Laborator 2 – Rezolvare

**Ex 1**. Deadlock:

Comenzi: ***cd /mnt/c/Users/andrei2timo/Desktop***

***gcc -pthread -o deadlock deadlock.c && ./deadlock***

*Run the compiled executable, does it hang?* → Da; se blochează.

*Is the deadlock perfect (meaning does it happen for sure all the time)?* → Nu, nu este. Avem această linie de cod: ***pthread\_mutex\_lock(&mtx[1-\*param]);*** . Pentru thread1, acesta va bloca mutexul la indecșii 0, respectiv 1; iar pentru thread2, la indecșii 1, respectiv 0. În teorie, în funcție de ordinea prin care thread-urile accesează funcția și blochează mutexul, este posibil ca acesta să nu se blocheze mereu: dacă un thread accesează resursele, le blochează și le eliberează, apoi vine celălalt thread și repetă acțiunea, totul ar trebui să funcționeze corect și să nu se producă deadlock.

**Ex 2**. *Create a perfect deadlock by adding a barrier after each thread takes the first resource*.

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

pthread\_mutex\_t mtx[2];

pthread\_barrier\_t barrier;

void \*ThrFunc(void \*p)

{

int \*param = (int\*)p;

pthread\_mutex\_lock(&mtx[\*param]); // here the thread takes the first resource

printf("Thread %d before lock\n", \*param);

pthread\_barrier\_wait(&barrier); // here the thread waits for the other thread to (take the first resource and) reach the barrier

pthread\_mutex\_lock(&mtx[1-\*param]);

return 0;

}

int main()

{

pthread\_t thr1;

pthread\_t thr2;

int i1 = 0, i2 = 1;

pthread\_barrier\_init(&barrier, NULL, 3); // initializing synchronization barrier with a count of the total number of threads that must be synchronized to the barrier before the threads may carry on (th1 + th2 + main th)

pthread\_mutex\_init(&mtx[0], NULL);

pthread\_mutex\_init(&mtx[1], NULL);

pthread\_create(&thr1, NULL, ThrFunc, &i1);

pthread\_create(&thr2, NULL, ThrFunc, &i2);

pthread\_join(thr1, NULL);

pthread\_join(thr2, NULL);

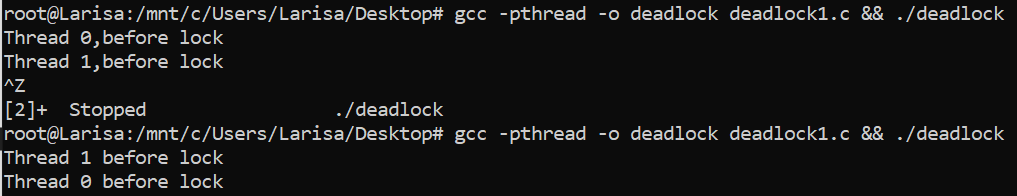
pthread\_mutex\_destroy(&mtx[0]);

pthread\_mutex\_destroy(&mtx[1]);

pthread\_barrier\_destroy(&barrier); // destroying barrier

return 0;

}



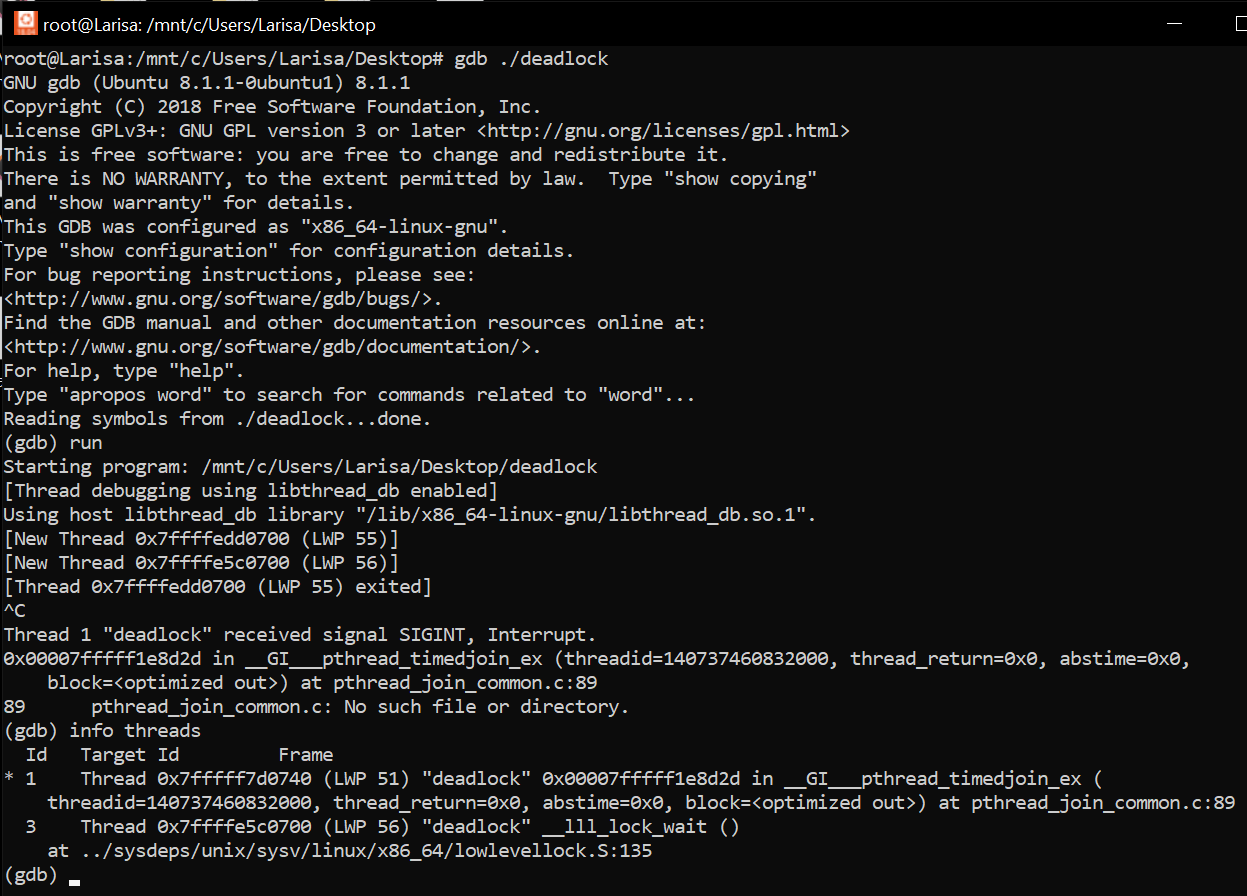
**Ex 3**. Comenzi: ***cd /mnt/c/Users/Larisa/Desktop***

***gcc -pthread -O0 -g -o deadlock deadlock.c***

***gdb ./deadlock***

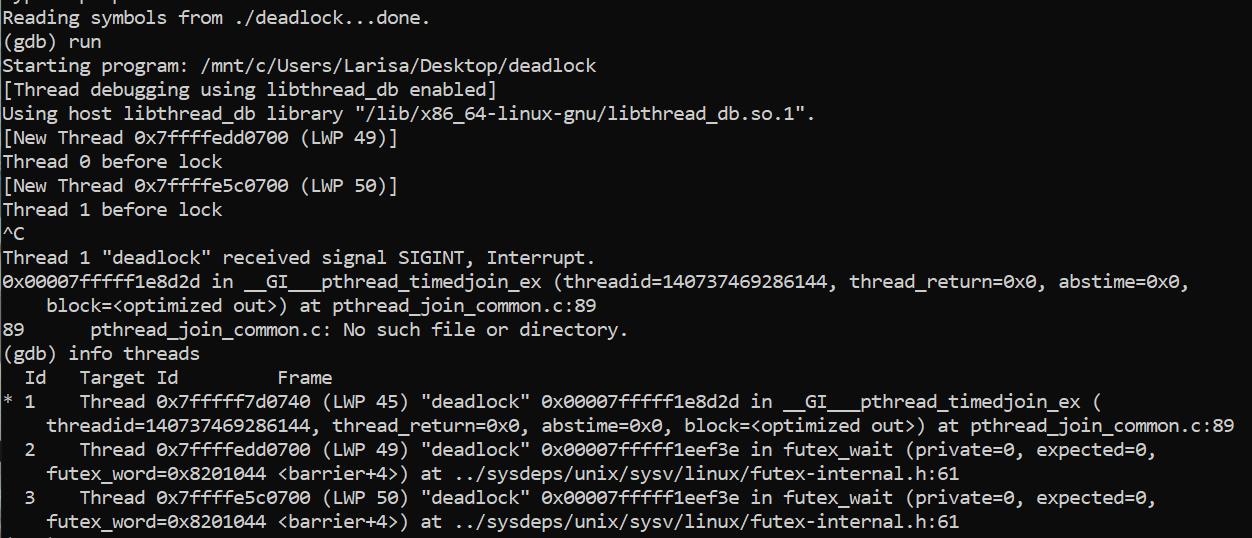
***run*** + ***CTRL+C***

***info threads***

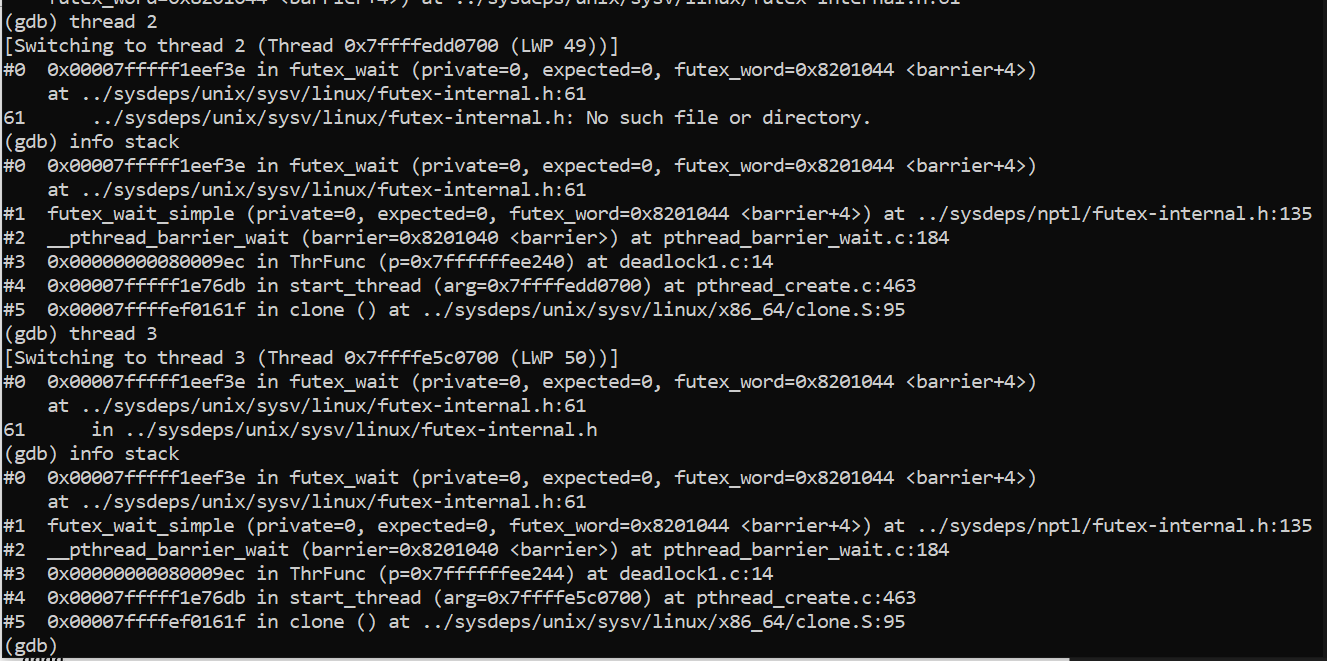


*How many threads does it display? Why?* → Avem doar 2 thread-uri afișate. Thread-ul ox7ffffedd0700 (LWP 55) a ieșit din execuție (a obținut ambele resurse și a terminat), așa că a rămas doar thread-ul principal (așteaptă ca ambele thread-uri să termine) și thread-ul LWP 56 (așteaptă o resursă).

Repetăm pașii pentru programul modificat:



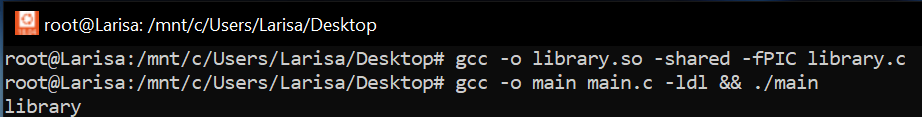
*Does info threads shows you now 3 threads? Why?* → Acum niciun thread nu a ieșit din execuție. Thread-ul principal le așteaptă pe cele 2 din program. Folosind bariera, ne asigurăm că fiecare thread are o resursă înainte ca ambele să își poată continua execuția, moment în care se crează deadlock perfect și programul se blochează (niciun thread nu termină execuția).



**Ex 3**. Comenzi: ***cd /mnt/c/Users/Larisa/Desktop***

***gcc -o library.so -shared -fPIC library.c***

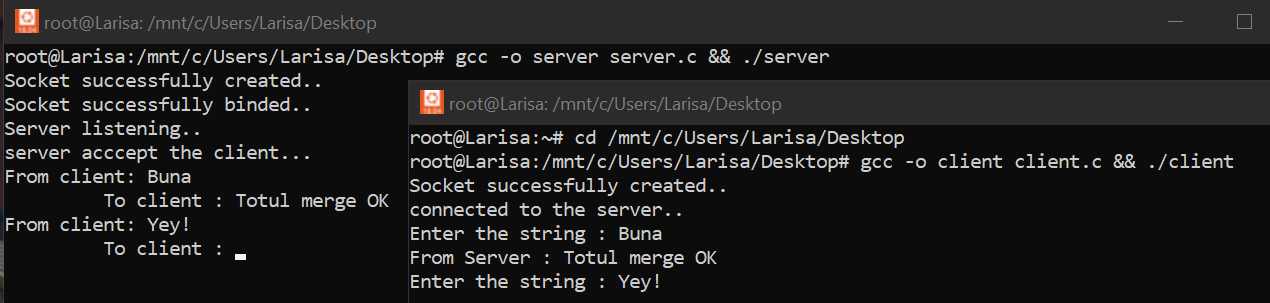
***gcc -o main main.c -ldl && ./main***



**Ex 4**. Comenzi: ***cd /mnt/c/Users/Larisa/Desktop***

***gcc -o server server.c && ./server***

***gcc -o client client.c && ./client***



*Now replace the content of function MyFunction from library* ***library.so*** *with a code which will connect to a server and will receive commands from that server which will be run using function* ***system****:* [*https://man7.org/linux/man-pages/man3/system.3.html*](https://man7.org/linux/man-pages/man3/system.3.html)

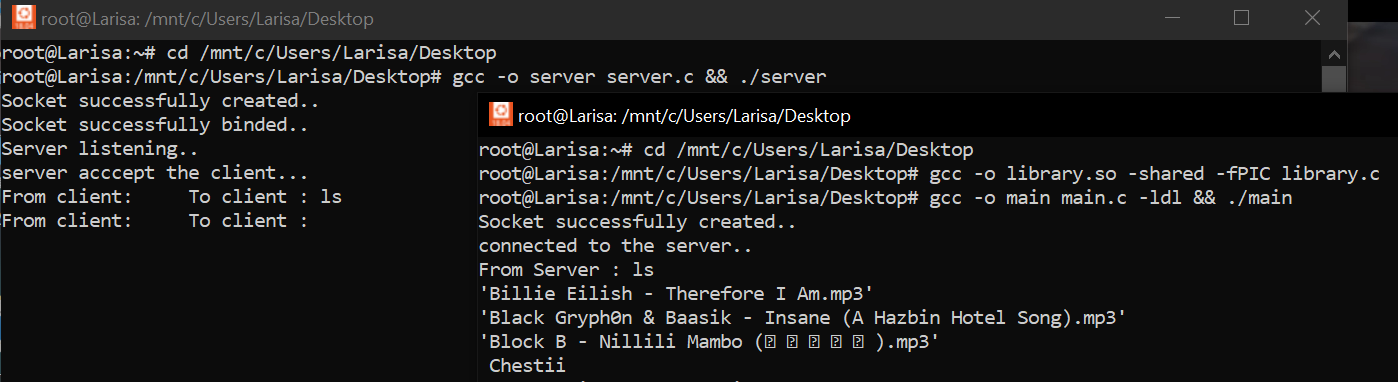
În *library.so* am copiat codul din *client.c* cu o singură linie de cod adăugată:



Comenzi rulate: ***gcc -o server server.c && ./server***

***gcc -o library.so -shared -fPIC library.c***

***gcc -o main main.c -ldl && ./main***

******