NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
ISLAMABAD

OPERATING SYSTEMS LAB SPRING 2019

Lab Manual 08
MUTEX

1 WHAT IS A MUTEX?

1. Mutex is an abbreviation for “mutual exclusion”
2. It is a special variable which act as a mutual exclusion device to protect sections of code (hence the name mutex)
3. Mutex variables are one of the primary means of implementing thread synchronization

2 MUTEX VARIABLES

1. A mutex variable acts like a "lock" protecting access to a shared data resource (variables, section of code e.t.c).
2. It can be either in:
   (a) **Locked state**: a distinguished thread that holds or owns the mutex, or
   (b) **Unlocked state**: no thread holds the mutex
3. The basic concept of a mutex as used in Pthreads is that only one thread can lock (or own) a mutex variable at any given time..
4. If several threads try to lock a mutex only one thread will be successful. *The rest block at that call*.

5. No other thread can own that mutex until the owning thread unlocks that mutex. Threads must "take turns" accessing protected data.

3 MUTEX SYSTEM CALLS

```
pthread_mutex_init()
pthread_mutex_destroy()
pthread_mutex_lock()
pthread_mutex_trylock()
pthread_mutex_unlock()
```

4 MUTEX VARIABLES

A typical sequence in the use of a mutex is as follows:

1. Create and initialize a mutex variable
2. Several threads attempt to lock the mutex
3. Only one succeeds and that thread owns the mutex
4. The owner thread performs some set of actions
5. The owner unlocks the mutex
6. Another thread acquires the mutex and repeats the process
7. Finally the mutex is destroyed

THE IDEA:

```
lock the mutex
        critical section
unlock the mutex
```

5 USING MUTEX

1. **Declare** an object of type pthread_mutex_t.
2. **Initialize** the object by calling pthread_mutex_init().
3. Call pthread_mutex_lock() to **gain exclusive access** to the shared data object.
4. Call pthread_mutex_unlock() to **release the exclusive access** and allow another thread to use the shared data object.
5. **Get rid of the object** by calling pthread_mutex_destroy().

6 MUTEX SYSTEM CALLS EXPLAINED

1. **Initialization:** Mutex variables must be declared with type pthread_mutex_t and must be initialized before they can be used.

   (a) Statically it is declared as:
   pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;

   (b) Dynamically,with the pthread_mutex_init() routine. This methos allows to set mutex attributes, may be specified as NULL to accept defaults

   The mutex is initially unlocked.

2. **pthread_mutex_destroy()** should be used to free a mutex object which is no longer needed.

3. **pthread_mutex_lock()** routine is used by a thread to acquire a lock on the specified mutex variable.

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

4. **pthread_mutex_trylock()** will attempt to lock a mutex. If the mutex is already locked by another thread, this call will block the calling thread until the mutex is unlocked.

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

   If the mutex is already locked, the routine will return immediately with a "busy" error code

5. **pthread_mutex_unlock()** will unlock a mutex if called by the owning thread.

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

   Calling this routine is required after a thread has completed its use of protected data if other threads are to acquire the mutex for their work with the protected data.

   An error will be returned:

   (a) If the mutex was already unlocked.

   (b) If the mutex is owned by another thread.

7 THINGS TO AVOID WHILE USING MUTEX

1. No thread should attempt to lock or unlock a mutex that has not been initialized.

2. The thread that locks a mutex must be the thread that unlocks it.

3. No thread should have the mutex locked when you destroy the mutex.
4. Any mutex that is initialized should eventually be destroyed, but only after any thread that uses it has either terminated or is no longer interesting in using it.

8 EXAMPLES

EXAMPLE 01

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *functionC(void *);
//pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex1;
pthread_t thread1, thread2;
int counter = 0;

int main() {

        pthread_mutex_init(&mutex1,NULL);
        pthread_create(&thread1,NULL,&functionC,NULL);
        pthread_create(&thread2,NULL,&functionC,NULL);

        printf("Thread 1 ID: %ld \n",thread1);
        printf("Thread 2 ID: %ld \n",thread2);
        sleep(3);
       // pthread_join( thread1, NULL);
       // pthread_join( thread2, NULL);
        pthread_mutex_destroy(&mutex1);
        pthread_exit(NULL);
        exit(0);
}

void * functionC(void * p){

        pthread_mutex_lock( &mutex1 );
        counter++;

        printf("Thread %ld Counter value: %d\n",pthread_self(),counter);
        pthread_mutex_unlock( &mutex1 );
        pthread_exit(NULL);
}
```

EXAMPLE 02

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *functionA(void *);
void *functionB(void *);
//pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex1;
pthread_t thread1, thread2;
```

```
int counter = 0;
int main() {
        pthread_mutex_init(&mutex1,NULL);
        pthread_create(&thread1,NULL,&functionA,NULL);
        pthread_create(&thread2,NULL,&functionB,NULL);
        printf("Thread 1 ID: %ld \n",thread1);
        printf("Thread 2 ID: %ld \n",thread2);
        pthread_join( thread1, NULL);
        pthread_join( thread2, NULL);
        pthread_mutex_destroy(&mutex1);
        pthread_exit(NULL);
        exit(0);
}
void * functionA(void * p){
        int a;
        pthread_mutex_lock( &mutex1 );
        a=counter;a--;counter=a;
        printf("Thread 1 ID: %ld Counter value: %d\n",pthread_self(),counter);
        pthread_mutex_unlock( &mutex1 );
        pthread_exit(NULL);
}

void * functionB(void * p){
        int b;
        pthread_mutex_lock( &mutex1 );
        b=counter;b++;counter=b;
        printf("Thread 2 ID: %ld Counter value: %d\n",pthread_self(),counter);
        pthread_mutex_unlock( &mutex1 );
        pthread_exit(NULL);
}
```

EXAMPLE 03

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *functionA(void *);
void *functionB(void *);
pthread_mutex_t mutex1;
pthread_t thread1, thread2;
int counter = 0;
int main() {
        pthread_mutex_init(&mutex1,NULL);
        pthread_create(&thread1,NULL,&functionA,NULL);
        pthread_create(&thread2,NULL,&functionB,NULL);
        printf("Thread %ld \n",thread1);
        printf("Thread %ld \n",thread2);
        pthread_join( thread1, NULL);
        pthread_join( thread2, NULL);
        pthread_mutex_destroy(&mutex1);
        pthread_exit(NULL);
        exit(0);
}
void * functionA(void * p){
        int a;
```

```
        pthread_mutex_lock( &mutex1 );
        a=counter;a--;counter=a;
        sleep(1);
        printf("Thread %ld Counter value: %d\n",pthread_self(),counter);
        pthread_mutex_unlock( &mutex1 );
        pthread_exit(NULL);
}

void * functionB(void * p){
        int b;int mycount=0;
       while(pthread_mutex_trylock( &mutex1 )!=0){
            while(mycount<=800000){
              mycount++;
           }
            mycount=0;
            printf("Trying to own lock\n");
        }
        b=counter;b++;counter=b;
        printf("Thread %ld Counter value: %d\n",pthread_self(),counter);
        pthread_mutex_unlock( &mutex1 );
        pthread_exit(NULL);
}
```