Mutex (MUTual EXclusion)

Multicore Programming

Introduction

- What is Mutex?
- Pthread Mutex API
- Example

What is Mutex?

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

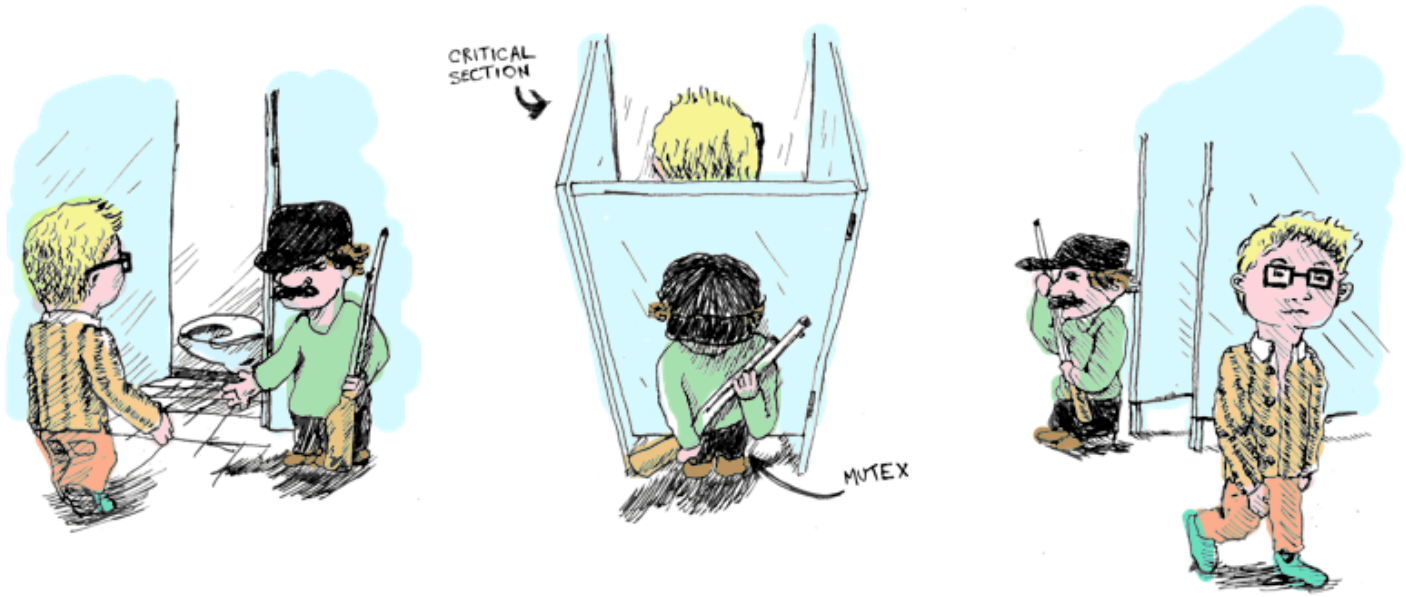

Photo reference: http://www.rudyhuyn.com/blog/2015/12/31/synchroniser-ses-agents-avec-lapplication/mutex/

Pthread Mutex API

- pthread_mutex_init
- pthread_mutex_lock
- pthread_mutex_trylock
- pthread_mutex_unlock
- more APIs, but not today

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                 Always 0
```

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.

Example

< prac_mutex.cpp >

```
 1 #include <stdio.h>
 2 #include <pthread.h>
 3
 4 #define NUM_THREAD      10
 5 #define NUM_INCREASE    1000000
 6
 7 int cnt_global = 0;
 8 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
 9
10 void* ThreadFunc(void* arg) {
11     long cnt_local = 0;
12
13     for (int i = 0; i < NUM_INCREASE; i++) {
14         pthread_mutex_lock(&mutex);
15         cnt_global++;   // increase global value
16         pthread_mutex_unlock(&mutex);
17         cnt_local++;    // increase local value
18     }
19
20     return (void*)cnt_local;
21 }
```

Example (continue..)

```
23 int main(void) {
24     pthread_t threads[NUM_THREAD];
25
26     // create threads
27     for (int i = 0; i < NUM_THREAD; i++) {
28         if (pthread_create(&threads[i], 0, ThreadFunc, NULL) < 0) {
29             printf("pthread_create error!\n");
30             return 0;
31         }
32     }
33
34     // wait threads end
35     long ret;
36     for (int i = 0; i < NUM_THREAD; i++) {
37         pthread_join(threads[i], (void**)&ret);
38         printf("thread %ld, local count: %ld\n", threads[i], ret);
39     }
40     printf("global count: %d\n", cnt_global);
41
42     return 0;
43 }
```

Example (continue..)

< Result >

```
mrbin2002@ubuntu:~/TA/Multicore/lab2$ time ./prac_mutex
thread 140293145884416, local count: 1000000
thread 140293133301504, local count: 1000000
thread 140293124908800, local count: 1000000
thread 140293116516096, local count: 1000000
thread 140293108123392, local count: 1000000
thread 140293099730688, local count: 1000000
thread 140293091337984, local count: 1000000
thread 140293082945280, local count: 1000000
thread 140293074552576, local count: 1000000
thread 140293066159872, local count: 1000000
global count: 10000000

real    0m1.134s
user    0m1.420s
sys     0m2.924s
```

Example (continue..)

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

```
32      movl    $mutex, %edi
33      call    pthread_mutex_lock
34      movl    cnt_global(%rip), %eax
35      addl    $1, %eax
36      movl    %eax, cnt_global(%rip)
37      movl    $mutex, %edi
38      call    pthread_mutex_unlock
39      addq    $1, -8(%rbp)
40      addl    $1, -12(%rbp)
```

Critical section

Thank You