

Напишем программу на языке C с преднамеренной ошибкой - взаимной блокировкой ресурсов из-за не освобождения мьютекса в потоке.

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

pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex2 = PTHREAD_MUTEX_INITIALIZER;

void* thread1(void* arg) {
    pthread_mutex_lock(&mutex1);
    printf("Thread 1 locked mutex1.\n");

    // Здесь поток 1 забывает освободить mutex2
    pthread_mutex_lock(&mutex2);
    printf("Thread 1 locked mutex2.\n");

    // Делаем что-то с общим ресурсом...

    pthread_mutex_unlock(&mutex1);
    printf("Thread 1 unlocked mutex1.\n");

    pthread_mutex_unlock(&mutex2);
    printf("Thread 1 unlocked mutex2.\n");
}

void* thread2(void* arg) {
    pthread_mutex_lock(&mutex2);
    printf("Thread 2 locked mutex2.\n");

    // Здесь поток 2 ждет, пока поток 1 освободит mutex1
    pthread_mutex_lock(&mutex1);
    printf("Thread 2 locked mutex1.\n");

    // Делаем что-то с общим ресурсом...

    pthread_mutex_unlock(&mutex1);
    printf("Thread 2 unlocked mutex1.\n");

    pthread_mutex_unlock(&mutex2);
    printf("Thread 2 unlocked mutex2.\n");
}

int main() {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, thread1, NULL);
```

```

pthread_create(&t2, NULL, thread2, NULL);
pthread_join(t1, NULL);
pthread_join(t2, NULL);
return 0;
}

```

В программе мы создаем два потока t1 и t2, каждый из которых должен заблокировать свой мьютекс, а затем заблокировать другой мьютекс. Однако, из-за того, что каждый поток блокирует только свой мьютекс и не освобождает его, другой поток не может заблокировать нужный мьютекс, что приводит к взаимной блокировке и зависанию программы.

Соберем программу с различными опциями gcc выдающими расширенные предупреждения.

```

parallels@ubuntu-linux-22-04-desktop: ~/Desktop
parallels@ubuntu-linux-22-04-desktop:~/Desktop$ gcc -Wshadow lock_mutex.c -o lock_mutex
parallels@ubuntu-linux-22-04-desktop:~/Desktop$ gcc -Wconversion lock_mutex.c -o lock_mutex
parallels@ubuntu-linux-22-04-desktop:~/Desktop$ gcc -Wformat lock_mutex.c -o lock_mutex
parallels@ubuntu-linux-22-04-desktop:~/Desktop$ gcc -Waddress lock_mutex.c -o lock_mutex
parallels@ubuntu-linux-22-04-desktop:~/Desktop$ gcc -Wall lock_mutex.c -o lock_mutex
lock_mutex.c: In function 'thread1':
lock_mutex.c:15:1: warning: control reaches end of non-void function [-Wreturn-type]
   15 | }
      | ^
lock_mutex.c: In function 'thread2':
lock_mutex.c:25:1: warning: control reaches end of non-void function [-Wreturn-type]
   25 | }
      | ^
parallels@ubuntu-linux-22-04-desktop:~/Desktop$ gcc -Wextra lock_mutex.c -o lock_mutex
parallels@ubuntu-linux-22-04-desktop:~/Desktop$

```

Воспользуемся санитайзером gcc -sanitize. Используем опцию thread.

```

parallels@ubuntu-linux-22-04-desktop: ~/Desktop
parallels@ubuntu-linux-22-04-desktop:~/Desktop$ gcc -fsanitize=thread lock_mutex.c -o lock_mutex
parallels@ubuntu-linux-22-04-desktop:~/Desktop$ ./lock_mutex
Thread 1 locked mutex1.
Thread 1 locked mutex2.
Thread 1 unlocked mutex1.
Thread 1 unlocked mutex2.
Thread 2 locked mutex2.
=====
WARNING: ThreadSanitizer: lock-order-inversion (potential deadlock) (pid=81181)
  Cycle in lock order graph: M0 (0xaaaaaaaa2018) => M1 (0xaaaaaaaa2048) => M0

  Mutex M1 acquired here while holding mutex M0 in thread T1:
   #0 pthread_mutex_lock ../../../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors.inc:4240 (libtsan.so.0+0x56880)
   #1 thread1 <null> (lock_mutex+0xb54)

  Hint: use TSAN_OPTIONS=second_deadlock_stack=1 to get more informative warning message

  Mutex M0 acquired here while holding mutex M1 in thread T2:
   #0 pthread_mutex_lock ../../../../src/libsanitizer/sanitizer_common/sanitizer_common_interceptors.inc:4240 (libtsan.so.0+0x56880)
   #1 thread2 <null> (lock_mutex+0xbe4)

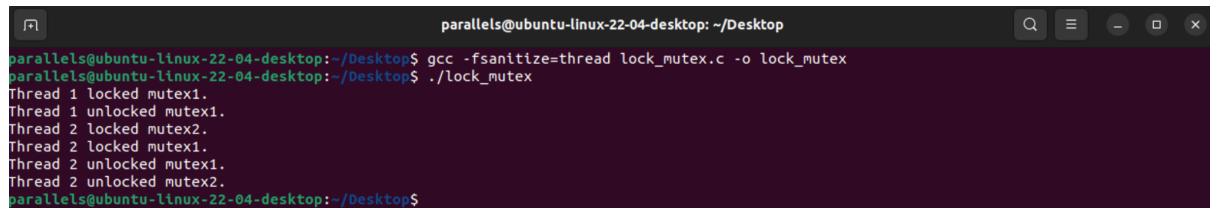
  Thread T1 (tid=81183, finished) created by main thread at:
   #0 pthread_create ../../../../src/libsanitizer/tsan/tsan_interceptors_posix.cpp:969 (libtsan.so.0+0x63bb0)
   #1 main <null> (lock_mutex+0xc80)

  Thread T2 (tid=81184, running) created by main thread at:
   #0 pthread_create ../../../../src/libsanitizer/tsan/tsan_interceptors_posix.cpp:969 (libtsan.so.0+0x63bb0)
   #1 main <null> (lock_mutex+0xc9c)

SUMMARY: ThreadSanitizer: lock-order-inversion (potential deadlock) (/home/parallels/Desktop/lock_mutex+0xb54) in thread1
=====
Thread 2 locked mutex1.
Thread 2 unlocked mutex1.
Thread 2 unlocked mutex2.
ThreadSanitizer: reported 1 warnings
parallels@ubuntu-linux-22-04-desktop:~/Desktop$

```

Исправим ошибку на основе информации, полученной от санитайзера. Чтобы исправить эту проблему, поток 1 должен правильно освободить мьютексы после завершения работы с общим ресурсом.



```
parallels@ubuntu-linux-22-04-desktop: ~/Desktop
parallels@ubuntu-linux-22-04-desktop:~/Desktop$ gcc -fsanitize=thread lock_mutex.c -o lock_mutex
parallels@ubuntu-linux-22-04-desktop:~/Desktop$ ./lock_mutex
Thread 1 locked mutex1.
Thread 1 unlocked mutex1.
Thread 2 locked mutex2.
Thread 2 locked mutex1.
Thread 2 unlocked mutex1.
Thread 2 unlocked mutex2.
parallels@ubuntu-linux-22-04-desktop:~/Desktop$
```

Предоставим результат.

В отличие от других опций компилятора gcc, которые показывают расширенные предупреждения о возможных проблемах в программе, санитайзер gcc показывает более подробную информацию об ошибках, указывая конкретные функции и переменные, а также места, где содержатся ошибки. Используя информацию, полученную от санитайзера, в программе был изменен порядок захвата мьютекса в потоке, чтобы исключить возможность взаимной блокировки ресурсов.