Laboratory 1

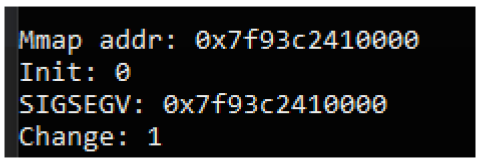
Practice:

1. memory management 1
    - sudo vim hello_world.c
    - cat hello_world.c
        ```
        #include <stdio.h>

        int main(){
                printf ("Hello World!");
                return 0;
        }
        ```
    - gcc -o hello_world hello_world.c
    - objdump -hx hello_world > objdump.txt
        - objdump.txt 3 kB
    - memory sections -> address
        - stack -> **0000000000000740**
        - heap -> **0000000000200dc8**
2. memory management 2
    - sudo vim loop.c
    - cat loop.c
        ```
        #include <stdio.h>


         int main(){
              int x = 1;
              while (x){
                     printf ("%d\n", x);
                     x++;
                     }
               return 0;
        }
        ```
    - gcc -o loop loop.c
    - ./loop
        - in paralel cu urmatoarea comanda (intr-un bash paralel)
    - ps -e | grep loop$
    - cat /proc/{pid_number}/maps > maps.txt
    - cat maps.txt
        - maps.txt 2 kB
        - -> address   perms offset dev   inode   pathname
        - libc si ld -> librarii
            - heap ->   **7fffef3c7000-7fffef3e8000 rw-p 00000000 00:00 0   [heap]**
            - stack ->   **7ffff6269000-7ffff6a69000 rw-p 00000000 00:00 0   [stack]**
            - vdso -> virtual dynamic shared object, pentru functii virtuale
            - vvar -> variabile folosite de vdso
            - vsyscall -> codul pentru funtii virtuale
    - ldd loop > ldd.txt
    - cat ldd.txt
        - ldd.txt 155 B
        - librariile de care depinde programul + locatia virtuala de memorie

1. threads 1
    - touch thread.c
    - cat thread.c
        ```
        #include <stdio.h>
        #include <sys/mman.h>
        ```

```
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>

void f (int flag1, siginfo_t *i, void *flag2) {
    printf("SIGSEGV: %p\n", i->si_addr);

    // function to add permission to write to the pages - function arguments
    mprotect(i->si_addr, sizeof(int), PROT_WRITE);

    // test changes
    *(int*)i->si_addr = 1;
    printf("Change: %d\n", *(int*)i->si_addr);

    _exit(EXIT_SUCCESS);
}

int main () {
        // aloca memorie virtuala
        /* mmap parameters:
                - addr       ->   NULL                                            ->  page-aligned chosen by kernel
                - length   ->   sizeof (int)                              ->  the length of the mapping
                - prot       ->   PROT_READ                                      ->  read rights to pages
                - flags     ->   MAP_PRIVATE | MAP_ANONYMOUS       ->  private copy-on-write mapping | mapping not backed
up and initialized with 0
                - fd     ->   0                                                     ->  file description
                - offset    ->   0                                                        ->  starting the addr at defined offset
        */
        int *ptr  = mmap (NULL, sizeof (int), PROT_READ, MAP_PRIVATE | MAP_ANONYMOUS, 0, 0);
        // resolve write rights
        // int *ptr  = mmap (NULL, sizeof (int), PROT_WRITE | PROT_READ, MAP_PRIVATE | MAP_ANONYMOUS, 0, 0);
        // error handleing
        if (ptr == MAP_FAILED) {
                printf("Error\n");
                return 0;
         }
        printf ("Mmap addr: %p\n", ptr);
        printf ("Init: %d\n", *ptr);

        // writing operation will raise a TypeError sau Segmentation Fault exception depending on used flags
        // const char *text = "hello";
        // memcpy (ptr, text, strlen(text)); //segmentation fault when only read rights
        struct sigaction x;

    x.sa_flags = SA_SIGINFO;
    x.sa_sigaction = f;

    // setup signal handler
    if (sigaction(SIGSEGV, &x, NULL) == -1) {
        perror("Sigaction");
        _exit(EXIT_FAILURE);
    }

    *ptr = 1; // memory violation by modifying the address

        return 0;
}
```

- gcc -o thread thread.c && ./thread
    1. output:

        1. 

```
Mmap addr: 0x7f93c2410000
Init: 0
SIGSEGV: 0x7f93c2410000
Change: 1
```

    -

4. threads 2
    - If I send a signal to a process which has more than one thread which thread will handle it?
        - at least one thread (which is not signal blocked) should be able receive signals
        - if there are more threads which can receive signals, a handle mechanism must be implemented in order to chose the thread
            - if there are more threads which are not signal blocked, the system chose a random one : "We can’t predict the thread that will be chosen to run the signal handler"
            - in order to chose a specific thread you can signal block the others or create a handling system
        - signal arrives > thread complete the instruction > thread jump to the signal handler
    - If a thread produces an invalid memory access, which thread will handle it?
        - the memory access produced by a thread will be treated by itself using SIGSEGV