Λειτουργικά Συστήματα Υπολογιστών

Αναφορά - 4η Εργαστηριακή Άσκηση

Ομάδα: oslab04

Κατσιάνης Κωνσταντίνος - 03120183

Κουρής Γεώργιος - 03120116

6ο Εξάμηνο - Ιούνιος 2023

1.1 Κλήσεις συστήματος και βασικοί μηχανισμοί του ΛΣ για τη διαχείριση της εικονικής μνήμης (Virtual Memory – VM)

Ο κώδικας μας είναι:

```
/*
 * 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;
char *file_shared_buf;
uint64_t buffer_size;



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

	/*
	 * Step 7 - Child
	 */
	if (0 != raise(SIGSTOP))
		die("raise(SIGSTOP)");

	printf("NM map of child process: \n");
	show_maps();

	/*
	 * Step 8 - Child
	 */
	if (0 != raise(SIGSTOP))
		die("raise(SIGSTOP)");

	printf("Physical Address of private buffer requested by child:
%ld\n",get_physical_address((uint64_t)heap_private_buf));

	printf("VA info of private buffer in child: ");
```

```
	show_va_info((uint64_t) heap_private_buf);


	/*
	 * Step 9 - Child
	 */
	if (0 != raise(SIGSTOP))
		die("raise(SIGSTOP)");


	int i;
	for (i=0; i<(int)buffer_size; i++) {
		heap_private_buf[i] = 0;
	}
	printf("VA info of private buffer in child: ");
	show_va_info((uint64_t)heap_private_buf);


	printf("Physical Address of private buffer requested by child:
%ld\n",get_physical_address((uint64_t)heap_private_buf));


	/*
	 * Step 10 - Child
	 */
	if (0 != raise(SIGSTOP))
		die("raise(SIGSTOP)");


	for (i=0; i<(int)buffer_size; i++) {
		heap_shared_buf[i] = 0;
	}


	printf("VA info of shared buffer in child: ");
	show_va_info((uint64_t)heap_shared_buf);


	printf("Physical Address of shared buffer requested by child:
%ld\n",get_physical_address((uint64_t)heap_shared_buf));
```

```
	/*
	 * Step 11 - Child
	 */
	if (0 != raise(SIGSTOP))
		die("raise(SIGSTOP)");

	mprotect(heap_shared_buf, buffer_size, PROT_READ);
	printf("VM map of child: \n");
	show_maps();
	show_va_info((uint64_t)heap_shared_buf);

	/*
	 * Step 12 - Child
	 */

	munmap(heap_shared_buf, buffer_size);
	munmap(heap_private_buf, buffer_size);
	munmap(file_shared_buf, buffer_size);
}

/*
 * 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();

	printf("VM map of parent process: \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 heap "
		"buffer (main) for both the parent and the child.\n" RESET);
	press_enter();

	printf("Physical Address of private buffer requested by parent:
%ld\n",get_physical_address((uint64_t)heap_private_buf));
	printf("VA info of private buffer in parent: ");
	show_va_info((uint64_t) heap_private_buf);

	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();


    printf("VA info of private buffer in parent: ");
    show_va_info((uint64_t) heap_private_buf);


    printf("Physical Address of private buffer requested by parent:
%ld\n",get_physical_address((uint64_t)heap_private_buf));


    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) from "
        "child and get the physical address for both the parent and "
        "the child. What happened?\n" RESET);
    press_enter();
```

```
	printf("VA info of shared buffer in parent: ");
	show_va_info((uint64_t) heap_shared_buf);

	printf("Physical Address of shared buffer requested by parent:
%ld\n",get_physical_address((uint64_t)heap_shared_buf));


	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 the "
		"child. Verify through the maps for the parent and the "
		"child.\n" RESET);
	press_enter();

	printf("VM map of parent");
	show_maps();
	show_va_info((uint64_t)heap_shared_buf);

	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
	 */

	munmap(heap_shared_buf, buffer_size);
	munmap(heap_private_buf, buffer_size);
	munmap(file_shared_buf, buffer_size);
}

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();

	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();

    heap_private_buf = mmap(NULL, buffer_size, PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS, fd, 0);
    if (heap_private_buf == MAP_FAILED) {
        die("mmap");
    }

    show_maps();
    show_va_info((uint64_t)heap_private_buf);


    /*
     * Step 3: Find the physical address of the first page of your buffer
     * in main memory. What do you see?
     */
    printf(RED "\nStep 3: Find and print the physical address of the "
        "buffer in main memory. What do you see?\n" RESET);
    press_enter();

    printf("Physical address: %ld\n", get_physical_address((uint64_t)heap_private_buf));

    /*
     * Step 4: Write zeros to the buffer and repeat Step 3.
     */
    printf(RED "\nStep 4: Initialize your buffer with zeros and repeat "
        "Step 3. What happened?\n" RESET);
    press_enter();
```

```
int i;
for (i=0; i<(int)buffer_size; i++) {
	heap_private_buf[i] = 0;
}

printf("Physical address: %ld\n", get_physical_address((uint64_t)heap_private_buf));

/*
 * 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. Print "
	"the new mapping information that has been created.\n" RESET);
press_enter();

fd = open("/home/oslab/oslab04/ex4/mmap/file.txt", O_RDONLY);
if (fd == -1) {
	die("open");
}

file_shared_buf = mmap(NULL, buffer_size, PROT_READ, MAP_SHARED, fd, 0);
if (file_shared_buf == MAP_FAILED) {
	die("mmap");
}

char c;
for (i=0; i<(int)buffer_size; i++) {
	c = file_shared_buf[i];
	if (c != EOF)
		putchar(c);
```

```
        else break;
    }

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



    /*
     * 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();

    heap_shared_buf = mmap(NULL, buffer_size, PROT_READ| PROT_WRITE,MAP_SHARED|
MAP_ANONYMOUS,-1,0);

    if (heap_shared_buf == MAP_FAILED) {
        die("mmap");
    }
    for (i=0; i<(int)buffer_size; i++) {
        heap_shared_buf[i] = i;
    }

    show_maps();
    show_va_info((uint64_t)heap_shared_buf);

    p = fork();
    if (p < 0)
        die("fork");
```

```
    if (p == 0) {
        child();
        return 0;
    }

    parent(p);

    if (-1 == close(fd))
        perror("close");
    return 0;
}
```

1) Αρχικά τυπώνουμε το χάρτη της εικονικής μνήμης της τρέχουσας διεργασίας.

```
Step 1: Print the virtual address space map of this process [166462].


Virtual Memory Map of process [166462]:
56220e57a000-56220e57b000 r--p 00000000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57b000-56220e57c000 r-xp 00001000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57c000-56220e57d000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57d000-56220e57e000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57e000-56220e57f000 rw-p 00003000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
5622104d2000-5622104f3000 rw-p 00000000 00:00 0                          [heap]
7f7517436000-7f7517458000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517458000-7f75175b1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f75175b1000-7f7517600000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517600000-7f7517604000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517604000-7f7517606000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517606000-7f751760c000 rw-p 00000000 00:00 0
7f7517611000-7f7517612000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517612000-7f7517632000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517632000-7f751763a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763b000-7f751763c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763c000-7f751763d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763d000-7f751763e000 rw-p 00000000 00:00 0
7ffc9614c000-7ffc9616d000 rw-p 00000000 00:00 0                          [stack]
7ffc961be000-7ffc961c2000 r--p 00000000 00:00 0                          [vvar]
7ffc961c2000-7ffc961c4000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------
```

2) Με την κλήση συστήματος mmap() δεσμεύουμε buffer (προσωρινή μνήμη) μεγέθους μίας σελίδας (page) και τυπώνουμε ξανά το χάρτη. Ο χώρος των εικονικών διευθύνσεων που δεσμεύσαμε φαίνεται στο κάτω μέρος της εικόνας.

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


Virtual Memory Map of process [166462]:
56220e57a000-56220e57b000 r--p 00000000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57b000-56220e57c000 r-xp 00001000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57c000-56220e57d000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57d000-56220e57e000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57e000-56220e57f000 rw-p 00003000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
5622104d2000-5622104f3000 rw-p 00000000 00:00 0                          [heap]
7f7517436000-7f7517458000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517458000-7f75175b1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f75175b1000-7f7517600000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517600000-7f7517604000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517604000-7f7517606000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517606000-7f751760c000 rw-p 00000000 00:00 0
7f7517611000-7f7517612000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517612000-7f7517632000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517632000-7f751763a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763a000-7f751763b000 rw-p 00000000 00:00 0
7f751763b000-7f751763c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763c000-7f751763d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763d000-7f751763e000 rw-p 00000000 00:00 0
7ffc9614c000-7ffc9616d000 rw-p 00000000 00:00 0                          [stack]
7ffc961be000-7ffc961c2000 r--p 00000000 00:00 0                          [vvar]
7ffc961c2000-7ffc961c4000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------

7f751763a000-7f751763b000 rw-p 00000000 00:00 0
```

3) Προσπαθούμε να βρούμε και να τυπώσουμε τη φυσική διεύθυνση μνήμης στην οποία απεικονίζεται η εικονική διεύθυνση του buffer.

Παρατηρούμε ότι η φυσική μνήμη είναι 0. Αυτό σημαίνει ότι δεν έχει γίνει ακόμα η απεικόνιση της εικονικής μνήμης σε φυσική, παρά το ότι έχει δεσμευτεί η μνήμη (βλ. χάρτη μνήμης).

Αυτό συμβαίνει λόγω του ότι η δέσμευση της φυσικής μνήμης γίνεται κατευθείαν, κατά την προσπάθεια προσπέλασης από το OS. Έτσι, όταν προσπαθούμε να προσπελάσουμε εικονική μνήμη που δεν έχει απεικονιστεί, θα έχουμε σφάλμα.

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

VA[0x7f751763a000] is not mapped; no physical memory allocated.
Physical address: 0
```

4) Γεμίσαμε τον buffer με μηδενικά και επαναλάβαμε το βήμα 3. Παρατηρούμε ότι τώρα γίνεται απεικόνιση στην φυσική μνήμη σε αντίθεση με πριν.

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

Physical address: 5907480576
```

5) Χρησιμοποιούμε την mmap() για να απεικονίσουμε (memory map) το αρχείο file.txt στον χώρο διευθύνσεων της διεργασίας σας και να τυπώσουμε το περιεχόμενό του. Στην συνέχεια εντοπίζουμε την νέα απεικόνιση (mapping) στο χάρτη μνήμης.

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

Hello Earth!

Virtual Memory Map of process [166462]:
56220e57a000-56220e57b000 r--p 00000000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57b000-56220e57c000 r-xp 00001000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57c000-56220e57d000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57d000-56220e57e000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57e000-56220e57f000 rw-p 00003000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
5622104d2000-5622104f3000 rw-p 00000000 00:00 0                          [heap]
7f7517436000-7f7517458000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517458000-7f75175b1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f75175b1000-7f7517600000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517600000-7f7517604000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517604000-7f7517606000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517606000-7f751760c000 rw-p 00000000 00:00 0
7f7517610000-7f7517611000 r--s 00000000 00:26 7619974                    /home/oslab/oslab04/ex4/mmap/file.txt
7f7517611000-7f7517612000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517612000-7f7517632000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517632000-7f751763a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763a000-7f751763b000 rw-p 00000000 00:00 0
7f751763b000-7f751763c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763c000-7f751763d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763d000-7f751763e000 rw-p 00000000 00:00 0
7ffc9614c000-7ffc9616d000 rw-p 00000000 00:00 0                          [stack]
7ffc961be000-7ffc961c2000 r--p 00000000 00:00 0                          [vvar]
7ffc961c2000-7ffc961c4000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------------

7f7517610000-7f7517611000 r--s 00000000 00:26 7619974                    /home/oslab/oslab04/ex4/mmap/file.txt
```

6) Δεσμεύουμε νέο buffer, διαμοιραζόμενο (shared) αυτή τη φορά μεταξύ διεργασιών, μεγέθους μιας σελίδας (page) με την χρήση της mmap() και τυπώνουμε ξανά το χάρτη.

```
Step 6: Use mmap(2) to allocate a shared buffer of size equal to 1 page. Initialize the buffer and print the new


Virtual Memory Map of process [166462]:
56220e57a000-56220e57b000 r--p 00000000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57b000-56220e57c000 r-xp 00001000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57c000-56220e57d000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57d000-56220e57e000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57e000-56220e57f000 rw-p 00003000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
5622104d2000-5622104f3000 rw-p 00000000 00:00 0                          [heap]
7f7517436000-7f7517458000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517458000-7f75175b1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f75175b1000-7f7517600000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517600000-7f7517604000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517604000-7f7517606000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517606000-7f751760c000 rw-p 00000000 00:00 0
7f751760f000-7f7517610000 rw-s 00000000 00:01 335                        /dev/zero (deleted)
7f7517610000-7f7517611000 r--s 00000000 00:26 7619974                    /home/oslab/oslab04/ex4/mmap/file.txt
7f7517611000-7f7517612000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517612000-7f7517632000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517632000-7f751763a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763a000-7f751763b000 rw-p 00000000 00:00 0
7f751763b000-7f751763c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763c000-7f751763d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763d000-7f751763e000 rw-p 00000000 00:00 0
7ffc9614c000-7ffc9616d000 rw-p 00000000 00:00 0                          [stack]
7ffc961be000-7ffc961c2000 r--p 00000000 00:00 0                          [vvar]
7ffc961c2000-7ffc961c4000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------------

7f751760f000-7f7517610000 rw-s 00000000 00:01 335                        /dev/zero (deleted)
```

7) Τυπώνουμε τον χάρτη της εικονικής μνήμης της διεργασίας πατέρα και της διεργασίας παιδιού.

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

VM map of parent process:

Virtual Memory Map of process [166462]:
56220e57a000-56220e57b000 r--p 00000000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57b000-56220e57c000 r-xp 00001000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57c000-56220e57d000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57d000-56220e57e000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57e000-56220e57f000 rw-p 00003000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
5622104d2000-5622104f3000 rw-p 00000000 00:00 0                          [heap]
7f7517436000-7f7517458000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517458000-7f75175b1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f75175b1000-7f7517600000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517600000-7f7517604000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517604000-7f7517606000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517606000-7f751760c000 rw-p 00000000 00:00 0
7f751760f000-7f7517610000 rw-s 00000000 00:01 335                        /dev/zero (deleted)
7f7517610000-7f7517611000 r--s 00000000 00:26 7619974                    /home/oslab/oslab04/ex4/mmap/file.txt
7f7517611000-7f7517612000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517612000-7f7517632000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517632000-7f751763a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763a000-7f751763b000 rw-p 00000000 00:00 0
7f751763b000-7f751763c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763c000-7f751763d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763d000-7f751763e000 rw-p 00000000 00:00 0
7ffc9614c000-7ffc9616d000 rw-p 00000000 00:00 0                          [stack]
7ffc961be000-7ffc961c2000 r--p 00000000 00:00 0                          [vvar]
7ffc961c2000-7ffc961c4000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------------
```

```
NM map of child process:

Virtual Memory Map of process [166463]:
56220e57a000-56220e57b000 r--p 00000000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57b000-56220e57c000 r-xp 00001000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57c000-56220e57d000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57d000-56220e57e000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57e000-56220e57f000 rw-p 00003000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
5622104d2000-5622104f3000 rw-p 00000000 00:00 0                          [heap]
7f7517436000-7f7517458000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517458000-7f75175b1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f75175b1000-7f7517600000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517600000-7f7517604000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517604000-7f7517606000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517606000-7f751760c000 rw-p 00000000 00:00 0
7f751760f000-7f7517610000 rw-s 00000000 00:01 335                        /dev/zero (deleted)
7f7517610000-7f7517611000 r--s 00000000 00:26 7619974                    /home/oslab/oslab04/ex4/mmap/file.txt
7f7517611000-7f7517612000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517612000-7f7517632000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517632000-7f751763a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763a000-7f751763b000 rw-p 00000000 00:00 0
7f751763b000-7f751763c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763c000-7f751763d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763d000-7f751763e000 rw-p 00000000 00:00 0
7ffc9614c000-7ffc9616d000 rw-p 00000000 00:00 0                          [stack]
7ffc961be000-7ffc961c2000 r--p 00000000 00:00 0                          [vvar]
7ffc961c2000-7ffc961c4000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------------
```

Παρατηρούμε ότι η νέα διεργασία child που δημιουργείται μέσω της fork() είναι αντίγραφο της παλιάς. Κληρονομεί όμως και ένα αντίγραφο της εικονικής μνήμης της αρχικής διεργασίας, καθώς και ένα αντίγραφο του πίνακα σελίδων (page table) της αρχικής διεργασίας, όπου αφαιρούνται και από τους δύο τα write δικαιώματα στις σελίδες που είναι private-COW.

8) Βρίσκουμε και τυπώνουμε τη φυσική διεύθυνση στη κύρια μνήμη του private buffer για τις διεργασίες πατέρα και παιδί. Παρατηρούμε ότι απεικονίζονται στην ίδια φυσική μνήμη.

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

Physical Address of private buffer requested by parent: 5907480576
VA info of private buffer in parent: 7f751763a000-7f751763b000 rw-p 00000000 00:00 0
Physical Address of private buffer requested by child: 5907480576
VA info of private buffer in child: 7f751763a000-7f751763b000 rw-p 00000000 00:00 0
```

9) Γράφουμε στον private buffer από τη διεργασία παιδί και επαναλαμβάνουμε το προηγούμενο βήμα. Παρατηρούμε ότι οι φυσικές διευθύνσεις είναι διαφορετικές σε αυτήν την περίπτωση. Αυτό συμβαίνει η εικονική μνήμη και ο πίνακας σελίδων αντιγράφονται. Η απεικόνιση της εικονικής διεύθυνσης δεν θα αλλάξει, μέχρι κάποια από τις διεργασίες επιχειρήσει να γράψει σε κάποια σελίδα, χωρίς δικαιώματα write. Το frame που θα βρεθεί τότε για την απεικόνιση θα είναι διαφορετικό και το περιεχόμενο της page θα αντιγραφεί σε νέα φυσική διεύθυνση.

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

VA info of private buffer in parent: 7f751763a000-7f751763b000 rw-p 00000000 00:00 0
Physical Address of private buffer requested by parent: 5907480576
VA info of private buffer in child: 7f751763a000-7f751763b000 rw-p 00000000 00:00 0
Physical Address of private buffer requested by child: 5580472320
```

10) Γράφουμε στον shared buffer (Βήμα 6) από τη διεργασία παιδί και τυπώνουμε τη φυσική του διεύθυνση για τις διεργασίες πατέρα και παιδί.

```
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?

VA info of shared buffer in parent: 7f751760f000-7f7517610000 rw-s 00000000 00:01 335                      /dev/zero (deleted)
Physical Address of shared buffer requested by parent: 5907505152
VA info of shared buffer in child: 7f751760f000-7f7517610000 rw-s 00000000 00:01 335                     /dev/zero (deleted)
Physical Address of shared buffer requested by child: 5907505152
```

Παρατηρούμε τώρα ότι ο shared buffer απεικονίζεται στην ίδια φυσική διεύθυνση και για τις δύο διεργασίες. Αυτό συμβαίνει διότι η σελίδα τώρα είναι διαμοιρασμένη μεταξύ των δύο διεργασιών.

11) Απαγορεύουμε τις εγγραφές στον shared buffer για τη διεργασία παιδί χρησιμοποιώντας την mprotect(). Παρατηρούμε ότι έχει αφαιρεθεί το δικαίωμα write από την διεργασία παιδί.

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

VM map of parent
Virtual Memory Map of process [166462]:
56220e57a000-56220e57b000 r--p 00000000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57b000-56220e57c000 r-xp 00001000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57c000-56220e57d000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57d000-56220e57e000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57e000-56220e57f000 rw-p 00003000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
5622104d2000-5622104f3000 rw-p 00000000 00:00 0                          [heap]
7f7517436000-7f7517458000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517458000-7f75175b1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f75175b1000-7f7517600000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517600000-7f7517604000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517604000-7f7517606000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517606000-7f751760c000 rw-p 00000000 00:00 0
7f751760f000-7f7517610000 rw-s 00000000 00:01 335                        /dev/zero (deleted)
7f7517610000-7f7517611000 r--s 00000000 00:26 7619974                    /home/oslab/oslab04/ex4/mmap/file.txt
7f7517611000-7f7517612000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517612000-7f7517632000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517632000-7f751763a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763a000-7f751763b000 rw-p 00000000 00:00 0
7f751763b000-7f751763c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763c000-7f751763d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763d000-7f751763e000 rw-p 00000000 00:00 0
7ffc9614c000-7ffc9616d000 rw-p 00000000 00:00 0                          [stack]
7ffc961be000-7ffc961c2000 r--p 00000000 00:00 0                          [vvar]
7ffc961c2000-7ffc961c4000 r-xp 00000000 00:00 0                          [vdso]
-------------------------------------------------------

7f751760f000-7f7517610000 rw-s 00000000 00:01 335                        /dev/zero (deleted)
```

```
VM map of child:

Virtual Memory Map of process [166463]:
56220e57a000-56220e57b000 r--p 00000000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57b000-56220e57c000 r-xp 00001000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57c000-56220e57d000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57d000-56220e57e000 r--p 00002000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
56220e57e000-56220e57f000 rw-p 00003000 00:26 7619936                    /home/oslab/oslab04/ex4/mmap/mmap
5622104d2000-5622104f3000 rw-p 00000000 00:00 0                          [heap]
7f7517436000-7f7517458000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517458000-7f75175b1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f75175b1000-7f7517600000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517600000-7f7517604000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517604000-7f7517606000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7517606000-7f751760c000 rw-p 00000000 00:00 0
7f751760f000-7f7517610000 r--s 00000000 00:01 335                        /dev/zero (deleted)
7f7517610000-7f7517611000 r--s 00000000 00:26 7619974                    /home/oslab/oslab04/ex4/mmap/file.txt
7f7517611000-7f7517612000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517612000-7f7517632000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7517632000-7f751763a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763a000-7f751763b000 rw-p 00000000 00:00 0
7f751763b000-7f751763c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763c000-7f751763d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f751763d000-7f751763e000 rw-p 00000000 00:00 0
7ffc9614c000-7ffc9616d000 rw-p 00000000 00:00 0                          [stack]
7ffc961be000-7ffc961c2000 r--p 00000000 00:00 0                          [vvar]
7ffc961c2000-7ffc961c4000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------

7f751760f000-7f7517610000 r--s 00000000 00:01 335                        /dev/zero (deleted)
```

12) Αποδεσμεύουμε όλους τους buffers στις δύο διεργασίες με χρήση της munmap().

1.2 Παράλληλος υπολογισμός Mandelbrot με διεργασίες αντί για νήματα

1.2.1 Semaphores πάνω από διαμοιραζόμενη μνήμη

```
#include <stdint.h>
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <signal.h>
#include <errno.h>
#include "mandel-lib.h"
#include <sys/mman.h>
#include <sys/wait.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, ymin)
*/
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;

sem_t *semaphore;
int num_threads;

/* case: usage of ctrl C*/
void sigint_handler(int signum) {
	reset_xterm_color(1);
	exit(1);
}

int safe_atoi(char *s, int *value) {
	long l;
	char *endp;
	l = strtol(s, &endp, 10);
	if(s != endp && *endp == '\0') {
		*value=l;
		return 0;
	} else
		return -1;
}
```

```
void *safe_malloc(size_t size)
{
	void *p;
	if((p=malloc(size))==NULL) {
		fprintf(stderr, "Out of memory, failed to allocate %zd bytes\n",
			size);
		exit(1);
	}
	return p;
}

/*
 * 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;
        }
    }
    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);
        }
    }
```

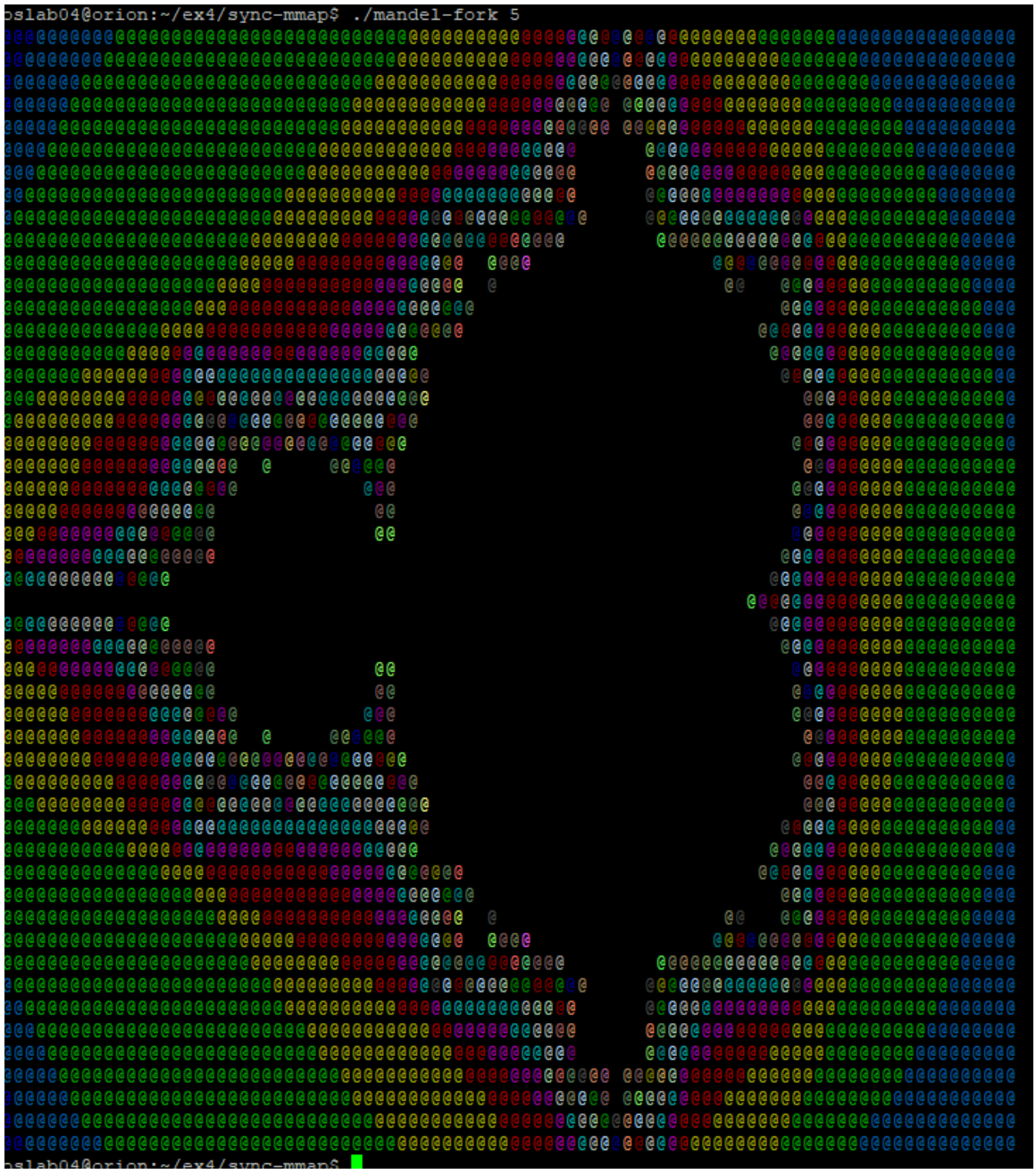
```
void *create_shared_memory_area(unsigned int numbytes)
{
	int pages;
	void *addr;
	if (numbytes == 0) {
		fprintf(stderr, "%s: internal error: called for numbytes == 0\n", __func__);
		 exit(1);
	}
	/*Determine the number of pages needed, round up the requested number of pages*/
	pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;
	/* Create a shared, anonymous mapping for this number of pages */
	/* TODO:*/
	addr = mmap(NULL, pages * sysconf(_SC_PAGE_SIZE), PROT_READ | PROT_WRITE, MAP_SHARED
| MAP_ANONYMOUS, -1,0);
	if(addr==MAP_FAILED) {
		perror("mmap");
		exit(1);
	}
	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 number 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);
```

```
        }
    }
    void fork_f(int x, int num_threads) {
        int num;
        int color_value[x_chars];
        for (num=x; num<y_chars; num+=num_threads) {
            compute_mandel_line(num, color_value);
            if(sem_wait(&semaphore[x])<0) {
                perror("semaphore wait error");
                exit(1);
            }
            output_mandel_line(1, color_value);
             if(sem_post(&semaphore[(num+1) % num_threads])<0) {
                 perror("sem_post");
                 exit(1);
             }
        }
    }
    int main(int argc, char **argv)
    {
        int i, wait_status;
        sigset_t sigset;
        pid_t pid;
        xstep = (xmax - xmin) / x_chars;
        ystep = (ymax - ymin) / y_chars;
        if (argc != 2) {
            perror("Incorrect input, please insert only the number of threadsto create");
            exit(1);
        }
        if (safe_atoi(argv[1], &num_threads) < 0 || num_threads <= 0) {
            perror("input error");
            exit(1);
```

```
}
struct sigaction sa;
sa.sa_handler = sigint_handler;
sa.sa_flags = 0;
sigemptyset(&sigset);
sa.sa_mask = sigset;
if (sigaction(SIGINT, &sa, NULL) < 0) {
      perror("sigaction");
      exit(1);
}
semaphore= create_shared_memory_area(num_threads * sizeof(sem_t));
for (i = 0; i < num_threads; i++) {
      if (sem_init(&semaphore[i], 1, 0) < 0) {
            perror("sem_init error");
            exit(1);
      }
}
if(sem_post(&semaphore[0])<0) {
      perror("sem_post");
      exit(1);
}




for (i = 0; i < num_threads; i++) {
      pid=fork();
      if (pid<0) {
            perror("fork error");
            exit(1);
      }
      else if (pid == 0) {
            fork_f(i,num_threads);
```

```
                exit(1);
            }
        }
        for (i = 0; i < num_threads; i++) {
            pid=wait(&wait_status);
            if (pid<0) {
                perror("wait error");
            }
        }
    for (i = 0; i < num_threads; i++) {
            sem_destroy(&semaphore[i]);
        }
        destroy_shared_memory_area(semaphore, num_threads * sizeof(sem_t));
        reset_xterm_color(1);
        return 0;
}
```

Η έξοδος του προγράμματος για 5 threads:



1) Ο συγχρονισμός μεταξύ των διεργασιών γίνεται και πάλι με semaphores, που παρέχονται από το πρότυπο POSIX. Σε αντίθεση με την άσκηση 3, δε χρησιμοποιούμε threads, αλλά processes για την παραλληλοποίηση του υπολογισμού του set. Γενικά, αναμένουμε καλύτερη επίδοση από τα threads, καθώς είναι πιο lightweight, από τις διεργασίες, οι οποίες προτιμώνται σε πιο heavyweight tasks. Ακόμη, η εναλλαγή νημάτων είναι πιο γρήγορο από την εναλλαγή των διαδικασιών, ενώ επίσης τα νήματα διαθέτουν κοινή μνήμη, κάτι που

δεν ισχύει με τις διαδικασίες. Τα semaphores βρίσκονται σε διαμοιραζόμενη μνήμη μεταξύ διεργασιών, οπότε χρησιμοποιούμε τις βοηθητικές συναρτήσεις, γεγονός που επιβαρύνει χρονικά τη χρήση τους έναντι των νημάτων.

1.2.2 Υλοποίηση χωρίς semaphores

```
#include <stdint.h>
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <signal.h>
#include <errno.h>
#include "mandel-lib.h"
#include <sys/mman.h>
#include <sys/wait.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, ymin)
*/
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;

sem_t *semaphore;
int num_threads;

/* case: usage of ctrl C*/
void sigint_handler(int signum) {
	reset_xterm_color(1);
	exit(1);
}

int safe_atoi(char *s, int *value) {
	long l;
	char *endp;
	l = strtol(s, &endp, 10);
	if(s != endp && *endp == '\0') {
		*value=l;
		return 0;
	} else
		return -1;
}
```

```
void *safe_malloc(size_t size)
{
	void *p;
	if((p=malloc(size))==NULL) {
		fprintf(stderr, "Out of memory, failed to allocate %zd bytes\n",
			size);
		exit(1);
	}
	return p;
}

/*
 * 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;
        }
    }
    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);
        }
    }
```

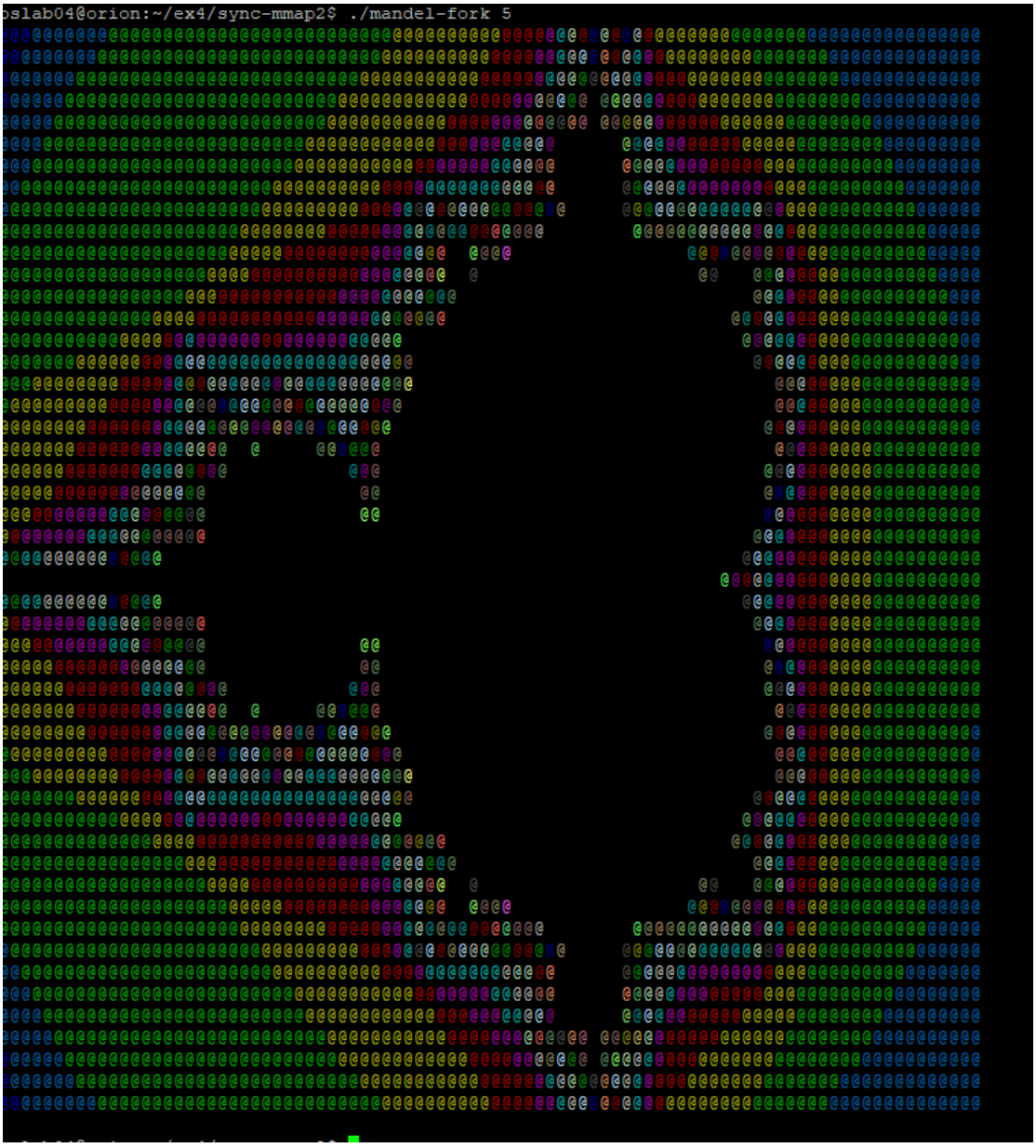
```
int **buffer;
void *create_shared_memory_area(unsigned int numbytes)

{
	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 number of pages*/
	pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;
	/* Create a shared, anonymous mapping for this number of pages */
	/* TODO:*/
	addr = mmap(NULL, pages * sysconf(_SC_PAGE_SIZE), PROT_READ | PROT_WRITE, MAP_SHARED
| MAP_ANONYMOUS, -1,0);
	if(addr==MAP_FAILED) {
		perror("mmap");
		exit(1);
	}
	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 number 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);
        }
}
void fork_f(int x, int num_threads) {
        int num;
        //int color_value[x_chars];
        for (num=x; num<y_chars; num+=num_threads) {
                compute_mandel_line(num, buffer[num]);
        }
        return;
}


int main(int argc, char **argv)
{
        int i, wait_status;
        sigset_t sigset;
        pid_t pid;
        xstep = (xmax - xmin) / x_chars;
        ystep = (ymax - ymin) / y_chars;
        if (argc != 2) {
                perror("Incorrect input, please insert only the number of threadsto create");
                exit(1);
        }
        if (safe_atoi(argv[1], &num_threads) < 0 || num_threads <= 0) {
                perror("input error");
                exit(1);
        }
```

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

struct sigaction sa;
sa.sa_handler = sigint_handler;
sa.sa_flags = 0;
sigemptyset(&sigset);
sa.sa_mask = sigset;

if (sigaction(SIGINT, &sa, NULL) < 0) {
      perror("sigaction");
      exit(1);
}

 buffer = create_shared_memory_area(y_chars * sizeof(int));

for (i = 0; i <= y_chars; i++) {
      buffer[i]=create_shared_memory_area(x_chars * sizeof(char));
}
for (i = 0; i < num_threads; i++) {

      pid=fork();

      if (pid<0) {
            perror("fork error");
            exit(1);
      }
      else if (pid == 0) {
            fork_f(i,num_threads);
```

```
                exit(1);
            }
        }

        for (i = 0; i < num_threads; i++) {

            pid=wait(&wait_status);

            if (pid<0) {

                perror("wait error");
            }
        }

        for (i = 0; i <= y_chars; i++) {
            output_mandel_line(1, buffer[i]);
        }

        for (i = 0; i <= y_chars; i++) {
            destroy_shared_memory_area(buffer[i], sizeof(buffer[i]));

        }

        destroy_shared_memory_area(buffer, sizeof(buffer));
        reset_xterm_color(1);
        return 0;
}
```

Εκτέλεση για 5 νήματα:



1) Σε αυτό το ερώτημα, δημιουργήσαμε ένα 2d buffer που θα κρατήσει ό,τι υπολογίζει η κάθε διεργασία. Το output γίνεται στο τέλος, για αυτό και καλούμε την αντίστοιχη συνάρτηση στην int man(). Έτσι, με τη χρήση του buffer, ο συγχρονισμός σε αυτήν την υλοποίηση επιτυγχάνεται καθώς η κάθε διεργασία αποθηκεύει τους υπολογισμούς στο τμήμα του buffer που της αντιστοιχεί. Έτσι, το output δημιουργείται παράλληλα.

Αν ο buffer είχε διαστάσεις NPROCS * x_chars, τότε θα ήταν αδύνατο να γίνει το output όλο μαζί, όπως κάνουμε τώρα και θα έπρεπε να γίνει σε τμήματα.