# PTHREAD MUTEX

**Mutex (mutual exclusion)**
When a thread need exclusive access sections of code, it can use mutex.
Mutex can prevent other thread from executing a sections of code.

Lets start with an example.

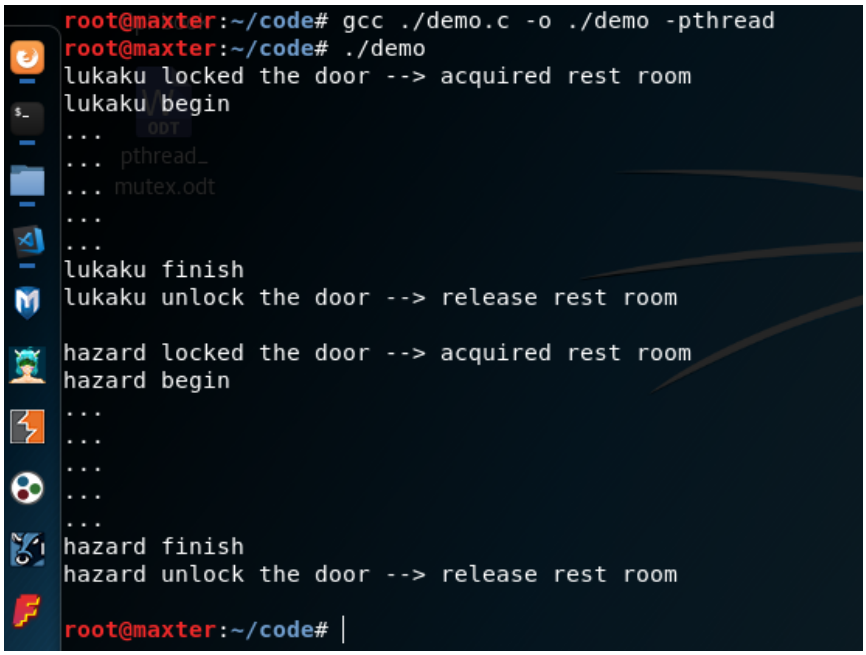
| Lukaku and Hazard need to enter a rest room.<br><br>However, the rest room capability is only one man at a time. | Thread_1 and Thread_2 need to execute a section of code.<br>However, this code is designed to be executed by only one thread at a time. |
|---|---|
| Lukaku is acquiring rest room.<br>The door is locked.<br><br>One minutes later.<br><br>Hazard need to enter the rest room.<br>However, the door is locked. Hazard must wait.<br><br>Lukaku finish his job.<br>Lukaku open the door and get out.<br><br>Now, Hazard can enter the rest room. | Thread_1 is executing the code.<br>The mutex is locked.<br><br>One minutes later.<br><br>Thread_2 need to execute the code, too.<br>However, mutex is locked, Thread_2 must wait<br><br>Thread_1 finish execution.<br>Thread_1 unlock the mutex<br><br>Now, Thread_2 can execute the code. |

Code

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

static pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;

void* enter_rest_room(void* arg){
    pthread_mutex_lock(&mtx); //lock the door

    printf("%s locked the door --> acquired rest room\n", (char*)arg);

    printf("%s begin\n", (char*)arg);

    //use the rest room in five minutes
    for(int i = 0; i < 5; i++) {
        printf("...\n");
        sleep(1);
    }

    printf("%s finish\n", (char*)arg);

    pthread_mutex_unlock(&mtx); //unlock the door

    printf("%s unlock the door --> release rest room\n\n", (char*)arg);
}

int main() {
    pthread_t lukaku;
    pthread_t hazard;

    //lukaku enter the rest room
    pthread_create(&lukaku, NULL, enter_rest_room, "lukaku");

    sleep(1); //one minute later

    //harard need to enter the rest room
    pthread_create(&hazard, NULL, enter_rest_room, "hazard");
```

```
    sleep(11);
    return 0;
}
```

Result




In above example, the mutex works like a door of the rest room.
If the door is locked, later person must wait.
Until the door is unlocked, he can acquire the rest room.

If the mutex is locked, later thead which need to execute the exclusive code must wait.
Util the mutex is unlocked, it can acquire the mutex and execute the code.

**Mutex Features**
*Atomicity*
Two thread can not lock the same mutex at the same time.

*Singularity*
If a thread acquire the mutex, no other thread will able to lock the mutex.

*Non-Busy Wait*
If a thread is waiting for a mutex. It will be suspended and not consume any CPU resouces.

**Initialize Mutex**
A mutex variable is represented by the *pthread_mutex_t* data type.
We must first initiallize it, before using it.

```
int pthread_mutex_init(pthread_mutex_t *mutex, pthread_mutexattr_t attr)
```

The mutex argument identifies the mutex to be initialized.
The attr argumen define attributes for the mutex. If attr is NULL, the mutex will be initialized with default attributes.

Attempting to initalize an already initalized mutex will lead to undefined behavior.

In case mutex is statically allocated, we should use macro *PTHREAD_MUTEX_INITIALIZER* to initialize it.
Among below case, we can use *pthread_mutex_init()* to initialize
- The mutex is dynamically allocated on the heap
- The mutex is an automatically variable allocated on the stack
- We want to custome attributes of static mutex.

**Destroy Mutex**
When a mutex is no long required, it should be destroyed by using *pthread_mutex_destroy()*

```
int pthread_mutex_destroy(pthread_mutex_t *mutex)
```

It will be safe to destroy an initialized mutex that is locked.
Atempting to destroy a locked mutex will lead to undefined behavior.

It is not neccessary to call *pthread_mutex_destroy* on static mutex. Static variable will be destroyed automatically when the process is terminated.

**Lock Mutex**

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

If the mutex is unlocked, the calling thread will acquire the mutex.
If the mutex is locked by another thread, the calling thread will block until the mutex becomes avaible.
If a thread attempt to lock a mutex that it already owns, deadlock will occur.

**Unlock Mutex**

```
int pthead_mutex_unlock(pthread_mutex_t *mutex)
```

The *pthead_mutex_unlock()* function unlock the mutex previously locked by the calling thread.
If other threads are currently waiting the mutex, one of these acquire the muex and resume its execution.

Attempting to unlock a mutex that is not currenly locked, deadlock or undefined behavior error will occur.
Attempting to unlock a mutex that is locked by another thread, deadlock or undefined behavior error will occur.

Sample Code
Initialize static mutex

```
#include <pthread.h>

static pthread_mutex_t = PTHREAD_MUTEX_INITIALIZER; //initialize static mutex
```

Inititalize dynamic mutex

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

```
void foo(){
    pthread_mutex_t mutex;
    pthread_mutex_init(&mutex, NULL);//initialize dynamic mutex

    pthread_mutex_destroy(&mutex);//destroy dynamic mutex
}

void bar(){
    pthread_mutex_t* p_mutex =
(pthread_mutex_t*)malloc(sizeof(pthread_mutex_t));
    pthread_mutex_init(p_mutex, NULL);//initialize dynamic mutex

    pthread_mutex_destroy(p_mutex);//destroy dynamic mutex

    free(p_mutex);
}

int main() {
    foo();
    bar();

    return 0;
}
```

**//continue**
**- Data race**
**- Resolve data race**

**- Deadlock**
**- Resolve deadlock**