1.

```c
#include<stdio.h>


int main()
{
   printf("Hello World!\n");
   return 0;
}
```

```
alex@alex-HP-Pavilion-Laptop-15-cs3xxx:~/Workspace/facultate/Master/OS: Design & Security$ objdump -h hello-world

hello-world:     file format elf64-x86-64

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .interp       0000001c  0000000000000318  0000000000000318  00000318  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .note.gnu.property 00000020  0000000000000338  0000000000000338  00000338  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .note.gnu.build-id 00000024  0000000000000358  0000000000000358  00000358  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .note.ABI-tag 00000020  000000000000037c  000000000000037c  0000037c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .gnu.hash     00000024  00000000000003a0  00000000000003a0  000003a0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .dynsym       000000c0  00000000000003c8  00000000000003c8  000003c8  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .dynstr       00000089  0000000000000488  0000000000000488  00000488  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  7 .gnu.version  00000010  0000000000000512  0000000000000512  00000512  2**1
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  8 .gnu.version_r 00000020  0000000000000528  0000000000000528  00000528  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  9 .rela.dyn     000000c0  0000000000000548  0000000000000548  00000548  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 10 .rela.plt     00000030  0000000000000608  0000000000000608  00000608  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 11 .init         0000001b  0000000000001000  0000000000001000  00001000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 12 .plt          00000030  0000000000001020  0000000000001020  00001020  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 13 .plt.got      00000010  0000000000001050  0000000000001050  00001050  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 14 .plt.sec      00000020  0000000000001060  0000000000001060  00001060  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 15 .text         000001c5  0000000000001080  0000000000001080  00001080  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 16 .fini         0000000d  0000000000001248  0000000000001248  00001248  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 17 .rodata       0000001a  0000000000002000  0000000000002000  00002000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 18 .eh_frame_hdr 00000044  000000000000201c  000000000000201c  0000201c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 19 .eh_frame     00000108  0000000000002060  0000000000002060  00002060  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 20 .init_array   00000008  0000000000003db0  0000000000003db0  00002db0  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 21 .fini_array   00000008  0000000000003db8  0000000000003db8  00002db8  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 22 .dynamic      000001f0  0000000000003dc0  0000000000003dc0  00002dc0  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 23 .got          00000050  0000000000003fb0  0000000000003fb0  00002fb0  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 24 .data         00000010  0000000000004000  0000000000004000  00003000  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 25 .bss          00000010  0000000000004010  0000000000004010  00003010  2**2
                  ALLOC
 26 .comment      0000002a  0000000000000000  0000000000000000  00003010  2**0
                  CONTENTS, READONLY
```

Din zonele de memorie descrise in laborator se pot identifica urmatoarele:

- Zona de cod (.text) pe linia numarul 15
- Zona de date constante (.rodata) pe linia numarul 17
- Zona de date globale sau statice intializate (.data) pe linia numarul 24
- Zona de date globale sau statice neinitializate (.bss) pe linia numarul 25

Zonele de stack si heap nu sunt regasite in output-ul acestei comenzi deoarece acestea sunt create dinamic in timpul rularii programului.

Coloana “LMA” reprezinta adresa logica unde incepe zona respectiva de memorie. Pentru a obtine adresa unde se termina o zona de memorie vom aduna la adresa de inceput valoarea de pe coloana “Size”, ce reprezinta dimensiunea zonei de memorie. Mai precis, pentru zonele de memorie mai sus mentionate vom obtine urmatoarele valori:

- (.text) adresa de inceput : 0000000000001080; adresa de sfarsit: 0000000000001245
- (.rodata) adresa de inceput: 0000000000002000; adresa de sfarsit: 000000000000201a
- (.data) adresa de inceput: 0000000000004000; adresa de sfarsit: 0000000000004010
- (.bss) adresa de inceput: 0000000000004010; adresa de sfarsit: 0000000000004020

2.

```c
#include<stdio.h>
#include<stdlib.h>

int main()
{
   int* x;
   x = (int*) malloc(sizeof(int));
   *x = 0;
   while (*x>=0) {
        (*x)++;
        (*x)--;
   }
   printf("Hello World!\n");
   return 0;
}
```

```
alex@alex-HP-Pavilion-Laptop-15-cs3xxx:~/Workspace/facultate/Master/OS: Design & Security$ cat /proc/13836/maps
55d354eaf000-55d354eb0000 r--p 00000000 103:02 14160448                  /home/alex/Workspace/facultate/Master/OS: Design & Security/loop
55d354eb0000-55d354eb1000 r-xp 00001000 103:02 14160448                  /home/alex/Workspace/facultate/Master/OS: Design & Security/loop
55d354eb1000-55d354eb2000 r--p 00002000 103:02 14160448                  /home/alex/Workspace/facultate/Master/OS: Design & Security/loop
55d354eb2000-55d354eb3000 r--p 00002000 103:02 14160448                  /home/alex/Workspace/facultate/Master/OS: Design & Security/loop
55d354eb3000-55d354eb4000 rw-p 00003000 103:02 14160448                  /home/alex/Workspace/facultate/Master/OS: Design & Security/loop
55d3568b8000-55d3568d9000 rw-p 00000000 00:00 0                          [heap]
7f1805f31000-7f1805f56000 r--p 00000000 103:02 50465433                  /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1805f56000-7f18060ce000 r-xp 00025000 103:02 50465433                  /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f18060ce000-7f1806118000 r--p 0019d000 103:02 50465433                  /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1806118000-7f1806119000 ---p 001e7000 103:02 50465433                  /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1806119000-7f180611c000 r--p 001e7000 103:02 50465433                  /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f180611c000-7f180611f000 rw-p 001ea000 103:02 50465433                  /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f180611f000-7f1806125000 rw-p 00000000 00:00 0
7f1806138000-7f1806139000 r--p 00000000 103:02 50465429                  /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1806139000-7f180615c000 r-xp 00001000 103:02 50465429                  /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f180615c000-7f1806164000 r--p 00024000 103:02 50465429                  /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1806165000-7f1806166000 r--p 0002c000 103:02 50465429                  /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1806166000-7f1806167000 rw-p 0002d000 103:02 50465429                  /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1806167000-7f1806168000 rw-p 00000000 00:00 0
7fff5d333000-7fff5d354000 rw-p 00000000 00:00 0                          [stack]
7fff5d39f000-7fff5d3a3000 r--p 00000000 00:00 0                          [vvar]
7fff5d3a3000-7fff5d3a5000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

Zona de stack espe mapata la adresa 7f1806168000, iar zona de heap la adresa 55d3568d9000.

Celelalte zone de memorie mapate apartin programului rulat (“loop”) sau altor librarii ce sunt utilizate de catre program (“libc”, “ld”).

```
alex@alex-HP-Pavilion-Laptop-15-cs3xxx:~/Workspace/facultate/Master/OS: Design & Security$ ldd loop
        linux-vdso.so.1 (0x00007fffb75a3000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f75e701e000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f75e722a000)
```

Folosind comanda ldd, se poate observa de ce librarii depinde executabilul creat, iar legatura cu output-ul comenzii precedente este ca fiecare librarie are mapata in memorie mai multe zone de memorie.

1.

```c
#include <stdio.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <signal.h>

// Function used to handle the SIGSEGV signal
void sigsegv_handler(int sig, siginfo_t* si, void* unused) {
   void* addr = si->si_addr;
   // Give write access to that memory
   int val = mprotect(si->si_addr, 20, PROT_WRITE | PROT_READ);
}

int main()
```

```
{
    int fd;
    char* addr;
    char* filename = "file.in";
    struct sigaction sa;

    // Opening the file required by mmap
    fd = open(filename, O_RDWR);
    if (fd == -1) {
        printf("Error when opening the file.\n");
        return 1;
    }

    // Map the file into the process
    addr = mmap(NULL, 20, PROT_READ, MAP_SHARED, fd, 0);
    if (addr == MAP_FAILED) {
        printf("mmap is not working.\n");
        return 1;
    }

    // Registering the SIGSEGV handler
    sa.sa_sigaction = sigsegv_handler;
    sa.sa_flags = SA_SIGINFO;
    sigaction(SIGSEGV, &sa, NULL);


    // Try to make a change in memory (will fail)
    printf("%s\n", addr); // "This is the content of the file."
    addr[0] = 'T';
    addr[1] = 'h';
    addr[2] = 'a';
    addr[3] = 't';
    printf("%s\n", addr); // "That is the content of the file."

    // Unmap the allocated memory
    munmap(addr, 20);
```

```
    return 0;
}
```

2.

```
#include <pthread.h>
#include <stdbool.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/mman.h>

const int n = 3;
pthread_t threads[3];



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

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

// Functions used to handle the SIGSEGV signal
void main_handler(int sig, siginfo_t* si, void* unused) {
    pthread_t thr = pthread_self();
    if(pthread_equal(thr, threads[0])) {
        printf("Thread 1\n");
    } else if (pthread_equal(thr, threads[1])) {
        printf("Thread 2\n");
    } else if (pthread_equal(thr, threads[2])) {
        printf("Thread 3\n");
    } else {
        printf("Main thread\n");
```

```
    }
    delay(5000000);
    printf("Handle done\n");
}


void* thread_function(void* args) {
    struct sigaction sa;
    int* x = (int*) args;
    int n = *x;

    printf("Starting with value: %d\n", n);

    raise(SIGSEGV);

    while ((*x) < 1000000) {
        (*x)++;
        delay(1000);
    }
    int* result = (int*) malloc(sizeof(int));
    *result = *x+n;
    return result;
}


int main() {
    int args[n];
    int* return_val[n];
    struct sigaction sa;

    // Registering the SIGSEGV handler
    sa.sa_sigaction = main_handler;
    sa.sa_flags = SA_SIGINFO;
    sigaction(SIGSEGV, &sa, NULL);

    for (int i=0; i<n; ++i) {
        args[i] = i;
```

```
        pthread_create(&threads[i], NULL, thread_function, &args[i]);
    }

    raise(SIGSEGV);

    for (int i=0; i<n; ++i) {
        pthread_join(threads[i], (void**)&return_val[i]);
    }
}
```

Cand semnalul SIGSEGV a fost transmis procesul folosind comanda kill, atunci oricare din cele 4 threaduri a putut fi ales pentru a procesa semnalul (atat threadul principal cat si cele create).

Daca un thread creaza un semnal SIGSEGV (acest lucru a fost testat folosind functia raise), atunci tot acesta va fi si cel care il va procesa.

Outputul unui experiment:

alex@alex-HP-Pavilion-Laptop-15-cs3xxx:~/Workspace/facultate/Master/OS: Design & Security/laborator 1$ ./thread

Main thread
Starting with value: 2
Thread 3
Starting with value: 0
Starting with value: 1
Thread 1
Thread 2
Handle done
Handle done
Handle done
Handle done
Main thread
Thread 1
Handle done
Main thread
Handle done
Thread 1
Handle done
Main thread
Thread 2
Handle done
Thread 3
Handle done
Handle done
Handle done
Main thread
Thread 3
Handle done
Main thread
Thread 1
Handle done
Handle done
Handle done
Main thread
Handle done

Threads handle signals sent by themselves

Threads handle signals sent to the process