Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Εξάμηνο 6ο

Μάθημα: Λειτουργικά Συστήματα

Μέλη Ομάδας: Γεωργία Μπουσμπουκέα 03119059
Παναγιώτης Μιχελάκης 03119119

Έτος: 2021 - 2022

4η Εργαστηριακή Άσκηση Αναφορά

Άσκηση 1.1

Παρακάτω παραθέτουμε τον πηγαίο κώδικα της άσκησης 1.1:

```
#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;
```

```
uint64_t get_file_size(const char* pathname)
{
	struct stat fileinfo;
	if (stat(pathname, &fileinfo) == -1) {
		perror("stat");
		exit(1);
	}
	return fileinfo.st_size;
}


void print_physical_address(char* variable, const char called_from[])
{
	uint64_t pa = get_physical_address((uint64_t) variable);
	printf("Physical address (%s): %lld\n", called_from, (long long int) pa);
}

// 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
Mapping\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.
	print_physical_address(heap_private_buf, "child"); // The moment we call the
fork system call, the child inherits the addresses of the parent as is but the moment it
goes on to write something
	// they get copied to new ones (at least the MEM_PRIVATE ones)



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

```
    // TODO: Write your code here to complete child's part of Step 9.

    strcpy(heap_private_buf, "Hi");
    print_physical_address(heap_private_buf, "child"); //because the buffer was marked as private, the memory address changed when it was written inside the child process



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

    // TODO: Write your code here to complete child's part of Step 10.
    strcpy(heap_shared_buf, "Sup");
    print_physical_address(heap_shared_buf, "child");// the buffer was marked as MAP_SHARED so the address does not get changed



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

    // TODO: Write your code here to complete child's part of Step 11.
    mprotect(heap_shared_buf, buffer_size, PROT_READ);
    show_va_info((uint64_t) heap_shared_buf);


    // Step 12 - Child
    // TODO: Write your code here to complete child's part of Step 12.
    munmap(heap_private_buf, buffer_size);
    munmap(heap_shared_buf, buffer_size);

munmap(file_shared_buf, get_file_size("file.txt"));
}

// 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 Mapping\n");
    show_maps(); // fork system call also copies the page table



    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();

    // TODO: Write your code here to complete parent's part of Step 8.
    print_physical_address(heap_private_buf, "parent");



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



        // 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.
    print_physical_address(heap_private_buf, "parent");



    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();

        // TODO: Write your code here to complete parent's part of Step 10.
        print_physical_address(heap_shared_buf, "parent");



        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();

        // TODO: Write your code here to complete parent's part of Step 11.
        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
        // TODO: Write your code here to complete parent's part of Step 12.
        munmap(heap_private_buf, buffer_size);
        munmap(heap_shared_buf, buffer_size);
        munmap(file_shared_buf, get_file_size("file.txt"));
}

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_ANONYMOUS | MAP_PRIVATE, -1, 0);//map anonymous fills the buffer with
zeros but since the user did not assign it, it do
	//es not map it to physical memory
	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();

	// TODO: Write your code here to complete Step 3.
	print_physical_address(heap_private_buf, "main");

//printf(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();
```

```
    // TODO: Write your code here to complete Step 4.
    memset(heap_private_buf, 0, buffer_size); // we manually fill the buffer with 0s,
thus it gets mapped to physical memory
    print_physical_address(heap_private_buf, "main");



    // 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();

    // TODO: Write your code here to complete Step 5.
    fd = open("file.txt", O_RDONLY);
    file_shared_buf = mmap(NULL, get_file_size("file.txt"), PROT_READ,
MAP_PRIVATE, fd, 0);
printf(file_shared_buf);
    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();

    // TODO: Write your code here to complete Step 6.
    heap_shared_buf = mmap(NULL, buffer_size, PROT_READ | PROT_WRITE,
MAP_ANONYMOUS | MAP_SHARED, -1, 0);
    memset(heap_shared_buf, 0, buffer_size);
    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. Τυπώνουμε τον χάρτη της εικονικής μνήμης της τρέχουσας διεργασίας με τη συνάρτηση show_maps().

```
Virtual Memory Map of process [16432]:
00400000-00403000 r-xp 00000000 fe:10 9712462                            /store/homes/oslab/oslaba59/seira4/ask1/mmap
00602000-00603000 rw-p 00002000 fe:10 9712462                            /store/homes/oslab/oslaba59/seira4/ask1/mmap
0222e000-0224f000 rw-p 00000000 00:00 0                                  [heap]
7f87a90c2000-7f87a9263000 r-xp 00000000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f87a9263000-7f87a9463000 ---p 001a1000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f87a9463000-7f87a9467000 r--p 001a1000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f87a9467000-7f87a9469000 rw-p 001a5000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f87a9469000-7f87a946d000 rw-p 00000000 00:00 0
7f87a946d000-7f87a948d000 r-xp 00000000 fe:01 1044705                    /lib/x86_64-linux-gnu/ld-2.19.so
7f87a9680000-7f87a9683000 rw-p 00000000 00:00 0
7f87a9688000-7f87a968d000 rw-p 00000000 00:00 0
7f87a968d000-7f87a968e000 r--p 00020000 fe:01 1044705                    /lib/x86_64-linux-gnu/ld-2.19.so
7f87a968e000-7f87a968f000 rw-p 00021000 fe:01 1044705                    /lib/x86_64-linux-gnu/ld-2.19.so
7f87a968f000-7f87a9690000 rw-p 00000000 00:00 0
7ffffa6d7000-7ffffa6f8000 rw-p 00000000 00:00 0                          [stack]
7ffffa744000-7ffffa746000 r-xp 00000000 00:00 0                          [vdso]
7ffffa746000-7ffffa748000 r--p 00000000 00:00 0                          [vvar]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

2. Δεσμεύουμε μνήμη με τη κλήση συστήματος mmap() μέγεθους μίας σελίδας. Τυπώνουμε ξανά το χάρτη μνήμης, εντοπίζουμε τη διεύθυνση που δεσμεύθηκε και την τυπώνουμε ξεχωριστά από κάτω

```
Virtual Memory Map of process [16432]:
00400000-00403000 r-xp 00000000 fe:10 9712462                            /store/homes/oslab/oslaba59/seira4/ask1/mmap
00602000-00603000 rw-p 00002000 fe:10 9712462                            /store/homes/oslab/oslaba59/seira4/ask1/mmap
0222e000-0224f000 rw-p 00000000 00:00 0                                  [heap]
7f87a90c2000-7f87a9263000 r-xp 00000000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f87a9263000-7f87a9463000 ---p 001a1000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f87a9463000-7f87a9467000 r--p 001a1000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f87a9467000-7f87a9469000 rw-p 001a5000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f87a9469000-7f87a946d000 rw-p 00000000 00:00 0
7f87a946d000-7f87a948d000 r-xp 00000000 fe:01 1044705                    /lib/x86_64-linux-gnu/ld-2.19.so
7f87a9680000-7f87a9683000 rw-p 00000000 00:00 0
7f87a9687000-7f87a968d000 rw-p 00000000 00:00 0
7f87a968d000-7f87a968e000 r--p 00020000 fe:01 1044705                    /lib/x86_64-linux-gnu/ld-2.19.so
7f87a968e000-7f87a968f000 rw-p 00021000 fe:01 1044705                    /lib/x86_64-linux-gnu/ld-2.19.so
7f87a968f000-7f87a9690000 rw-p 00000000 00:00 0
7ffffa6d7000-7ffffa6f8000 rw-p 00000000 00:00 0                          [stack]
7ffffa744000-7ffffa746000 r-xp 00000000 00:00 0                          [vdso]
7ffffa746000-7ffffa748000 r--p 00000000 00:00 0                          [vvar]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------

7f87a9687000-7f87a968d000 rw-p 00000000 00:00 0
```

3. Χρησιμοποιούμε τη συνάρτηση get_physical_address() για να τυπώσουμε τη φυσική διεύθυνση του buffer αλλά βλέπουμε πως αυτή δεν έχει καταχωρηθεί.

```
VA[0x7fbe093de000] is not mapped; no physical memory allocated.
Physical address (main): 0
```

Αυτό συμβαίνει λόγω της αρχής του Demand Paging την οποία αναλύσαμε στη θεωρία.

4. Γεμίζουμε με μηδενικά τον πίνακα και καλούμε ξανά τη συνάρτηση του προηούμενου ερωτήματος.

```
Physical address (main): 2872442880
```

Εφόσον τώρα επεξεργαστήκαμε τον πίνακα, το λειτουργικό φρόντισε να αντιστοιχίσει την εικονική μνήμη που του είχε δοθεί σε φυσική.

5. Χρησιμοποιούμε την κλήση συστήματος mmap() για να απεικονίουμε (memory map) το αρχείο file.txt στον χώρο διευθύνσεων της διεργασίας μας και να τυπώσουμε το περιεχόμενό του.

```
Hello everyone!

Virtual Memory Map of process [18117]:
00400000-00403000 r-xp 00000000 00:21 9712462                            /home/oslab/oslaba59/seira4/ask1/mmap
00602000-00603000 rw-p 00002000 00:21 9712462                            /home/oslab/oslaba59/seira4/ask1/mmap
00861000-00882000 rw-p 00000000 00:00 0                                  [heap]
7fbe08e18000-7fbe08fb9000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe08fb9000-7fbe091b9000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe091b9000-7fbe091bd000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe091bd000-7fbe091bf000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe091bf000-7fbe091c3000 rw-p 00000000 00:00 0
7fbe091c3000-7fbe091e4000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fbe093d6000-7fbe093d9000 rw-p 00000000 00:00 0
7fbe093dc000-7fbe093dd000 rw-p 00000000 00:00 0
7fbe093dd000-7fbe093de000 r--p 00000000 00:21 9712535                    /home/oslab/oslaba59/seira4/ask1/file.txt
7fbe093de000-7fbe093e3000 rw-p 00000000 00:00 0
7fbe093e3000-7fbe093e4000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fbe093e4000-7fbe093e5000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fbe093e5000-7fbe093e6000 rw-p 00000000 00:00 0
7fffa5ad7000-7fffa5af8000 rw-p 00000000 00:00 0                          [stack]
7fffa5b9e000-7fffa5ba1000 r--p 00000000 00:00 0                          [vvar]
7fffa5ba1000-7fffa5ba3000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------

7fbe093dd000-7fbe093de000 r--p 00000000 00:21 9712535                    /home/oslab/oslaba59/seira4/ask1/file.txt
```

Ο χώρος που δεσμεύτηκε για το αρχείο αυτό φαίνεται στην τελευταία γραμμή της εξόδου.

6. Χρησιμοποιούμε την mmap() για να δεσμεύσουμε έναν νέο buffer, διαμοιραζόμενο (shared) αυτή τη φορά μεταξύ διεργασιών με μέγεθος μια σελίδας. Εντοπίζουμε τη νέα απεικόνιση (mapping) στο χάρτη μνήμης.

```
Virtual Memory Map of process [18117]:
00400000-00403000 r-xp 00000000 00:21 9712462                            /home/oslab/oslaba59/seira4/ask1/mmap
00602000-00603000 rw-p 00002000 00:21 9712462                            /home/oslab/oslaba59/seira4/ask1/mmap
00861000-00882000 rw-p 00000000 00:00 0                                  [heap]
7fbe08e18000-7fbe08fb9000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe08fb9000-7fbe091b9000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe091b9000-7fbe091bd000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe091bd000-7fbe091bf000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe091bf000-7fbe091c3000 rw-p 00000000 00:00 0
7fbe091c3000-7fbe091e4000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fbe093d6000-7fbe093d9000 rw-p 00000000 00:00 0
7fbe093db000-7fbe093dc000 rw-p 00000000 00:00 0
7fbe093dc000-7fbe093dd000 rw-s 00000000 00:04 3995050                    /dev/zero (deleted)
7fbe093dd000-7fbe093de000 r--p 00000000 00:21 9712535                    /home/oslab/oslaba59/seira4/ask1/file.txt
7fbe093de000-7fbe093e3000 rw-p 00000000 00:00 0
7fbe093e3000-7fbe093e4000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fbe093e4000-7fbe093e5000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fbe093e5000-7fbe093e6000 rw-p 00000000 00:00 0
7fffa5ad7000-7fffa5af8000 rw-p 00000000 00:00 0                          [stack]
7fffa5b9e000-7fffa5ba1000 r--p 00000000 00:00 0                          [vvar]
7fffa5ba1000-7fffa5ba3000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------

7fbe093dc000-7fbe093dd000 rw-s 00000000 00:04 3995050                    /dev/zero (deleted)
```

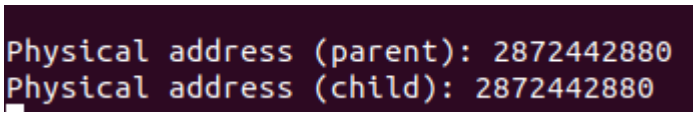
7. Καλούμε τη κλήση συστήματος fork() και δημιουργούμε μία νέα διεργασία.

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

```
Parent's Virtual Memory Mapping

Virtual Memory Map of process [18117]:
00400000-00403000 r-xp 00000000 00:21 9712462                            /home/oslab/oslaba59/seira4/ask1/mmap
00602000-00603000 rw-p 00002000 00:21 9712462                            /home/oslab/oslaba59/seira4/ask1/mmap
00861000-00882000 rw-p 00000000 00:00 0                                  [heap]
7fbe08e18000-7fbe08fb9000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe08fb9000-7fbe091b9000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe091b9000-7fbe091bd000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe091bd000-7fbe091bf000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe091bf000-7fbe091c3000 rw-p 00000000 00:00 0
7fbe091c3000-7fbe091e4000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fbe093d6000-7fbe093d9000 rw-p 00000000 00:00 0
7fbe093db000-7fbe093dc000 rw-p 00000000 00:00 0
7fbe093dc000-7fbe093dd000 rw-s 00000000 00:04 3995050                    /dev/zero (deleted)
7fbe093dd000-7fbe093de000 r--p 00000000 00:21 9712535                    /home/oslab/oslaba59/seira4/ask1/file.txt
7fbe093de000-7fbe093e3000 rw-p 00000000 00:00 0
7fbe093e3000-7fbe093e4000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fbe093e4000-7fbe093e5000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fbe093e5000-7fbe093e6000 rw-p 00000000 00:00 0
7fffa5ad7000-7fffa5af8000 rw-p 00000000 00:00 0                          [stack]
7fffa5b9e000-7fffa5ba1000 r--p 00000000 00:00 0                          [vvar]
7fffa5ba1000-7fffa5ba3000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------

Child's Virtual Memory Mapping

Virtual Memory Map of process [18180]:
00400000-00403000 r-xp 00000000 00:21 9712462                            /home/oslab/oslaba59/seira4/ask1/mmap
00602000-00603000 rw-p 00002000 00:21 9712462                            /home/oslab/oslaba59/seira4/ask1/mmap
00861000-00882000 rw-p 00000000 00:00 0                                  [heap]
7fbe08e18000-7fbe08fb9000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe08fb9000-7fbe091b9000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe091b9000-7fbe091bd000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe091bd000-7fbe091bf000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fbe091bf000-7fbe091c3000 rw-p 00000000 00:00 0
7fbe091c3000-7fbe091e4000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fbe093d6000-7fbe093d9000 rw-p 00000000 00:00 0
7fbe093db000-7fbe093dc000 rw-p 00000000 00:00 0
7fbe093dc000-7fbe093dd000 rw-s 00000000 00:04 3995050                    /dev/zero (deleted)
7fbe093dd000-7fbe093de000 r--p 00000000 00:21 9712535                    /home/oslab/oslaba59/seira4/ask1/file.txt
7fbe093de000-7fbe093e3000 rw-p 00000000 00:00 0
7fbe093e3000-7fbe093e4000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fbe093e4000-7fbe093e5000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fbe093e5000-7fbe093e6000 rw-p 00000000 00:00 0
7fffa5ad7000-7fffa5af8000 rw-p 00000000 00:00 0                          [stack]
7fffa5b9e000-7fffa5ba1000 r--p 00000000 00:00 0                          [vvar]
7fffa5ba1000-7fffa5ba3000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------
```

Παρατηρούμε πως οι δύο χάρτες μνήμης είναι πανομοιότυποι το οποίο συνάδει με αυτά που ξέρουμε από τη θεωρία για την κλήση συστήματος fork() των συστημάτων UNIX.

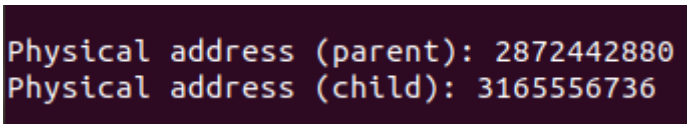
8. Βρίσκουμε και τυπώνουμε τη φυσική διεύθυνση στη κύρια μνήμη του private buffer (Βήμα 3) για τις διεργασίες πατέρα και παιδί.



Παρατηρούμε πως η φυσική διεύθυνση του πίνακα για τις δειργασίες πατέρα και παιδί είναι η ίδια παρότι αυτός είναι “private”. Αυτό συμβαίνει λόγω της τεχνικής Copy-on-Write που αναφέραμε στη θεωρία του μαθήματος.

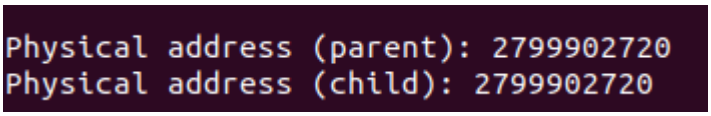
9.

Γράφουμε στον πίνακα από τη διεργασία παιδί και βλέπουμε , όπως αναμέναμε, τη φυσική διεύθυνση μεταξύ των δύο διεργασιών να αλλάζει.



10.

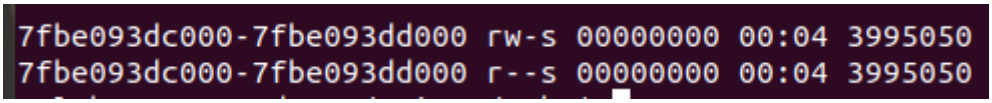
Γράφομε στον shared buffer (Βήμα 6) από τη διεργασία παιδί και τυπώνουμε τη φυσική του διεύθυνση για τις διεργασίες πατέρα και παιδί.



Όπως περιμέναμε, οι φυσικές διευθύνσεις είναι ίδιες καθώς ο πίνακας είναι shared.

11.

Απαγορεύουμε τις εγγραφές στον shared buffer για τη διεργασία παιδί μέσω της κλήσης συστήματος mprotect(). Εντοπίζουμε και τυπώνουμε την απεικόνιση του shared buffer στο χάρτη μνήμης των δύο διεργασιών για να επιβεβαιώσουμε την απαγόρευση.



12.

Αποδεσμεύουμε όλους τους buffers στις δύο διεργασίες μέση της κλήσης συστήματος munmap().

Άσκηση 1.2.1

Ο πηγαίος κώδικας για την άσκηση αυτή παρατίθεται παρακάτω:

```
/*
 * mandel.c
 *
 * A program to draw the Mandelbrot Set on a 256-color xterm.
 *
 */
#include <sys/mman.h>
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <semaphore.h>
#include "mandel-lib.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;
```

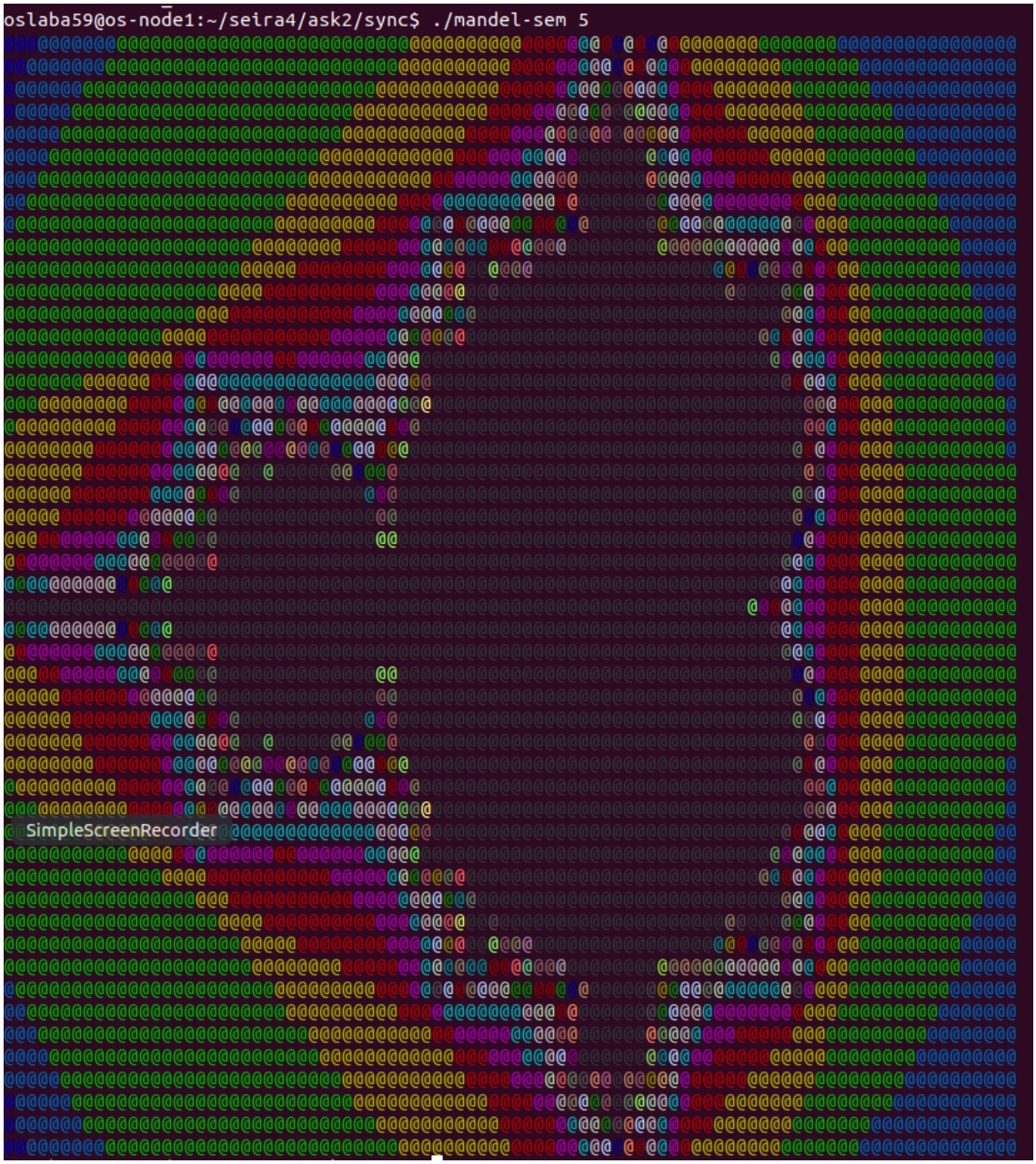
```
double ystep;

/*
 * 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);
        }
}
/*
 * Create a shared memory area, usable by all descendants of the calling
 * 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 number of
         * pages
         */
        pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;// sysconf retrieves the
size of pages in t      //current system

        /* Create a shared, anonymous mapping for this number of pages */
        addr = mmap(NULL, pages * sysconf(_SC_PAGE_SIZE), PROT_READ |
PROT_WRITE, MAP_ANONYMOUS | MAP_SHARED, -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 proc_func(int id, int proccnt, sem_t* semaphores)
{
        // stdout
        int fd = 1;
        // A temporary array, used to hold color values for the line being drawn
        int color_val[x_chars];
        sem_t* curr = &semaphores[id];
        sem_t* next = &semaphores[(id + 1) % proccnt];

        int line;
        for (line = id; line < y_chars; line += proccnt) {
            // Computing can be done asynchronous
            compute_mandel_line(line, color_val);

            // Lock the current semaphore
            sem_wait(curr);
            // Output
            output_mandel_line(fd, color_val);
            // Unlock the next semaphore
            sem_post(next);
        }
        exit(0);
}



int main(int argc, char** argv)
{
        int proccnt, id;
        sem_t* semaphores;

        // Parse the command line
        if (argc != 2) {
            fprintf(stderr, "Usage: %s thread_count\n\n"
```

```
                "Exactly one argument required:\n"
                "\tthread_count: The number of threads to create.\n",
                argv[0]);
        exit(1);
    }
    proccnt = atoi(argv[1]);

    // Create and initialize the semaphores in a shared memory area
    semaphores = (sem_t*) create_shared_memory_area(sizeof(*semaphores) *
proccnt);
    // Value of 1 (binary semaphore), acts like a lock
    if (sem_init(&semaphores[0], 1, 1) != 0) {
        perror("sem_init()");
        exit(1);
    }
    for (id = 1; id < proccnt; id++) {
        // Value of 0, needs another process to sem_post() in order to run
 if (sem_init(&semaphores[id], 1, 0) != 0) {
            perror("sem_init()");
            exit(1);
        }
    }

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

    for (id = 0; id < proccnt; id++) {
        pid_t pid = fork();
        if (pid == -1) {
            perror("fork");
            exit(1);
        }
        if (pid == 0) {
            // Child
            proc_func(id, proccnt, semaphores);
        }
    }

    // Wait for all children to finish
    while (wait(NULL) > 0);

    destroy_shared_memory_area(semaphores, sizeof(*semaphores) * proccnt);
    reset_xterm_color(1);
    return 0;
}
```

Η έξοδος του προγράμματος είναι η ακόλουθη:



Ερωτήσεις:

1. Ρεαλιστικά, αναμένουμε περίπου την ίδια απόδοση και στις δύο υλοποιήσεις. Σε θεωρητικό επίπεδο, η υλοποίηση με νήματα ευνοείται διότι από τη μία απαιτεί λιγότερο χώρο (με τη χρήση της κλήσης συστήματος fork() αντιγράφεται όλος ο χώρος διευθύνσεων της διεργασίας) καθώς και προσφέρει καλύτερη επίδοση αφού στις περιπτώσεις του context switch είναι πιο γρήγορο να εναλλάσουμε διεργασίες παρά νήματα.

Σημείωση: Για μικρό αριθμό N, οι δύο υλοποιήσεις έχουν πρακτικά την ίδια επίδοση. Όσο το N μεγαλώνει, η υλοποίηση με νήματα αρχίζει να εμφανίζει καλύτερη επίδοση.

Άσκηση 1.2.2

Ο πηγαίος κώδικας για την άσκηση αυτή παρατίθεται παρακάτω:

```
/*
 * 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/mman.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <sys/types.h>

/*TODO header file for m(un)map*/

#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, 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;

/*
 * 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);
        }
}




/*
 * Create a shared memory area, usable by all descendants of the calling
 * 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 number of
         * pages
         */
        pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;

        /* Create a shared, anonymous mapping for this number of pages */
        addr = mmap(NULL, pages * sysconf(_SC_PAGE_SIZE), PROT_READ |
PROT_WRITE, MAP_ANONYMOUS | MAP_SHARED, -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 proc_func(int id, int proccnt, int *buffer)
{
        int line;
        for (line = id; line < y_chars; line += proccnt) {
                // Store computed data in the correct position of the shared memory area
                compute_mandel_line(line, &buffer[line * x_chars]);
        }
        exit(0);
}
int main(int argc, char** argv)
{
        int proccnt, id;

        // Parse the command line
        if (argc != 2) {
                fprintf(stderr, "Usage: %s thread_count\n\n"
                        "Exactly one argument required:\n"
                        "\tthread_count: The number of threads to create.\n",
                        argv[0]);
                exit(1);
        }
        proccnt = atoi(argv[1]);
        // Create a shared memory area to store the output data
        int *buffer = create_shared_memory_area(x_chars * y_chars * sizeof(int));

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

        for (id = 0; id < proccnt; id++) {
```

```
            pid_t pid = fork();
            if (pid == -1) {
                perror("fork");
                exit(1);
            }
            if (pid == 0) {
                // Child
                proc_func(id, proccnt, buffer);
            }
        }
        // Wait for all children to finish
        while (wait(NULL) > 0);
    // stdout
        int fd = 1;
        int line;
        for (line = 0; line < y_chars; line++) {
            output_mandel_line(fd, &buffer[line * x_chars]);
        }

        destroy_shared_memory_area(buffer, x_chars * y_chars * sizeof(int));
        reset_xterm_color(1);
        return 0;
}
```

Η έξοδος του προγράμματος είναι η ακόλουθη:

Ερωτήσεις:

1. Ο συγχρονισμός εδώ εξασφαλίζεται με το γεγονός ότι κάθε διεργασία καλή τη συνάρτηση computer_mandel_line(line, &buffer[line*x_chars]); μέσω της οποίας εξασφαλίζεται ότι κάθε διεργασία θα γράφει στο κατάλληλο μέρος του διαμοιραζόμενου μπάφερ που αντιστοιχεί σε αυτή και μόνο αυτή. Όταν όλες οι διεργασίες-παιδιά τελειώσουν τη δουλειά αυτή και πεθάνουν, η διεργασία-πατέρας τυπώνει όλον μπάφερ στην έξοδο και παίρνουμε το επιθυμητό αποτέλεσμα.

Το σχήμα συγχνορισμού, στην περίπτωση που ο μπάφερ είχε μέγεθος NPROCS x x_chars θα επηρεαζόταν ως εξής:

Αν ίσχυε αυτό, θα έπρεπε να εξασφαλίζαμε πώς κάθε διεργασία-παιδί θα έγραφε μία γραμμή, μετά θα σταμάταγε, θα στέλναμε στην έξοδο τη γραμμή αυτή που έγραψε μέσω του μπάφερ και έπειτα θα συνέχιζε η επόμενη διεργασία-παιδί η οποία θα έγραφε στον μπάφερ τη δική της, τώρα, γραμμή και η διαδικασία αυτή θα επαναλαμβάνοταν για όλες τις διεργασίες.