# 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.

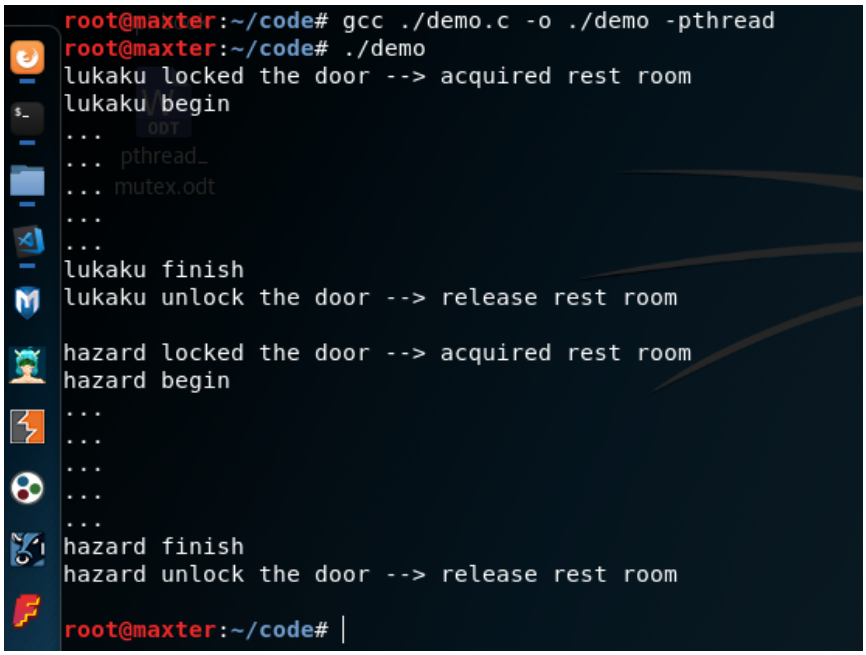
| | |
|---|---|
| Lukaku and Hazard need to enter a rest room.<br><br>However, the rest room capability is only one man at a time. | Thread_1 and Thread_2 need to execute a section of code.<br>However, this code is designed to be executed by only one thread at a time. |
| Lukaku is acquiring rest room.<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.

**//continue**
**- Mutex API**
    **- Init (static, dynamic)**
    **- Lock, Unlock**
    **- Destroy**

**- Data race**
**- Resolve data race**

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