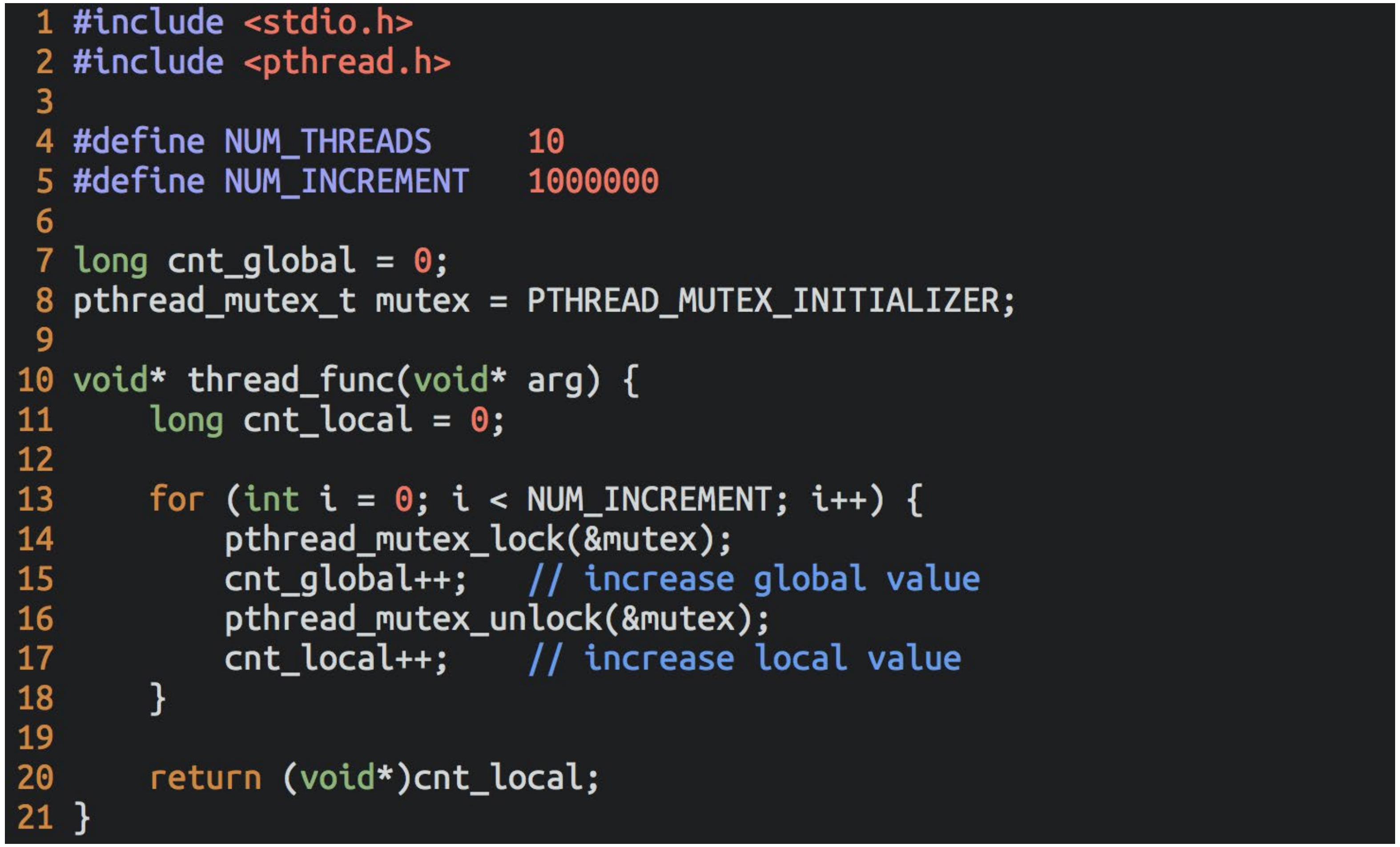
< PTHREAD_MUTEX_RECURSIVE >



```
~/TA/MC2021  ./prac_mutex_attr
Thread acuqires a mutex.
Thread attempts to relock a mutex.
Success to acquire a mutex.
Thread releases a mutex.
Thread attempts to re-unlock a mutex.
Success to release the mutex.
```

Example2

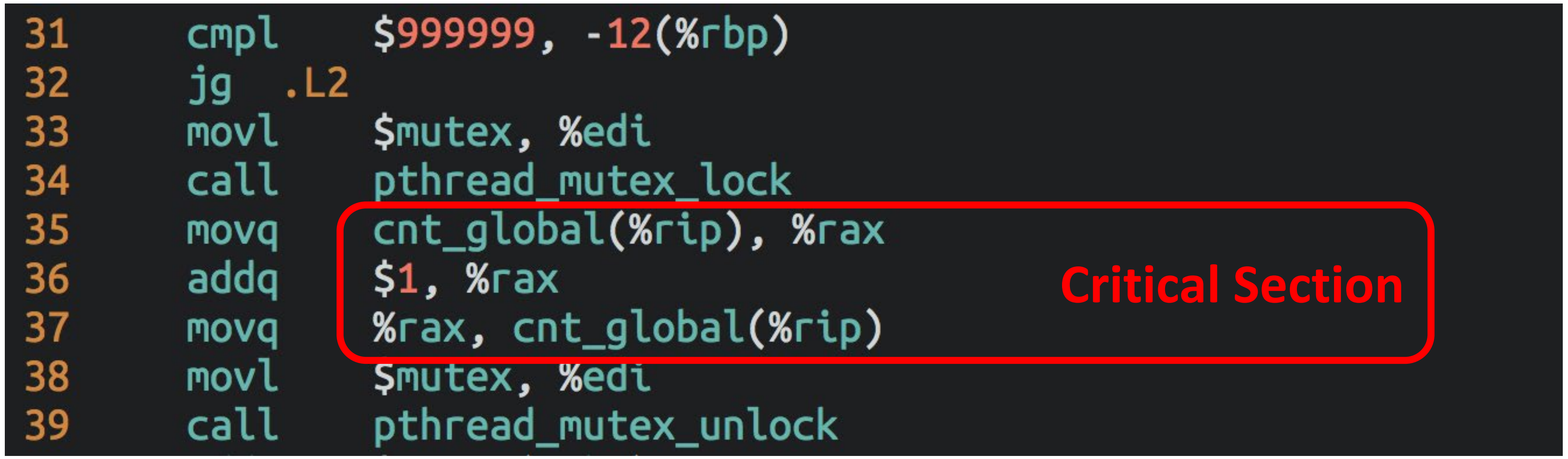
< prac_mutex.cpp >



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

#define NUM_THREADS     10
#define NUM_INCREMENT   1000000

long cnt_global = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void* thread_func(void* arg) {
    long cnt_local = 0;

    for (int i = 0; i < NUM_INCREMENT; i++) {
        pthread_mutex_lock(&mutex);
        cnt_global++;   // increase global value
        pthread_mutex_unlock(&mutex);
        cnt_local++;    // increase local value
    }

    return (void*)cnt_local;
}
```

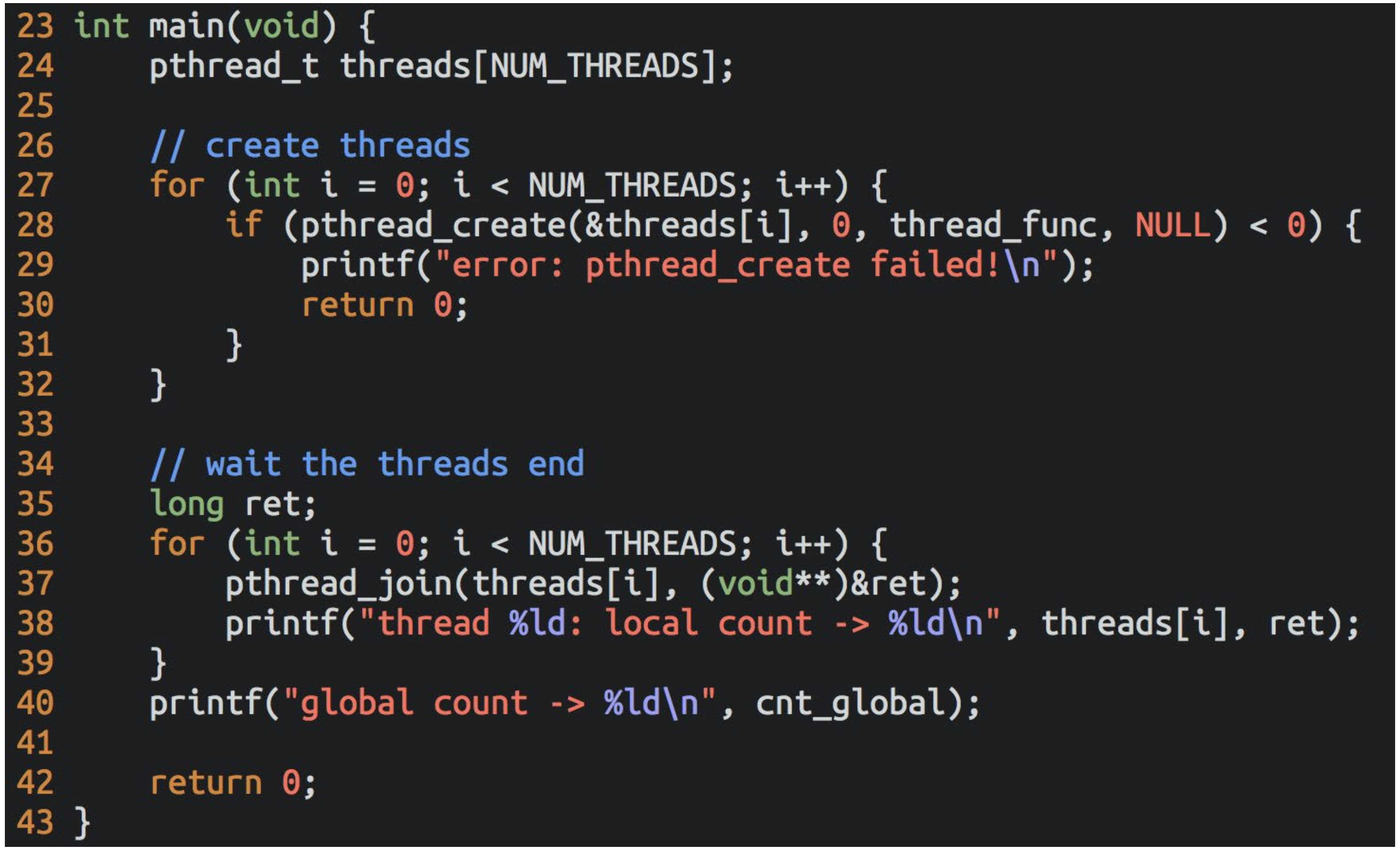
Example2 (continue..)

< Assembly instructions for *cnt_global++* in the C code >



```
        cmpl    $999999, -12(%rbp)
        jg  .L2
        movl    $mutex, %edi
        call    pthread_mutex_lock
        movq    cnt_global(%rip), %rax
        addq    $1, %rax
        movq    %rax, cnt_global(%rip)
        movl    $mutex, %edi
        call    pthread_mutex_unlock
```

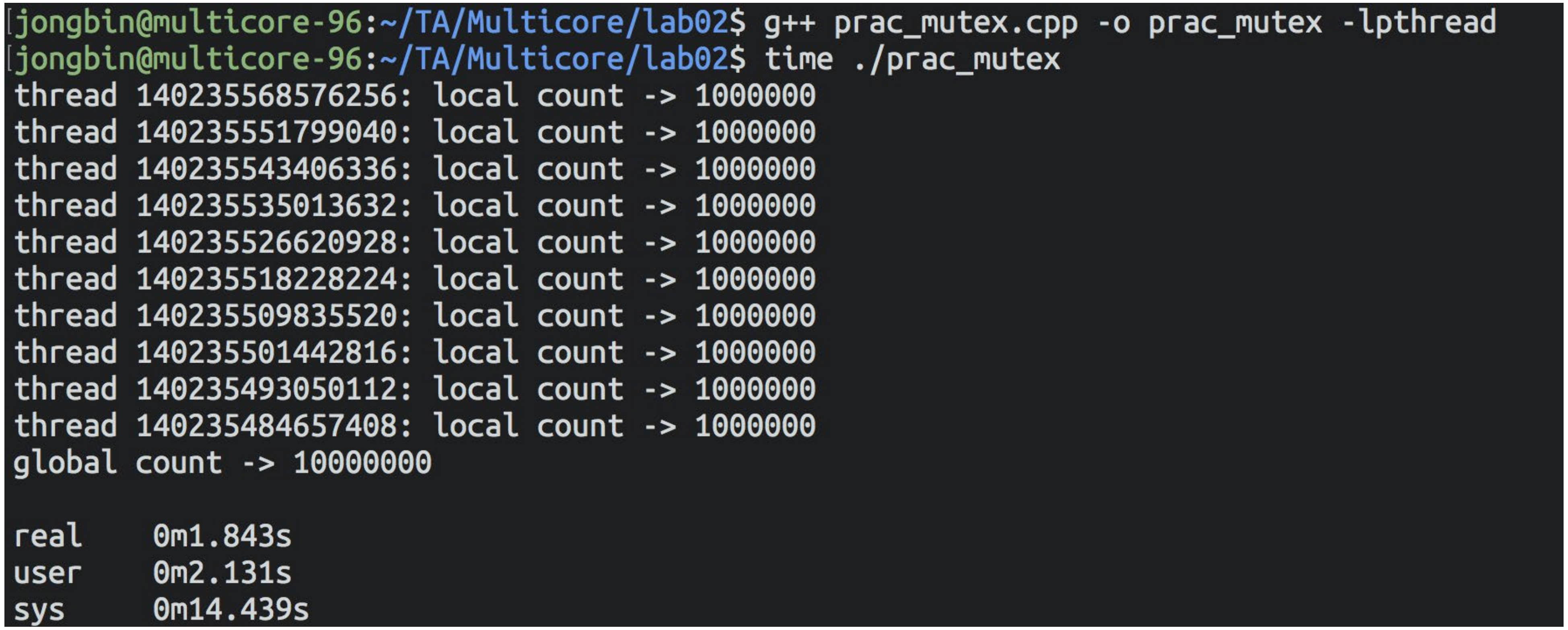
Example2 (continue..)



```c
int main(void) {
    pthread_t threads[NUM_THREADS];

    // create threads
    for (int i = 0; i < NUM_THREADS; i++) {
        if (pthread_create(&threads[i], 0, thread_func, NULL) < 0) {
            printf("error: pthread_create failed!\n");
            return 0;
        }
    }

    // wait the threads end
    long ret;
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], (void**)&ret);
        printf("thread %ld: local count -> %ld\n", threads[i], ret);
    }
    printf("global count -> %ld\n", cnt_global);

    return 0;
}
```

Example2 (continue..)

< Result >



```
[jongbin@multicore-96:~/TA/Multicore/lab02$ g++ prac_mutex.cpp -o prac_mutex -lpthread
[jongbin@multicore-96:~/TA/Multicore/lab02$ time ./prac_mutex
thread 140235568576256: local count -> 1000000
thread 140235551799040: local count -> 1000000
thread 140235543406336: local count -> 1000000
thread 140235535013632: local count -> 1000000
thread 140235526620928: local count -> 1000000
thread 140235518228224: local count -> 1000000
thread 140235509835520: local count -> 1000000
thread 140235501442816: local count -> 1000000
thread 140235493050112: local count -> 1000000
thread 140235484657408: local count -> 1000000
global count -> 10000000

real    0m1.843s
user    0m2.131s
sys     0m14.439s
```

Pthreads rwlock API

- pthread_rwlock_init
- pthread_rwlock_rdlock
- pthread_rwlock_wrlock
- pthread_rwlock_unlock

Pthread rwlock API – pthread_rwlock_rdlock

```
int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);
```

- Read lock the rwlock object. If the rwlock is already locked by a writer, the calling thread shall block until the rwlock becomes available.

```
@param[in]  rwlock         rwlock to be locked
@return                    0 if acquired. Error number related to the rwlockattr if failed.
```

Pthread rwlock API – pthread_rwlock_wrlock

```
int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);
```

- Write-lock the rwlock object. If the rwlock is already locked, the calling thread shall block until the mutex becomes available.

```
@param[in]  rwlock          rwlock to be locked
@return                     0 if acquired. Error number related to the mutexattr if failed.
```

Thank You