Λειτουργικά Συστήματα – Άσκηση 4

Κυριακόπουλος Γιώργος – el18153
Τζελέπης Σεραφείμ – el18849

1.1

Πηγαίος κώδικας:

```
/*
 * mmap.c
 *
 * Examining the virtual memory of processes.
 *
 * Operating Systems course, CSLab, ECE, NTUA
 *
 */

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <stdint.h>
#include <signal.h>
#include <sys/wait.h>

#include "help.h"

#define RED     "\033[31m"
#define RESET   "\033[0m"


char *heap_private_buf;
char *heap_shared_buf;
```

```c
char *file_shared_buf;

uint64_t buffer_size;
off_t size;

/*
 * Child process' entry point.
 */
void child(void)
{
    uint64_t pa;

    /*
     * Step 7 - Child
     */
    if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
    /*
     * TODO: Write your code here to complete child's part of Step 7.
     */
    printf("Child's Virtual Memory Map\n");
    show_maps();

    /*
     * Step 8 - Child
     */
    if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
    /*
     * TODO: Write your code here to complete child's part of Step 8.
     */
    pa = get_physical_address((uint64_t)heap_private_buf);
    if(pa) {
        printf("Child: 0x%lu\n", pa);
    }

    /*
     * Step 9 - Child
     */
    if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
    /*
     * TODO: Write your code here to complete child's part of Step 9.
```

```
     */
    int i;
    for(i = 0; i < get_page_size(); i++){
        heap_private_buf[i] = 2;
    }

    pa = get_physical_address((uint64_t)heap_private_buf);
    if(pa) {
        printf("Child: 0x%lu\n", pa);
    }

    /*
     * Step 10 - Child
     */
    if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
    /*
     * TODO: Write your code here to complete child's part of Step 10
.
     */
    for(i = 0; i < get_page_size(); i++){
        heap_shared_buf[i] = 3;
    }

    pa = get_physical_address((uint64_t)heap_shared_buf);
    if(pa) {
        printf("Child: 0x%lu\n", pa);
    }

    /*
     * Step 11 - Child
     */
    if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
    /*
     * TODO: Write your code here to complete child's part of Step 11
.
     */
    if (mprotect(heap_shared_buf, buffer_size, PROT_READ) == -1) {
            perror("Failed to remove write protection on child proces
s.");
    }
```

```
    printf("Child:\n");
    show_va_info((uint64_t)heap_shared_buf);
    show_maps();

    /*
     * Step 12 - Child
     */
    /*
     * TODO: Write your code here to complete child's part of Step 12
.
     */
    if(munmap(heap_private_buf, 2 * get_page_size()) == -1) {
        perror("Failed to unmap heap_private_buf.");
    }
    if(munmap(heap_shared_buf, get_page_size()) == -1) {
        perror("Failed to unmap heap_shared_buf.");
    }
    if(munmap(file_shared_buf, size) == -1) {
        perror("Failed to unmap file_shared_buf.");
    }

    printf("Child:\n");
    show_va_info((uint64_t)heap_private_buf);
    show_va_info((uint64_t)heap_shared_buf);
    show_va_info((uint64_t)file_shared_buf);
    printf("\n");
}

/*
 * Parent process' entry point.
 */
void parent(pid_t child_pid)
{
    uint64_t pa;
    int status;

    /* Wait for the child to raise its first SIGSTOP. */
    if (-1 == waitpid(child_pid, &status, WUNTRACED))
        die("waitpid");

    /*
     * Step 7: Print parent's and child's maps. What do you see?
     * Step 7 - Parent
```

```
     */
    printf(RED "\nStep 7: Print parent's and child's map.\n" RESET);
    press_enter();

    /*
     * TODO: Write your code here to complete parent's part of Step 7
.
     */
    printf("Parent's Virtual Memory Map\n");
    show_maps();

    if (-1 == kill(child_pid, SIGCONT))
        die("kill");
    if (-1 == waitpid(child_pid, &status, WUNTRACED))
        die("waitpid");


    /*
     * Step 8: Get the physical memory address for heap_private_buf.
     * Step 8 - Parent
     */
    printf(RED "\nStep 8: Find the physical address of the private he
ap "
        "buffer (main) for both the parent and the child.\n" RESET);
    press_enter();

    /*
     * TODO: Write your code here to complete parent's part of Step 8
.
     */
    pa = get_physical_address((uint64_t)heap_private_buf);
    if(pa) {
        printf("Parent: 0x%lu\n", pa);
    }

    if (-1 == kill(child_pid, SIGCONT))
        die("kill");
    if (-1 == waitpid(child_pid, &status, WUNTRACED))
        die("waitpid");


    /*
     * Step 9: Write to heap_private_buf. What happened?
```

```
     * Step 9 - Parent
     */
    printf(RED "\nStep 9: Write to the private buffer from the child
and "
        "repeat step 8. What happened?\n" RESET);
    press_enter();

    /*
     * TODO: Write your code here to complete parent's part of Step 9
.
     */
    pa = get_physical_address((uint64_t)heap_private_buf);
    if(pa) {
        printf("Parent: 0x%lu\n", pa);
    }

    if (-1 == kill(child_pid, SIGCONT))
        die("kill");
    if (-1 == waitpid(child_pid, &status, WUNTRACED))
        die("waitpid");


    /*
     * Step 10: Get the physical memory address for heap_shared_buf.
     * Step 10 - Parent
     */
    printf(RED "\nStep 10: Write to the shared heap buffer (main) fro
m "
        "child and get the physical address for both the parent and "
        "the child. What happened?\n" RESET);
    press_enter();

    /*
     * TODO: Write your code here to complete parent's part of Step 1
0.
     */
    pa = get_physical_address((uint64_t)heap_shared_buf);
    if(pa) {
        printf("Parent: 0x%lu\n", pa);
    }

    if (-1 == kill(child_pid, SIGCONT))
        die("kill");
```

```
    if (-1 == waitpid(child_pid, &status, WUNTRACED))
        die("waitpid");


    /*
     * Step 11: Disable writing on the shared buffer for the child
     * (hint: mprotect(2)).
     * Step 11 - Parent
     */
    printf(RED "\nStep 11: Disable writing on the shared buffer for t
he "
        "child. Verify through the maps for the parent and the "
        "child.\n" RESET);
    press_enter();

    /*
     * TODO: Write your code here to complete parent's part of Step 1
1.
     */
    printf("Parent:\n");
    show_va_info((uint64_t)heap_shared_buf);
    show_maps();

    if (-1 == kill(child_pid, SIGCONT))
        die("kill");
    if (-1 == waitpid(child_pid, &status, 0))
        die("waitpid");


    /*
     * Step 12: Free all buffers for parent and child.
     * Step 12 - Parent
     */

    /*
     * TODO: Write your code here to complete parent's part of Step 1
2.
     */
    if(munmap(heap_private_buf, 2 * get_page_size()) == -1) {
        perror("Failed to unmap heap_private_buf.");
    }
    if(munmap(heap_shared_buf, get_page_size()) == -1) {
        perror("Failed to unmap heap_shared_buf.");
```

```
    }
    if(munmap(file_shared_buf, size) == -1) {
        perror("Failed to unmap file_shared_buf.");
    }

    printf("Parent:\n");
    show_va_info((uint64_t)heap_private_buf);
    show_va_info((uint64_t)heap_shared_buf);
    show_va_info((uint64_t)file_shared_buf);
    printf("\n");
}

int main(void)
{
    pid_t mypid, p;
    int fd = -1;
    uint64_t pa;

    mypid = getpid();
    buffer_size = 1 * get_page_size();

    /*
     * Step 1: Print the virtual address space layout of this process
.
     */
    printf(RED "\nStep 1: Print the virtual address space map of this
 "
        "process [%d].\n" RESET, mypid);
    press_enter();
    /*
     * TODO: Write your code here to complete Step 1.
     */
    show_maps();

    /*
     * Step 2: Use mmap to allocate a buffer of 1 page and print the
map
     * again. Store buffer in heap_private_buf.
     */
    printf(RED "\nStep 2: Use mmap(2) to allocate a private buffer of
 "
        "size equal to 1 page and print the VM map again.\n" RESET);
    press_enter();
```

```
    /*
     * TODO: Write your code here to complete Step 2.
     */
    heap_private_buf = mmap(NULL, buffer_size, PROT_READ | PROT_WRITE
, MAP_PRIVATE | MAP_ANONYMOUS, fd, 0);
    if(heap_private_buf == MAP_FAILED){
        perror("Failed to create new mapping (heap_private_buf - Step
 2)");
    }

    printf("\nThe Virtual Address Area, the Permissions etc. of the h
eap_private_buf is:\n");
    show_va_info((uint64_t)heap_private_buf);
    show_maps();

    /*
     * Step 3: Find the physical address of the first page of your bu
ffer
     * in main memory. What do you see?
    a */
    printf(RED "\nStep 3: Find and print the physical address of the
"
        "buffer in main memory. What do you see?\n" RESET);
    press_enter();
    /*
     * TODO: Write your code here to complete Step 3.
     */
    pa = get_physical_address((uint64_t)heap_private_buf);
    if(pa) {
        printf("0x%lu\n", pa);
    }
    /*
     * Step 4: Write zeros to the buffer and repeat Step 3.
     */
    printf(RED "\nStep 4: Initialize your buffer with zeros and repea
t "
        "Step 3. What happened?\n" RESET);
    press_enter();
    /*
     * TODO: Write your code here to complete Step 4.
     */
    int i;
    for(i = 0; i < get_page_size(); i++){
```

```
        heap_private_buf[i] = 0;
    }

    pa = get_physical_address((uint64_t)heap_private_buf);
    if(pa) {
        printf("0x%lu\n", pa);
    }

    /*
     * Step 5: Use mmap(2) to map file.txt (memory-
mapped files) and print
     * its content. Use file_shared_buf.
     */
    printf(RED "\nStep 5: Use mmap(2) to read and print file.txt. Pri
nt "
        "the new mapping information that has been created.\n" RESET)
;
    press_enter();
    /*
     * TODO: Write your code here to complete Step 5.
     */
    fd = open("file.txt", O_RDONLY);
    struct stat buf;
    fstat(fd, &buf);
    size = buf.st_size;

    // fseek(fd, 0, SEEK_END);
    // size = ftell(fp);
    // fseek(fd, 0, SEEK_SET);

    file_shared_buf = mmap(NULL, size, PROT_READ, MAP_PRIVATE, fd, 0)
;
    if(file_shared_buf == MAP_FAILED){
        perror("Failed to create new mapping (file_shared_buf - Step
5)");
    }

    for(i = 0; i < size; i++) {
        printf("%c", file_shared_buf[i]);
    }

    printf("\nThe Virtual Address Area, the Permissions etc. of the f
ile_shared_buf is:\n");
```

```
    show_va_info((uint64_t)file_shared_buf);
    show_maps();

    /*
     * Step 6: Use mmap(2) to allocate a shared buffer of 1 page. Use
     * heap_shared_buf.
     */
    printf(RED "\nStep 6: Use mmap(2) to allocate a shared buffer of
size "
        "equal to 1 page. Initialize the buffer and print the new "
        "mapping information that has been created.\n" RESET);
    press_enter();
    /*
     * TODO: Write your code here to complete Step 6.
     */
    heap_shared_buf = mmap(NULL, buffer_size, PROT_READ | PROT_WRITE,
 MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    if(heap_shared_buf == MAP_FAILED){
        perror("Failed to create new mapping (heap_shared_buf - Step
6)");
    }

    for(i = 0; i < get_page_size(); i++){
        heap_shared_buf[i] = 1;
    }

    printf("\nThe Virtual Address Area, the Permissions etc. of the h
eap_shared_buf is:\n");
    show_va_info((uint64_t)heap_shared_buf);
    show_maps();

    p = fork();
    if (p < 0)
        die("fork");
    if (p == 0) {
        child();
        return 0;
    }

    parent(p);

    if (-1 == close(fd))
        perror("close");
```

```
    return 0;
}
```

Έξοδος εκτέλεσης προγράμματος:

oslaba40@os-node1:~/ex4/a$ ./mmap

Step 1: Print the virtual address space map of this process [15104].

Virtual Memory Map of process [15104]:

00400000-00403000 r-xp 00000000 00:21 20323478 /home/oslab/oslaba40/ex4/a/mmap

00602000-00603000 rw-p 00002000 00:21 20323478 /home/oslab/oslaba40/ex4/a/mmap

010f3000-01114000 rw-p 00000000 00:00 0 [heap]

7f6695532000-7f66956d3000 r-xp 00000000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66956d3000-7f66958d3000 ---p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d3000-7f66958d7000 r--p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d7000-7f66958d9000 rw-p 001a5000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d9000-7f66958dd000 rw-p 00000000 00:00 0

7f66958dd000-7f66958fe000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695af0000-7f6695af3000 rw-p 00000000 00:00 0

7f6695af8000-7f6695afd000 rw-p 00000000 00:00 0

7f6695afd000-7f6695afe000 r--p 00020000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695afe000-7f6695aff000 rw-p 00021000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695aff000-7f6695b00000 rw-p 00000000 00:00 0

7ffc45556000-7ffc45577000 rw-p 00000000 00:00 0 [stack]

7ffc455a0000-7ffc455a3000 r--p 00000000 00:00 0 [vvar]

7ffc455a3000-7ffc455a5000 r-xp 00000000 00:00 0 [vdso]

ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]

Step 2: Use mmap(2) to allocate a private buffer of size equal to 1 page and print the VM map again.

The Virtual Address Area, the Permissions etc. of the heap_private_buf is:

7f6695af7000-7f6695afd000 rw-p 00000000 00:00 0

Virtual Memory Map of process [15104]:

00400000-00403000 r-xp 00000000 00:21 20323478 /home/oslab/oslaba40/ex4/a/mmap

00602000-00603000 rw-p 00002000 00:21 20323478 /home/oslab/oslaba40/ex4/a/mmap

010f3000-01114000 rw-p 00000000 00:00 0 [heap]

7f6695532000-7f66956d3000 r-xp 00000000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66956d3000-7f66958d3000 ---p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d3000-7f66958d7000 r--p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d7000-7f66958d9000 rw-p 001a5000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d9000-7f66958dd000 rw-p 00000000 00:00 0

7f66958dd000-7f66958fe000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695af0000-7f6695af3000 rw-p 00000000 00:00 0

7f6695af7000-7f6695afd000 rw-p 00000000 00:00 0

7f6695afd000-7f6695afe000 r--p 00020000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695afe000-7f6695aff000 rw-p 00021000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695aff000-7f6695b00000 rw-p 00000000 00:00 0

7ffc45556000-7ffc45577000 rw-p 00000000 00:00 0 [stack]

7ffc455a0000-7ffc455a3000 r--p 00000000 00:00 0 [vvar]

7ffc455a3000-7ffc455a5000 r-xp 00000000 00:00 0 [vdso]

ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]

Step 3: Find and print the physical address of the buffer in main memory. What do you see?

VA[0x7f6695af8000] is not mapped; no physical memory allocated.

Step 4: Initialize your buffer with zeros and repeat Step 3. What happened?

0x138948608

Step 5: Use mmap(2) to read and print file.txt. Print the new mapping information that has been created.

Hello everyone!

The Virtual Address Area, the Permissions etc. of the file_shared_buf is:

7f6695af7000-7f6695af8000 r--p 00000000 00:21 20324384 /home/oslab/oslaba40/ex4/a/file.txt

Virtual Memory Map of process [15104]:

00400000-00403000 r-xp 00000000 00:21 20323478 /home/oslab/oslaba40/ex4/a/mmap

00602000-00603000 rw-p 00002000 00:21 20323478 /home/oslab/oslaba40/ex4/a/mmap

010f3000-01114000 rw-p 00000000 00:00 0 [heap]

7f6695532000-7f66956d3000 r-xp 00000000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66956d3000-7f66958d3000 ---p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d3000-7f66958d7000 r--p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d7000-7f66958d9000 rw-p 001a5000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d9000-7f66958dd000 rw-p 00000000 00:00 0

7f66958dd000-7f66958fe000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695af0000-7f6695af3000 rw-p 00000000 00:00 0

7f6695af6000-7f6695af7000 rw-p 00000000 00:00 0

7f6695af7000-7f6695af8000 r--p 00000000 00:21 20324384 /home/oslab/oslaba40/ex4/a/file.txt

7f6695af8000-7f6695afd000 rw-p 00000000 00:00 0

7f6695afd000-7f6695afe000 r--p 00020000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695afe000-7f6695aff000 rw-p 00021000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695aff000-7f6695b00000 rw-p 00000000 00:00 0

7ffc45556000-7ffc45577000 rw-p 00000000 00:00 0 [stack]

7ffc455a0000-7ffc455a3000 r--p 00000000 00:00 0 [vvar]

7ffc455a3000-7ffc455a5000 r-xp 00000000 00:00 0 [vdso]

ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]

Step 6: Use mmap(2) to allocate a shared buffer of size equal to 1 page. Initialize the buffer and print the new mapping information that has been created.

The Virtual Address Area, the Permissions etc. of the heap_shared_buf is:

7f6695af6000-7f6695af7000 rw-s 00000000 00:04 2144510 /dev/zero (deleted)

Virtual Memory Map of process [15104]:

00400000-00403000 r-xp 00000000 00:21 20323478 /home/oslab/oslaba40/ex4/a/mmap

00602000-00603000 rw-p 00002000 00:21 20323478 /home/oslab/oslaba40/ex4/a/mmap

010f3000-01114000 rw-p 00000000 00:00 0 [heap]

7f6695532000-7f66956d3000 r-xp 00000000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66956d3000-7f66958d3000 ---p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d3000-7f66958d7000 r--p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d7000-7f66958d9000 rw-p 001a5000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d9000-7f66958dd000 rw-p 00000000 00:00 0

7f66958dd000-7f66958fe000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695af0000-7f6695af3000 rw-p 00000000 00:00 0

7f6695af5000-7f6695af6000 rw-p 00000000 00:00 0

7f6695af6000-7f6695af7000 rw-s 00000000 00:04 2144510 /dev/zero (deleted)

7f6695af7000-7f6695af8000 r--p 00000000 00:21 20324384
/home/oslab/oslaba40/ex4/a/file.txt

7f6695af8000-7f6695afd000 rw-p 00000000 00:00 0

7f6695afd000-7f6695afe000 r--p 00020000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695afe000-7f6695aff000 rw-p 00021000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695aff000-7f6695b00000 rw-p 00000000 00:00 0

7ffc45556000-7ffc45577000 rw-p 00000000 00:00 0 [stack]

7ffc455a0000-7ffc455a3000 r--p 00000000 00:00 0 [vvar]

7ffc455a3000-7ffc455a5000 r-xp 00000000 00:00 0 [vdso]

ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]

Step 7: Print parent's and child's map.

Parent's Virtual Memory Map

Virtual Memory Map of process [15104]:

00400000-00403000 r-xp 00000000 00:21 20323478
/home/oslab/oslaba40/ex4/a/mmap

00602000-00603000 rw-p 00002000 00:21 20323478
/home/oslab/oslaba40/ex4/a/mmap

010f3000-01114000 rw-p 00000000 00:00 0 [heap]

7f6695532000-7f66956d3000 r-xp 00000000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66956d3000-7f66958d3000 ---p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d3000-7f66958d7000 r--p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d7000-7f66958d9000 rw-p 001a5000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d9000-7f66958dd000 rw-p 00000000 00:00 0

7f66958dd000-7f66958fe000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695af0000-7f6695af3000 rw-p 00000000 00:00 0

7f6695af5000-7f6695af6000 rw-p 00000000 00:00 0

7f6695af6000-7f6695af7000 rw-s 00000000 00:04 2144510 /dev/zero (deleted)

7f6695af7000-7f6695af8000 r--p 00000000 00:21 20324384 /home/oslab/oslaba40/ex4/a/file.txt

7f6695af8000-7f6695afd000 rw-p 00000000 00:00 0

7f6695afd000-7f6695afe000 r--p 00020000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695afe000-7f6695aff000 rw-p 00021000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695aff000-7f6695b00000 rw-p 00000000 00:00 0

7ffc45556000-7ffc45577000 rw-p 00000000 00:00 0 [stack]

7ffc455a0000-7ffc455a3000 r--p 00000000 00:00 0 [vvar]

7ffc455a3000-7ffc455a5000 r-xp 00000000 00:00 0 [vdso]

ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]

Child's Virtual Memory Map

Virtual Memory Map of process [15105]:

00400000-00403000 r-xp 00000000 00:21 20323478 /home/oslab/oslaba40/ex4/a/mmap

00602000-00603000 rw-p 00002000 00:21 20323478 /home/oslab/oslaba40/ex4/a/mmap

010f3000-01114000 rw-p 00000000 00:00 0 [heap]

7f6695532000-7f66956d3000 r-xp 00000000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66956d3000-7f66958d3000 ---p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d3000-7f66958d7000 r--p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d7000-7f66958d9000 rw-p 001a5000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d9000-7f66958dd000 rw-p 00000000 00:00 0

7f66958dd000-7f66958fe000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695af0000-7f6695af3000 rw-p 00000000 00:00 0

7f6695af5000-7f6695af6000 rw-p 00000000 00:00 0

7f6695af6000-7f6695af7000 rw-s 00000000 00:04 2144510 /dev/zero (deleted)

7f6695af7000-7f6695af8000 r--p 00000000 00:21 20324384 /home/oslab/oslaba40/ex4/a/file.txt

7f6695af8000-7f6695afd000 rw-p 00000000 00:00 0

7f6695afd000-7f6695afe000 r--p 00020000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695afe000-7f6695aff000 rw-p 00021000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695aff000-7f6695b00000 rw-p 00000000 00:00 0

7ffc45556000-7ffc45577000 rw-p 00000000 00:00 0 [stack]

7ffc455a0000-7ffc455a3000 r--p 00000000 00:00 0 [vvar]

7ffc455a3000-7ffc455a5000 r-xp 00000000 00:00 0 [vdso]

ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]

Step 8: Find the physical address of the private heap buffer (main) for both the parent and the child.

Parent: 0x138948608

Child: 0x138948608

Step 9: Write to the private buffer from the child and repeat step 8. What happened?

Parent: 0x138948608

Child: 0x1043533824

Step 10: Write to the shared heap buffer (main) from child and get the physical address for both the parent and the child. What happened?

Parent: 0x3057258496

Child: 0x3057258496

Step 11: Disable writing on the shared buffer for the child. Verify through the maps for the parent and the child.

Parent:

7f6695af6000-7f6695af7000 rw-s 00000000 00:04 2144510 /dev/zero (deleted)

Virtual Memory Map of process [15104]:

00400000-00403000 r-xp 00000000 00:21 20323478
/home/oslab/oslaba40/ex4/a/mmap

00602000-00603000 rw-p 00002000 00:21 20323478
/home/oslab/oslaba40/ex4/a/mmap

010f3000-01114000 rw-p 00000000 00:00 0 [heap]

7f6695532000-7f66956d3000 r-xp 00000000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66956d3000-7f66958d3000 ---p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d3000-7f66958d7000 r--p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d7000-7f66958d9000 rw-p 001a5000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d9000-7f66958dd000 rw-p 00000000 00:00 0

7f66958dd000-7f66958fe000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695af0000-7f6695af3000 rw-p 00000000 00:00 0

7f6695af5000-7f6695af6000 rw-p 00000000 00:00 0

7f6695af6000-7f6695af7000 rw-s 00000000 00:04 2144510 /dev/zero (deleted)

7f6695af7000-7f6695af8000 r--p 00000000 00:21 20324384 /home/oslab/oslaba40/ex4/a/file.txt

7f6695af8000-7f6695afd000 rw-p 00000000 00:00 0

7f6695afd000-7f6695afe000 r--p 00020000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695afe000-7f6695aff000 rw-p 00021000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695aff000-7f6695b00000 rw-p 00000000 00:00 0

7ffc45556000-7ffc45577000 rw-p 00000000 00:00 0 [stack]

7ffc455a0000-7ffc455a3000 r--p 00000000 00:00 0 [vvar]

7ffc455a3000-7ffc455a5000 r-xp 00000000 00:00 0 [vdso]

ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]

Child:

7f6695af6000-7f6695af7000 r--s 00000000 00:04 2144510 /dev/zero (deleted)

Virtual Memory Map of process [15105]:

00400000-00403000 r-xp 00000000 00:21 20323478 /home/oslab/oslaba40/ex4/a/mmap

00602000-00603000 rw-p 00002000 00:21 20323478 /home/oslab/oslaba40/ex4/a/mmap

010f3000-01114000 rw-p 00000000 00:00 0 [heap]

7f6695532000-7f66956d3000 r-xp 00000000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66956d3000-7f66958d3000 ---p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d3000-7f66958d7000 r--p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d7000-7f66958d9000 rw-p 001a5000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so

7f66958d9000-7f66958dd000 rw-p 00000000 00:00 0

7f66958dd000-7f66958fe000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695af0000-7f6695af3000 rw-p 00000000 00:00 0

7f6695af5000-7f6695af6000 rw-p 00000000 00:00 0

7f6695af6000-7f6695af7000 r--s 00000000 00:04 2144510 /dev/zero (deleted)

7f6695af7000-7f6695af8000 r--p 00000000 00:21 20324384 /home/oslab/oslaba40/ex4/a/file.txt

7f6695af8000-7f6695afd000 rw-p 00000000 00:00 0

7f6695afd000-7f6695afe000 r--p 00020000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695afe000-7f6695aff000 rw-p 00021000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so

7f6695aff000-7f6695b00000 rw-p 00000000 00:00 0

7ffc45556000-7ffc45577000 rw-p 00000000 00:00 0 [stack]

7ffc455a0000-7ffc455a3000 r--p 00000000 00:00 0 [vvar]

7ffc455a3000-7ffc455a5000 r-xp 00000000 00:00 0 [vdso]

ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]

Child:

VA is not allocated!

VA is not allocated!

VA is not allocated!

Parent:

VA is not allocated!

VA is not allocated!

VA is not allocated!

Ερωτήσεις:

1. Βλέπουμε το χάρτη της εικονικής μνήμης της διεργασία, με τις αντίστοιχες στήλες για τις εικονικές διευθύνσεις, protections, offset, devices, inode και τέλος pathname του αρχείου που γίνεται mapping (ή κενό για ANONYMOUS mappings χωρίς αρχείο).

2. Εντοπίζουμε στον χάρτη μνήμης τον δεσμευμένο χώρο εικονικών διευθύνσεων για τον heap_private_buf, με τα αναμενόμενα protections και χωρίς κάποιο pathname λόγω της MAP_ANONYMOUS.

3. Λαμβάνουμε το μήνυμα πως η συγκεκριμένη εικονική διεύθυνση δεν έχει, αντίστοιχα, κάποια δεσμευμένη φυσική διεύθυνση μνήμης. Αυτό συμβαίνει επειδή έχουμε κάνει το map για τον heap_private_buf, αλλά δεν τον έχουμε χρησιμοποιήσει για να αποθηκεύσουμε δεδομένα. Λόγω και του memory overcommitment concept, το λειτουργικό μας σύστημα θέλει να δίνει τη ψευδαίσθηση στις διεργασίες πως έχουν πρόσβαση σε περισσότερη μνήμη από ότι υπάρχει διαθέσιμη, επομένως κάνει τα mappings για τις εικονικές μνήμες, όμως κρατάει τη φυσική μνήμη χωρίς αντίστοιχες δεσμεύσεις έως ότου πραγματικά χρειαστεί για αποθήκευση δεδομένων.

4. Αφού αρχικοποιήσουμε τον heap_private_buf με μηδενικά, τρέχουμε πάλι το 3ο βήμα και βλέπουμε ότι πλέον τυπώνεται η φυσική διεύθυνση μνήμης που έχει δεσμευθεί.

5. Μετά το mapping του αρχείου, παρατηρούμε το περιεχόμενο του αρχείου ως έξοδο στο τερματικό μας, καθώς και τη νέα εγγραφή στο χάρτη μνήμης.

6. Εντοπίζουμε στο χάρτη μνήμης τη νέα εγγραφή για τον heap_shared_buf, με pathname /dev/zero (deleted), λόγω της zero αρχικοποίησης των shared και uninitialized buffers.

7. Παρατηρούμε τους δύο χάρτες μνήμης και όπως περιμέναμε, είναι πιστά αντίγραφα ο ένας του άλλου, λόγω της fork() που έγινε για τη δημιουργία της διαδικασίας παιδί.

8. Βλέπουμε ότι ο πατέρας και το παιδί έχουν λάβει ένα αντίγραφο της φυσικής διεύθυνσης μνήμης που είχε ο heap_private_buf.

9. Η φυσική διεύθυνση του heap_private_buf της διεργασίας παιδί έχει αλλάξει. Αυτό συμβαίνει αφού ο buffer αυτός είναι private και όχι shared, επομένως τα δύο αυτά αντίγραφα του πατέρα και του παιδιού πρέπει να διαχωριστούν μεταξύ τους και να δημιουργηθεί ένα διαφορετικό για το παιδί, αφού ο πατέρας δεν πρέπει να έχει πρόσβαση στις αλλαγές που κάνει το παιδί και αντίστροφα.

10. Σε αντίθετη περίπτωση με το 9ο βήμα, έχουμε τον heap_shared_buf που είναι ένας shared buffer, επομένως ότι αλλαγή κάνει ο πατέρας θα πρέπει να τη βλέπει και το παιδί και αντίστροφα. Για αυτό το λόγο κρατάνε και τα δύο το ίδιο αντίγραφο του buffer και δεν αλλάζει για το παιδί, μετά τις αλλαγές που πραγματοποίησε.

11. Παρατηρούμε στο χάρτη μνήμη πως ο πατέρας έχει ακόμα rw-s protections, δηλαδή δικαίωμα ανάγνωσης και εγγραφής στον shared buffer, ενώ το παιδί έχει r--s protections, δηλαδή έχει επιτυχώς χάσει το δικαίωμα εγγραφής του στον shared buffer και έχει, επομένως, μόνο δικαίωμα ανάγνωσης.

12. Αποδεσμεύουμε όλους τους buffers και εκτυπώνουμε κατάλληλα μηνύματα για επιβεβαίωση.

1.2.1

Πηγαίος κώδικας:

```c
/*
 * mandel.c
 *
 * A program to draw the Mandelbrot Set on a 256-color xterm.
 *
 */

#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/wait.h>

/*TODO header file for m(un)map*/
#include <sys/mman.h>
#include "mandel-lib.h"


#define MANDEL_MAX_ITERATION 100000

/***************************
 * Compile-time parameters *
 ***************************/

/*
 * Output at the terminal is is x_chars wide by y_chars long
*/
int y_chars = 50;
int x_chars = 90;

/*
 * The part of the complex plane to be drawn:
 * upper left corner is (xmin, ymax), lower right corner is (xmax, ym
in)
*/
double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;
```

```
/*
 * Every character in the final output is
 * xstep x ystep units wide on the complex plane.
 */
double xstep;
double ystep;

int nprocs;
sem_t *mutex;

int safe_atoi(char *s, int *val)
{
    long l;
    char *endp;

    l = strtol(s, &endp, 10);
    if (s != endp && *endp == '\0') {
        *val = l;
        return 0;
    } else
        return -1;
}

void usage(char *argv0)
{
    fprintf(stderr, "Usage: %s thread_count \n\n"
        "Exactly one argument required:\n"
        "    thread_count: The number of threads to create.\n",
        argv0);
    exit(1);
}

/*
 * This function computes a line of output
 * as an array of x_char color values.
 */
void compute_mandel_line(int line, int color_val[])
{
    /*
     * x and y traverse the complex plane.
     */
    double x, y;
```

```
    int n;
    int val;

    /* Find out the y value corresponding to this line */
    y = ymax - ystep * line;

    /* and iterate for all points on this line */
    for (x = xmin, n = 0; n < x_chars; x+= xstep, n++) {

        /* Compute the point's color value */
        val = mandel_iterations_at_point(x, y, MANDEL_MAX_ITERATION);
        if (val > 255)
            val = 255;

        /* And store it in the color_val[] array */
        val = xterm_color(val);
        color_val[n] = val;
    }
}

/*
 * This function outputs an array of x_char color values
 * to a 256-color xterm.
 */
void output_mandel_line(int fd, int color_val[])
{
    int i;

    char point ='@';
    char newline='\n';

    for (i = 0; i < x_chars; i++) {
        /* Set the current color, then output the point */
        set_xterm_color(fd, color_val[i]);
        if (write(fd, &point, 1) != 1) {
            perror("compute_and_output_mandel_line: write point");
            exit(1);
        }
    }

    /* Now that the line is done, output a newline character */
    if (write(fd, &newline, 1) != 1) {
```

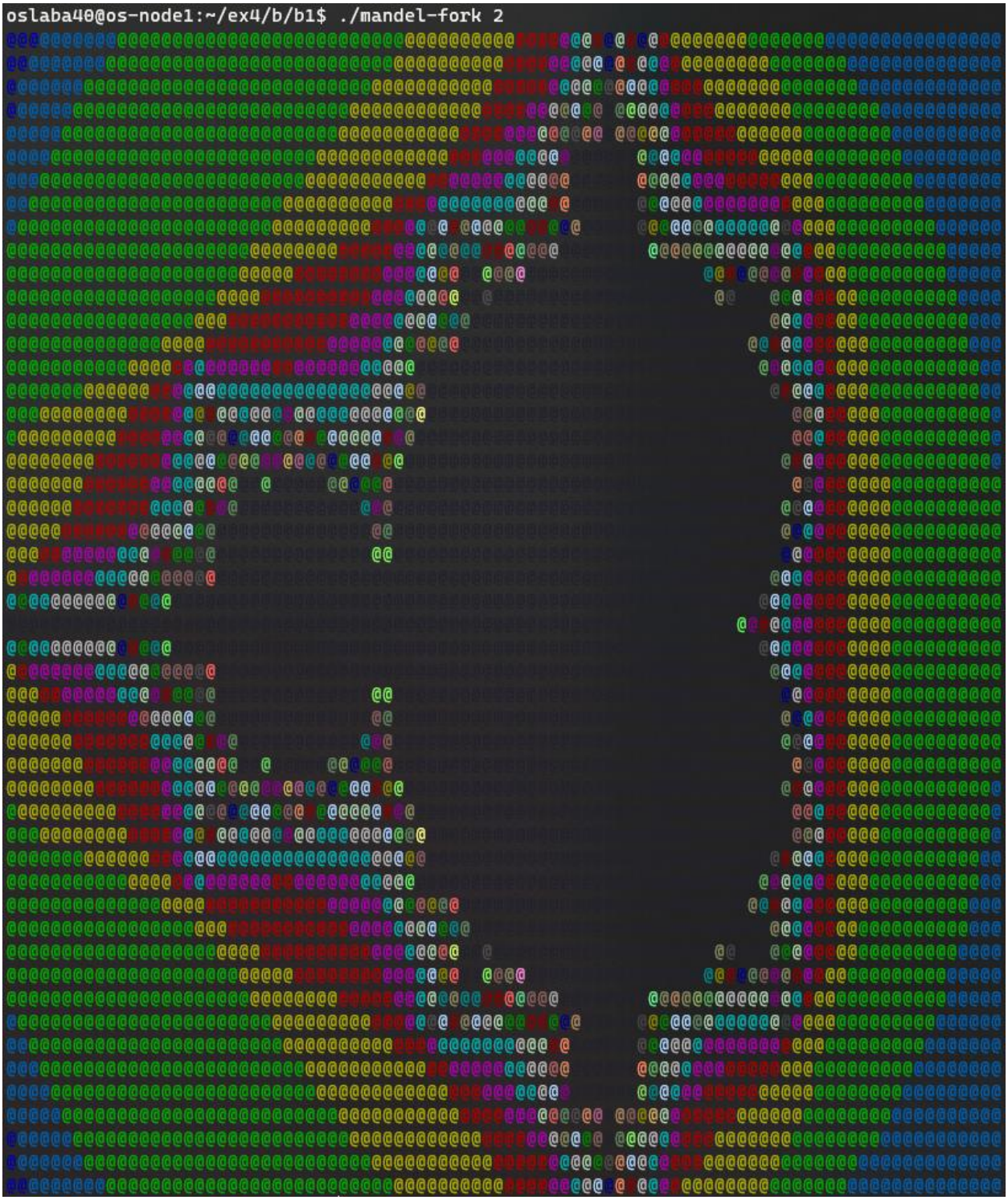
```
        perror("compute_and_output_mandel_line: write newline");
        exit(1);
    }
}

void compute_and_output_mandel_line(int fd, int line)
{
    /*
     * A temporary array, used to hold color values for the line bein
g drawn
     */
    int i, color_val[x_chars];

    for (i = line; i < y_chars; i += nprocs){
        int current = i % nprocs;
        int next = (i + 1) % nprocs;

        compute_mandel_line(i, color_val);
        sem_wait(&mutex[current]);

        output_mandel_line(fd, color_val);
        sem_post(&mutex[next]);
    }
}

/*
 * Create a shared memory area, usable by all descendants of the call
ing
 * process.
 */
void *create_shared_memory_area(unsigned int numbytes)
{
    int pages;
    void *addr;

    if (numbytes == 0) {
        fprintf(stderr, "%s: internal error: called for numbytes == 0
\n", __func__);
        exit(1);
    }

    /*
```

```
     * Determine the number of pages needed, round up the requested n
umber of
     * pages
     */
    pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;

    /* Create a shared, anonymous mapping for this number of pages */
    /* TODO:
        addr = mmap(...)
    */
    addr = mmap(NULL, pages * sysconf(_SC_PAGE_SIZE), PROT_READ | PRO
T_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);

    return addr;
}

void destroy_shared_memory_area(void *addr, unsigned int numbytes) {
    int pages;

    if (numbytes == 0) {
        fprintf(stderr, "%s: internal error: called for numbytes == 0
\n", __func__);
        exit(1);
    }

    /*
     * Determine the number of pages needed, round up the requested n
umber of
     * pages
     */
    pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;

    if (munmap(addr, pages * sysconf(_SC_PAGE_SIZE)) == -1) {
        perror("destroy_shared_memory_area: munmap failed");
        exit(1);
    }
}

int main(int argc, char **argv)
{
    int line, status;
    pid_t pid;
```

```
    if (argc != 2) {
        usage(argv[0]);
    }
    if (safe_atoi(argv[1], &nprocs) < 0 || nprocs <= 0) {
        fprintf(stderr, "`%s' is not valid for number of processes.\n
", argv[1]);
        exit(1);
    }

    xstep = (xmax - xmin) / x_chars;
    ystep = (ymax - ymin) / y_chars;

    mutex = create_shared_memory_area(nprocs * sizeof(sem_t));

    for (line = 0; line < nprocs; line++) {
        if(line == 0) {
            sem_init(&mutex[line], 1, 1);
        }
        else {
            sem_init(&mutex[line], 1, 0);
        }

        pid = fork();
        if(pid < 0) {
            perror("failed to fork: fork\n");
            exit(1);
        }
        else if(pid == 0) {
            compute_and_output_mandel_line(1, line);
            exit(0);
        }
    }

    for(line = 0; line < nprocs; line++) {
        pid = waitpid(-1, &status, 0);
        if (pid == -1) {
            perror("waitpid");
            exit(1);
        }
    }

    for(line = 0; line < nprocs; line++) {
        sem_destroy(&mutex[line]);
```

```
    }

    destroy_shared_memory_area(mutex, nprocs * sizeof(sem_t));

    /*
     * draw the Mandelbrot Set, one line at a time.
     * Output is sent to file descriptor '1', i.e., standard output.
     */

    // for (line = 0; line < y_chars; line++) {
    //  compute_and_output_mandel_line(1, line);
    // }

    reset_xterm_color(1);
    return 0;
}
```

Έξοδος εκτέλεσης προγράμματος:



Ερωτήσεις:

1. Παρότι στο δικό μας πρόβλημα με χρήση της time δεν παρατηρούμε κάποια αισθητή διαφορά στις επιδόσεις των δύο προγραμμάτων, η έκδοση με τα threads, θεωρητικά, θα περιμέναμε να έχει καλύτερη επίδοση. Αυτό οφείλεται κυρίως στο γεγονός ότι για να χρησιμοποιήσουμε processes απαιτείται αρχικά η προετοιμασία της κοινής μνήμης που θα χρησιμοποιούν οι διεργασίες (στην περίπτωση μας γίνεται απλά με ένα mmap() system call), καθώς και η χρήση της fork() για τη δημιουργία των διεργασιών-παιδιών, ένα system call το οποίο είναι αρκετά βαρύ σε σχέση με τη χρήση των threads, που έχουν πολύ πιο ελαφριά υλοποίηση και δεν απαιτούν κάποια επιπλέον κίνηση για τη μνήμη που θα χρησιμοποιούν, καθώς μοιράζονται αυτή της διεργασίας τους. Παρ' όλα αυτά, το δικό μας πρόβλημα είναι ένα απλό υπολογιστικά πρόβλημα και αυτές οι διαφορές δεν προλαβαίνουν να μας δημιουργήσουν κάποια διαφορά στο χρόνο εκτέλεσης κλπ. του προγράμματος.

1.2.2

Πηγαίος κώδικας:

```c
/*
 * mandel.c
 *
 * A program to draw the Mandelbrot Set on a 256-color xterm.
 *
 */

#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <sys/wait.h>

/*TODO header file for m(un)map*/
#include <sys/mman.h>
#include "mandel-lib.h"


#define MANDEL_MAX_ITERATION 100000

/***************************
 * Compile-time parameters *
 ***************************/

/*
 * Output at the terminal is is x_chars wide by y_chars long
*/
int y_chars = 50;
int x_chars = 90;

/*
 * The part of the complex plane to be drawn:
 * upper left corner is (xmin, ymax), lower right corner is (xmax, ym
in)
*/
double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;
```

```c
/*
 * Every character in the final output is
 * xstep x ystep units wide on the complex plane.
 */
double xstep;
double ystep;

int nprocs;
int *buf;

int safe_atoi(char *s, int *val)
{
    long l;
    char *endp;

    l = strtol(s, &endp, 10);
    if (s != endp && *endp == '\0') {
        *val = l;
        return 0;
    } else
        return -1;
}

void usage(char *argv0)
{
    fprintf(stderr, "Usage: %s thread_count \n\n"
        "Exactly one argument required:\n"
        "    thread_count: The number of threads to create.\n",
        argv0);
    exit(1);
}

/*
 * This function computes a line of output
 * as an array of x_char color values.
 */
void compute_mandel_line(int line, int color_val[])
{
    /*
     * x and y traverse the complex plane.
     */
    double x, y;
```

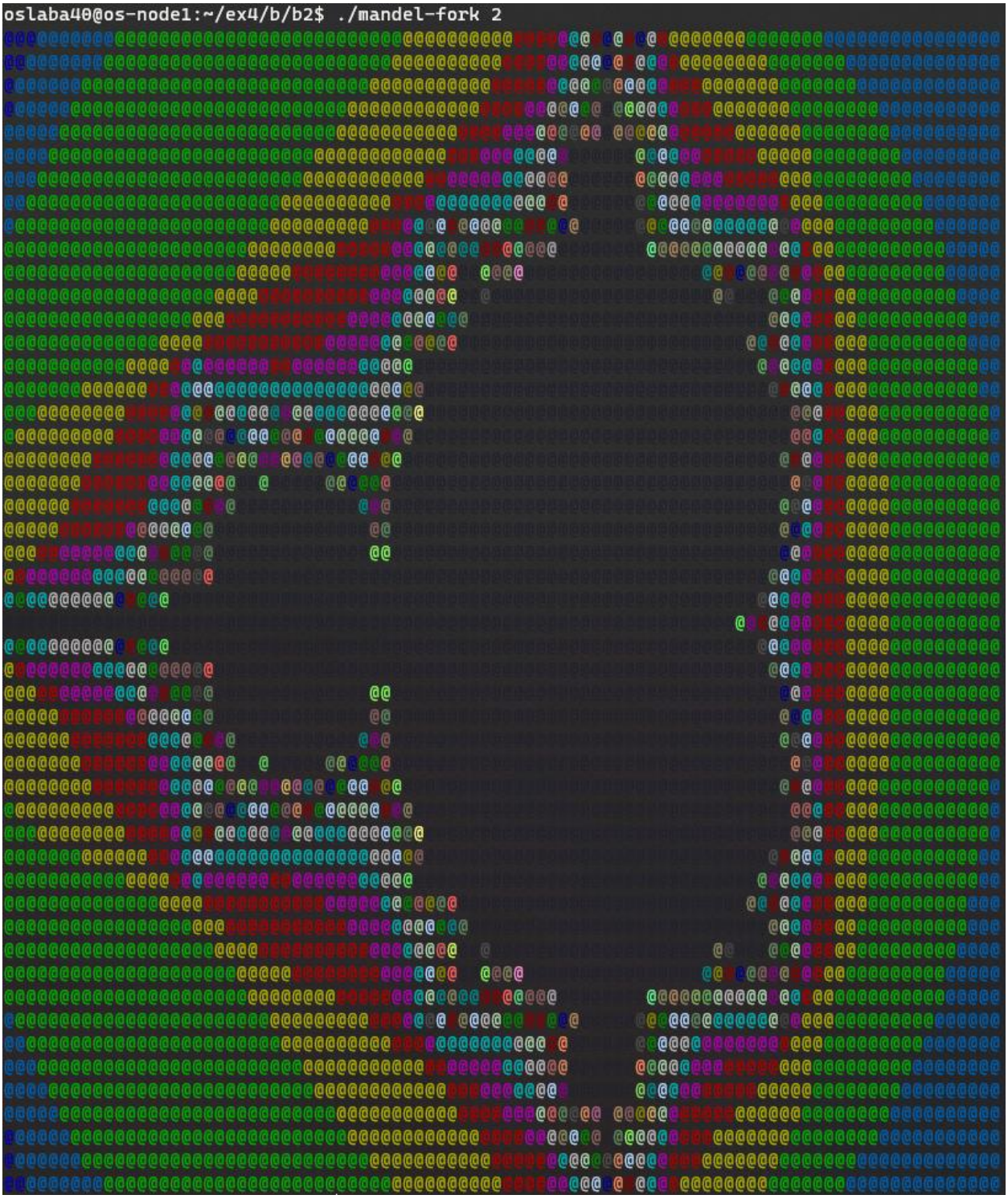
```
    int n;
    int val;

    /* Find out the y value corresponding to this line */
    y = ymax - ystep * line;

    /* and iterate for all points on this line */
    for (x = xmin, n = 0; n < x_chars; x+= xstep, n++) {

        /* Compute the point's color value */
        val = mandel_iterations_at_point(x, y, MANDEL_MAX_ITERATION);
        if (val > 255)
            val = 255;

        /* And store it in the color_val[] array */
        val = xterm_color(val);
        color_val[n] = val;
    }
}

/*
 * This function outputs an array of x_char color values
 * to a 256-color xterm.
 */
void output_mandel_line(int fd, int color_val[])
{
    int i;

    char point ='@';
    char newline='\n';

    for (i = 0; i < x_chars; i++) {
        /* Set the current color, then output the point */
        set_xterm_color(fd, color_val[i]);
        if (write(fd, &point, 1) != 1) {
            perror("compute_and_output_mandel_line: write point");
            exit(1);
        }
    }

    /* Now that the line is done, output a newline character */
    if (write(fd, &newline, 1) != 1) {
        perror("compute_and_output_mandel_line: write newline");
```

```
        exit(1);
    }
}

void compute_and_output_mandel_line(int fd, int line)
{
    /*
     * A temporary array, used to hold color values for the line bein
g drawn
     */
    int i;

    for (i = line; i < y_chars; i += nprocs){
        compute_mandel_line(i, buf + i * x_chars);
    }
}

/*
 * Create a shared memory area, usable by all descendants of the call
ing
 * process.
 */
void *create_shared_memory_area(unsigned int numbytes)
{
    int pages;
    void *addr;

    if (numbytes == 0) {
        fprintf(stderr, "%s: internal error: called for numbytes == 0
\n", __func__);
        exit(1);
    }

    /*
     * Determine the number of pages needed, round up the requested n
umber of
     * pages
     */
    pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;

    /* Create a shared, anonymous mapping for this number of pages */
    /* TODO:
        addr = mmap(...)
```

```
    */
    addr = mmap(NULL, pages * sysconf(_SC_PAGE_SIZE), PROT_READ | PRO
T_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);

    return addr;
}

void destroy_shared_memory_area(void *addr, unsigned int numbytes) {
    int pages;

    if (numbytes == 0) {
        fprintf(stderr, "%s: internal error: called for numbytes == 0
\n", __func__);
        exit(1);
    }

    /*
     * Determine the number of pages needed, round up the requested n
umber of
     * pages
     */
    pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;

    if (munmap(addr, pages * sysconf(_SC_PAGE_SIZE)) == -1) {
        perror("destroy_shared_memory_area: munmap failed");
        exit(1);
    }
}

int main(int argc, char **argv)
{
    int line, status;
    pid_t pid;

    if (argc != 2) {
        usage(argv[0]);
    }
    if (safe_atoi(argv[1], &nprocs) < 0 || nprocs <= 0) {
        fprintf(stderr, "`%s' is not valid for number of processes.\n
", argv[1]);
        exit(1);
    }
```

```
    xstep = (xmax - xmin) / x_chars;
    ystep = (ymax - ymin) / y_chars;

    buf = create_shared_memory_area(y_chars * x_chars * sizeof(int));

    for (line = 0; line < nprocs; line++) {
        pid = fork();
        if(pid < 0) {
            perror("failed to fork in main: fork\n");
            exit(1);
        }
        else if(pid == 0) {
            compute_and_output_mandel_line(1, line);
            exit(0);
        }
    }

    for(line = 0; line < nprocs; line++) {
        pid = waitpid(-1, &status, 0);
        if (pid == -1) {
            perror("waitpid in main");
            exit(1);
        }
    }

    for(line = 0; line < y_chars; line++) {
        output_mandel_line(1, buf + line * x_chars);
    }

    destroy_shared_memory_area(buf, y_chars * x_chars * sizeof(int));

    /*
     * draw the Mandelbrot Set, one line at a time.
     * Output is sent to file descriptor '1', i.e., standard output.
     */

    // for (line = 0; line < y_chars; line++) {
    //  compute_and_output_mandel_line(1, line);
    // }

    reset_xterm_color(1);
    return 0;
}
```

Έξοδος εκτέλεσης προγράμματος:



Ερωτήσεις:

1. Σε αυτήν την υλοποίηση δεν χρησιμοποιούμε κάποιο είδους συγχρονισμό με τους γνωστούς τρόπους όπως semaphores, mutexes ή σήματα. Ο μόνος συγχρονισμός που λαμβάνει μέρος είναι η αναμονή της διεργασίας πατέρας για τον τερματισμό όλων των διεργασιών παιδιών (μέσω της waitpid()). Οι διεργασίες παιδιά υπολογίζουν παράλληλα την έξοδο (nprocs γραμμές κάθε φορά, όπου nprocs ο αριθμός των διεργασιών) και τη γράφουν στο σημείο του buffer που αντιστοιχεί κάθε γραμμή. Όταν τελειώσουν οι υπολογισμοί, η διεργασία πατέρας αναλαμβάνει τη σειριακή εκτύπωση του μηνύματος στο τερματικό.

Σε περίπτωση που ο buffer είχε μέγεθος NPROCS x x_chars θα έπρεπε κάθε nprocs γραμμές που υπολόγιζαν οι διεργασίες παιδιά να υπήρχε συγχρονισμός (μέσω semaphores για παράδειγμα), ώστε το κρίσιμο σημείο του κώδικα, που είναι η εγγραφή στον buffer, να κλειδωνόταν μέχρι η διεργασία πατέρας να εκτυπώσει αυτές τις nprocs γραμμές και να ξεκλειδώσει το κρίσιμο σημείο για να συνεχίσουν οι διεργασίες παιδιά τον υπολογισμό.