CL-2006 Operating System	LAB – 08 Process Synchronization using Pthread libraries

Synchronization in Multithreaded Programs:

When multiple threads access and manipulate a shared resource (ex: a variable for instance), the access to the shared resource needs to be controlled through a lock mechanism so that only one thread is allowed access to the shared resource at any point of time while the other threads are waiting to gain access the shared resource. In Posix thread (pthread) this is enforced by using a synchronization primitive called **mutex lock**.

The **mutex lock** is the simplest and most primitive synchronization variable. It provides a single, absolute owner for the section of code (aka a **critical section**) that brackets between the calls
to **pthread_mutex_lock()** and **pthread_mutex_unlock()**. The first thread that locks the mutex gets ownership, and any subsequent attempts to lock it will fail, causing the calling thread to go to sleep. When the owner unlocks it, one of the sleepers will be awakened, made runnable, and given the chance to obtain ownership.

EXAMPLE: Exploring mutex with multi-threads sharing the same resource:

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

volatile int counter = 0;
pthread_mutex_t myMutex;
int argc, char *argv[]
void *mutex_testing(void *param)
{
        int i;
        for(i = 0; i < 5; i++) {
                pthread_mutex_lock(&myMutex;);
                counter++;
                printf("thread %d counter = %d\n", (int)param,  counter);
                pthread_mutex_unlock(&myMutex;);
        }
}

int main()
{
        int one = 1, two = 2, three = 3;
        pthread_t thread1, thread2, thread3;
        pthread_mutex_init(&myMutex;,0);
        pthread_create(&thread1;, 0, mutex_testing, (void*)one);
        pthread_create(&thread2;, 0, mutex_testing, (void*)two);
        pthread_create(&thread3;, 0, mutex_testing, (void*)three);
        pthread_join(thread1, 0);
        pthread_join(thread2, 0);
        pthread_join(thread3, 0);
        pthread_mutex_destroy(&myMutex;);
        return 0;
}
```

Mutex Attributes:

Though mutexes, by default, are private to a process, they can be shared between multiple processes. To create a mutex that can be shared between processes, we need to set up the attributes for **pthread_mutex_init():**

```c
#include <pthread.h>

int main()
{
        pthread_mutex_t myMutex;
        pthread_mutexattr_t myMutexAttr;
        pthread_mutexattr_init(&myMutexAttr;);
        pthread_mutexattr_setpshared(&myMutexAttr;, PTHREAD_PROCESS_SHARED);

        pthread_mutex_init(&myMutex;, &myMutexAttr;);
        //...

        pthread_mutexattr_destroy(&myMutexAttr;);
        pthread_mutex_destroy(&myMutex;);
        return 0;
}
```

pthread_mutexattr_setpshared() with a pointer to the attribute structure and the value **PTHREAD_PROCESS_SHARED** sets the attributes to cause a shared mutex to be created.

Mutexes are not shared between processes by default. Calling **pthread_mutexattr_setpshared()** with the value **PTHREAD_PROCESS_PRIVATE** restores the attribute to the default.

These attributes are passed into the call to **pthread_mutexattr_init()** to set the attributes of the initialized mutex. Once the attributes have been used, they can be disposed of by a call to **pthread_mutexattr_destroy().**