Λειτουργικά Συστήματα Υπολογιστών

4η Εργαστηριακή Αναφορά

Θοδωρής Παπαρρηγόπουλος el18040
Ορφέας Φιλιππόπουλος el18082

Άσκηση 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 <inttypes.h>
#include "help.h"

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

struct stat statbuf;
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)");
    /*
     * TODO: Write your code here to complete child's part of Step 7.
     */

    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.
     */

    //printf("%c \n", heap_private_buf[0]);
    show_va_info((long unsigned int)heap_private_buf);
    pa = get_physical_address((long unsigned int)heap_private_buf);
    printf("Physical address of child = x%" PRIx64 "\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.
     */

    heap_private_buf[0] = "a";
    show_va_info((long unsigned int)heap_private_buf);
  pa = get_physical_address((long unsigned int)heap_private_buf);
  printf("Physical address of child = x%" PRIx64 "\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.
     */

    heap_shared_buf[0] = 'a';
//      printf("isssssssssssssssssssssssssssssss111111 = %c", heap_shared_buf[0]);
  show_va_info((long unsigned int)heap_shared_buf);
  pa = get_physical_address((long unsigned int)heap_shared_buf);
```

```
        printf("Physical address of child = x%" PRIx64 "\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.
            */
//         press_enter();

//         heap_shared_buf = mmap(heap_shared_buf, buffer_size, PROT_READ, MAP_SHARED, -
1, 0);
//         heap_shared_buf[0] = 142;
//         printf("isssssssssssssssssss = %c", heap_shared_buf[0]);
           printf("Child at 0 %d\n", heap_shared_buf[0]);
           int err = mprotect(heap_shared_buf, buffer_size, PROT_READ);
        if(err == EACCES){
              printf("mprotect error.. \n");
              exit(5);
        }
        show_va_info((long unsigned int)heap_shared_buf);



           /*
            * Step 12 - Child
            */
           /*
            * TODO: Write your code here to complete child's part of Step 12.
            */
           press_enter();
           printf("Last question\n");
           err = munmap(file_shared_buf, statbuf.st_size);
           if(err == EINVAL){
                   printf("unmap error for file_shared_buf");
                   exit(5);
           }

           err = munmap(heap_shared_buf, buffer_size);
           if(err == EINVAL){
              printf("unmap error for heap_shared_buf");
              exit(5);
        }

           err = munmap(heap_private_buf, buffer_size);
        if(err == EINVAL){
              printf("unmap error for heap_private_buf");
              exit(5);
        }
```

```
//      delete statbuf;

        show_va_info((long unsigned int)file_shared_buf);
        show_maps();
        uint64_t t = get_physical_address((long unsigned int)file_shared_buf);
        printf("last x%" PRIx64 "\n", t);
}

/*
 * 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.
         */
        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();

        /*
         * TODO: Write your code here to complete parent's part of Step 8.
         */

        show_va_info((long unsigned int)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();

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

    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.
     */

//      heap_shared_buf[0] = "a";
    show_va_info((long unsigned int)heap_shared_buf);
    pa = get_physical_address((long unsigned int)heap_shared_buf);
    printf("Physical address of parent = x%" PRIx64 "\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 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((long unsigned int)heap_shared_buf);
    pa = get_physical_address((long unsigned int)heap_shared_buf);
printf("Physical address of parent = x%" PRIx64 "\n", pa);
    printf("Zero at father %d\n", heap_shared_buf[0]);


    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.
     */

    press_enter();
printf("Last question\n");

    int err;
    err = munmap(file_shared_buf, statbuf.st_size);
if(err == EINVAL){
        printf("unmap error for file_shared_buf");
        exit(5);
}

err = munmap(heap_shared_buf, buffer_size);
if(err == EINVAL){
        printf("unmap error for heap_shared_buf");
        exit(5);
}

err = munmap(heap_private_buf, buffer_size);
```

```
        if(err == EINVAL){
                printf("unmap error for heap_private_buf");
                exit(5);
        }

//          delete statbuf;

            show_va_info((long unsigned int)file_shared_buf);
            show_maps();
            uint64_t t = get_physical_address((long unsigned int)file_shared_buf);
        printf("last x%" PRIx64 "\n", t);

}

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();
//          heap_private_buf = mmap(NULL, buffer_size, PROT_READ|PROT_WRITE,
MAP_ANONYMOUS|MAP_PRIVATE, fd, 0);
            show_maps();

             /* TODO: Write your code here to complete Step 2.
             */

            heap_private_buf = mmap(NULL, buffer_size, PROT_READ|PROT_WRITE,
MAP_ANONYMOUS|MAP_PRIVATE, 0, 0);
        //show_maps();
```

```
if(heap_private_buf == MAP_FAILED){
printf("Mapping Failed\n");
return 1;
}
show_va_info((long unsigned int)heap_private_buf);
//heap_private_buf[0] = 'a';
show_maps();
/*
 * 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.
 */

get_physical_address((long unsigned int)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.
 */

uint64_t i = 0;
for(i = 0; i<buffer_size; i++){
        heap_private_buf[i] = 0;
}


uint64_t address = get_physical_address((long unsigned int)heap_private_buf);
printf("Physical address of buffer = x%" PRIx64 "\n", address);
/*
 * 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.
 */

int fd1;
fd1 = open("file.txt", O_RDONLY);
if(fd1 < 0){
```

```
        printf("Could not open file.txt\n");
        exit(1);
        }

        //struct stat statbuf;
        int err = fstat(fd1, &statbuf);
        if(err < 0){
        printf("Error with struct\n");
        exit(2);
        }

        file_shared_buf = mmap(NULL, statbuf.st_size, PROT_READ, MAP_SHARED, fd1,0);
        if(file_shared_buf == MAP_FAILED){
                printf("Mapping Failed\n");
                return 1;
        }


        show_va_info((long unsigned int)file_shared_buf);
        show_maps();
        //close(fd1);
        ssize_t n = write(1, file_shared_buf, statbuf.st_size);
        if(n != statbuf.st_size){
              printf("Write failed");
        }

        int j;
        for(j = 0; j < statbuf.st_size; j++){
                printf("%c", file_shared_buf[j]);
        }

        printf("\n");
        /*
         * 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, 0, 0);
      //show_maps();
      if(heap_shared_buf == MAP_FAILED){
            printf("Mapping Failed\n");
            return 1;
      }
```

```
        uint64_t k;
    for(k = 0; k < buffer_size; k++){
         heap_shared_buf[k] = 0;
    }

//-------------------------------------------------------------------------------------------------------------

//      pa = get_physical_address((long unsigned int)heap_shared_buf);
//      printf("Physical address of shared  buffer = x%" PRIx64 "\n", pa);

//-------------------------------------------------------------------------------------------------------------
    show_va_info((long unsigned int)heap_shared_buf);
    //heap_private_buf[0] = 'a';
    show_maps();

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

        parent(p);

        if (-1 == close(fd1))
                perror("close");

        printf("main show_maps()");
        show_maps();

        show_va_info((long unsigned int)file_shared_buf);
    show_maps();
    uint64_t t = get_physical_address((long unsigned int)file_shared_buf);
    printf("last x%" PRIx64 "\n", t);

        return 0;
}
```

Ερωτήσεις

1)

```
Step 1: Print the virtual address space map of this process [5602].


Virtual Memory Map of process [5602]:
00400000-00403000 r-xp 00000000 00:21 20197597                           /home/oslab/oslaba33/fourth_lab/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 20197597                           /home/oslab/oslaba33/fourth_lab/mmap/mmap
01bf9000-01c1a000 rw-p 00000000 00:00 0                                  [heap]
7fee16e0b000-7fee16fac000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fee16fac000-7fee171ac000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fee171ac000-7fee171b0000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fee171b0000-7fee171b2000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fee171b2000-7fee171b6000 rw-p 00000000 00:00 0
7fee171b6000-7fee171d7000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fee173c9000-7fee173cc000 rw-p 00000000 00:00 0
7fee173d1000-7fee173d6000 rw-p 00000000 00:00 0
7fee173d6000-7fee173d7000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fee173d7000-7fee173d8000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fee173d8000-7fee173d9000 rw-p 00000000 00:00 0
7ffe89ad3000-7ffe89af4000 rw-p 00000000 00:00 0                          [stack]
7ffe89bda000-7ffe89bdd000 r--p 00000000 00:00 0                          [vvar]
7ffe89bdd000-7ffe89bdf000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

2)

```
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 [6281]:
00400000-00403000 r-xp 00000000 00:21 20197597                           /home/oslab/oslaba33/fourth_lab/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 20197597                           /home/oslab/oslaba33/fourth_lab/mmap/mmap
021ac000-021cd000 rw-p 00000000 00:00 0                                  [heap]
7f536adb2000-7f536af53000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f536af53000-7f536b153000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f536b153000-7f536b157000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f536b157000-7f536b159000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f536b159000-7f536b15d000 rw-p 00000000 00:00 0
7f536b15d000-7f536b17e000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f536b370000-7f536b373000 rw-p 00000000 00:00 0
7f536b377000-7f536b37d000 rw-p 00000000 00:00 0
7f536b37d000-7f536b37e000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f536b37e000-7f536b37f000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f536b37f000-7f536b380000 rw-p 00000000 00:00 0
7ffdc8e9e000-7ffdc8ebf000 rw-p 00000000 00:00 0                          [stack]
7ffdc8f7c000-7ffdc8f7f000 r--p 00000000 00:00 0                          [vvar]
7ffdc8f7f000-7ffdc8f81000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------

7f536b377000-7f536b37d000 rw-p 00000000 00:00 0
```

Παρατηρούμε πως έχουμε μία έξτρα απεικόνιση (κάτω από τις τελείες) στην περιοχή εικονικής μνήμης ανώνυμου τύπου (για την ακρίβεια επεκτάθηκε μία ήδη υπάρχουσα περιοχή εικονικής μνήμης) .

3)

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

VA[0x7fee173d1000] is not mapped; no physical memory allocated.
```

Βλέπουμε πως δεν έχουμε ακόμα αντιστοίχιση σε φυσική μνήμη και είναι λογικό δεδομένου ότι το λειτουργικό δουλεύει με paging on demand που πρακτικά σημαίνει ότι η διασύνδεση θα γίνει μόνο όταν θα χρησιμοποιηθεί αυτή η περιοχή για read/write.

4)

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

Physical address of buffer = x126996000
```

Βλέπουμε πως έχουμε αντιστοίχιση αφού χρησιμοποιήσαμε (initialize) τον buffer.

5)

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

7fee173d0000-7fee173d1000 r--s 00000000 00:21 20197621                    /home/oslab/oslaba33/fourth_lab/mmap/file.txt

Virtual Memory Map of process [5602]:
00400000-00403000 r-xp 00000000 00:21 20197597                            /home/oslab/oslaba33/fourth_lab/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 20197597                            /home/oslab/oslaba33/fourth_lab/mmap/mmap
01bf9000-01c1a000 rw-p 00000000 00:00 0                                   [heap]
7fee16e0b000-7fee16fac000 r-xp 00000000 08:01 6032227                     /lib/x86_64-linux-gnu/libc-2.19.so
7fee16fac000-7fee171ac000 ---p 001a1000 08:01 6032227                     /lib/x86_64-linux-gnu/libc-2.19.so
7fee171ac000-7fee171b0000 r--p 001a1000 08:01 6032227                     /lib/x86_64-linux-gnu/libc-2.19.so
7fee171b0000-7fee171b2000 rw-p 001a5000 08:01 6032227                     /lib/x86_64-linux-gnu/libc-2.19.so
7fee171b2000-7fee171b6000 rw-p 00000000 00:00 0
7fee171b6000-7fee171d7000 r-xp 00000000 08:01 6032224                     /lib/x86_64-linux-gnu/ld-2.19.so
7fee173c9000-7fee173cc000 rw-p 00000000 00:00 0
7fee173cf000-7fee173d0000 rw-p 00000000 00:00 0
7fee173d0000-7fee173d1000 r--s 00000000 00:21 20197621                    /home/oslab/oslaba33/fourth_lab/mmap/file.txt
7fee173d1000-7fee173d6000 rw-p 00000000 00:00 0
7fee173d6000-7fee173d7000 r--p 00020000 08:01 6032224                     /lib/x86_64-linux-gnu/ld-2.19.so
7fee173d7000-7fee173d8000 rw-p 00021000 08:01 6032224                     /lib/x86_64-linux-gnu/ld-2.19.so
7fee173d8000-7fee173d9000 rw-p 00000000 00:00 0
7ffe89ad3000-7ffe89af4000 rw-p 00000000 00:00 0                           [stack]
7ffe89bda000-7ffe89bdd000 r--p 00000000 00:00 0                           [vvar]
7ffe89bdd000-7ffe89bdf000 r-xp 00000000 00:00 0                           [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                   [vsyscall]
-----------------------------------------------------------
```

Βλέπουμε πως δεξιά φαίνεται το path του αρχείου. Στην αριστερή στήλη βλέπουμε την εικονική του μνήμη.

6)

```
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.

7fee173cf000-7fee173d0000 rw-s 00000000 00:04 2263223                     /dev/zero (deleted)

Virtual Memory Map of process [5602]:
00400000-00403000 r-xp 00000000 00:21 20197597                            /home/oslab/oslaba33/fourth_lab/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 20197597                            /home/oslab/oslaba33/fourth_lab/mmap/mmap
01bf9000-01c1a000 rw-p 00000000 00:00 0                                   [heap]
7fee16e0b000-7fee16fac000 r-xp 00000000 08:01 6032227                     /lib/x86_64-linux-gnu/libc-2.19.so
7fee16fac000-7fee171ac000 ---p 001a1000 08:01 6032227                     /lib/x86_64-linux-gnu/libc-2.19.so
7fee171ac000-7fee171b0000 r--p 001a1000 08:01 6032227                     /lib/x86_64-linux-gnu/libc-2.19.so
7fee171b0000-7fee171b2000 rw-p 001a5000 08:01 6032227                     /lib/x86_64-linux-gnu/libc-2.19.so
7fee171b2000-7fee171b6000 rw-p 00000000 00:00 0
7fee171b6000-7fee171d7000 r-xp 00000000 08:01 6032224                     /lib/x86_64-linux-gnu/ld-2.19.so
7fee173c9000-7fee173cc000 rw-p 00000000 00:00 0
7fee173ce000-7fee173cf000 rw-p 00000000 00:00 0
7fee173cf000-7fee173d0000 rw-s 00000000 00:04 2263223                     /dev/zero (deleted)
7fee173d0000-7fee173d1000 r--s 00000000 00:21 20197621                    /home/oslab/oslaba33/fourth_lab/mmap/file.txt
7fee173d1000-7fee173d6000 rw-p 00000000 00:00 0
7fee173d6000-7fee173d7000 r--p 00020000 08:01 6032224                     /lib/x86_64-linux-gnu/ld-2.19.so
7fee173d7000-7fee173d8000 rw-p 00021000 08:01 6032224                     /lib/x86_64-linux-gnu/ld-2.19.so
7fee173d8000-7fee173d9000 rw-p 00000000 00:00 0
7ffe89ad3000-7ffe89af4000 rw-p 00000000 00:00 0                           [stack]
7ffe89bda000-7ffe89bdd000 r--p 00000000 00:00 0                           [vvar]
7ffe89bdd000-7ffe89bdf000 r-xp 00000000 00:00 0                           [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                   [vsyscall]
-----------------------------------------------------------
```

7)
Πατέρας

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


Virtual Memory Map of process [5602]:
00400000-00403000 r-xp 00000000 00:21 20197597                           /home/oslab/oslaba33/fourth_lab/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 20197597                           /home/oslab/oslaba33/fourth_lab/mmap/mmap
01bf9000-01c1a000 rw-p 00000000 00:00 0                                  [heap]
7fee16e0b000-7fee16fac000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fee16fac000-7fee171ac000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fee171ac000-7fee171b0000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fee171b0000-7fee171b2000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fee171b2000-7fee171b6000 rw-p 00000000 00:00 0
7fee171b6000-7fee171d7000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fee173c9000-7fee173cc000 rw-p 00000000 00:00 0
7fee173ce000-7fee173cf000 rw-p 00000000 00:00 0
7fee173cf000-7fee173d0000 rw-s 00000000 00:04 2263223                    /dev/zero (deleted)
7fee173d0000-7fee173d1000 r--s 00000000 00:21 20197621                   /home/oslab/oslaba33/fourth_lab/mmap/file.txt
7fee173d1000-7fee173d6000 rw-p 00000000 00:00 0
7fee173d6000-7fee173d7000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fee173d7000-7fee173d8000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fee173d8000-7fee173d9000 rw-p 00000000 00:00 0
7ffe89ad3000-7ffe89af4000 rw-p 00000000 00:00 0                          [stack]
7ffe89bda000-7ffe89bdd000 r--p 00000000 00:00 0                          [vvar]
7ffe89bdd000-7ffe89bdf000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

Παιδί

```
Virtual Memory Map of process [5711]:
00400000-00403000 r-xp 00000000 00:21 20197597                           /home/oslab/oslaba33/fourth_lab/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 20197597                           /home/oslab/oslaba33/fourth_lab/mmap/mmap
01bf9000-01c1a000 rw-p 00000000 00:00 0                                  [heap]
7fee16e0b000-7fee16fac000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fee16fac000-7fee171ac000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fee171ac000-7fee171b0000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fee171b0000-7fee171b2000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fee171b2000-7fee171b6000 rw-p 00000000 00:00 0
7fee171b6000-7fee171d7000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fee173c9000-7fee173cc000 rw-p 00000000 00:00 0
7fee173ce000-7fee173cf000 rw-p 00000000 00:00 0
7fee173cf000-7fee173d0000 rw-s 00000000 00:04 2263223                    /dev/zero (deleted)
7fee173d0000-7fee173d1000 r--s 00000000 00:21 20197621                   /home/oslab/oslaba33/fourth_lab/mmap/file.txt
7fee173d1000-7fee173d6000 rw-p 00000000 00:00 0
7fee173d6000-7fee173d7000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fee173d7000-7fee173d8000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fee173d8000-7fee173d9000 rw-p 00000000 00:00 0
7ffe89ad3000-7ffe89af4000 rw-p 00000000 00:00 0                          [stack]
7ffe89bda000-7ffe89bdd000 r--p 00000000 00:00 0                          [vvar]
7ffe89bdd000-7ffe89bdf000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

Παρατηρούμε ίδιους χάρτες μνήμης πριν και μέτα το fork().
8)

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

7fee173d1000-7fee173d6000 rw-p 00000000 00:00 0
7fee173d1000-7fee173d6000 rw-p 00000000 00:00 0
Physical address of child = x126996000
```

Για αρχή, η αντιστοίχιση vm – pm είναι ίδια στις διεργασίες αφού και ο χάρτης μνήμης είναι ο ίδιος.
9)

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

7fee173d1000-7fee173d6000 rw-p 00000000 00:00 0
Physical address of child = x47aca000
```

Εδώ ο buffer είναι private. Έτσι, όταν πάει το παιδί να γράψει, τότε αυτόματα θα γίνει Copy on Write και έτσι η μνήμη του πατέρα θα είναι διαφορετική από το παιδί. Αυτό συμβαίνει προκειμένου το λειτουργικό να αντιγράφει τη σελίδα φυσικής μνήμης σε μία νέα σελίδα και οι διεργασίες να έχουν διαφορετικό private buffer.

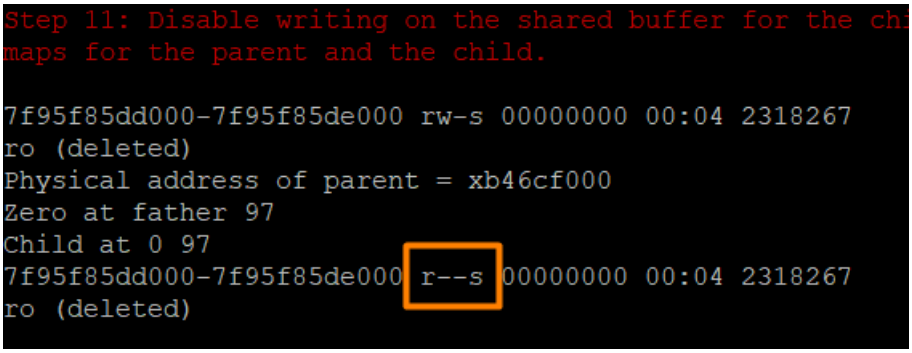
10)

```
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?

7fee173cf000-7fee173d0000 rw-s 00000000 00:04 2263223                    /dev/zero (deleted)
Physical address of parent = xc79b8000
7fee173cf000-7fee173d0000 rw-s 00000000 00:04 2263223                    /dev/zero (deleted)
Physical address of child = xc79b8000
```

Παρατηρούμε πως επειδή ο buffer είναι διαμοιραζόμενος, οι 2 διεργασίες κοιτάνε την ίδια φυσική μνήμη, μπορούν να την προσπελάσουν και να την αλλάξουν ελεύθερα.

11)



Παρατηρούμε πως το δικαίωμα εγγραφής έχει αφαιρεθεί.

Άσκηση 2

Πηγαίος Κώδικας

```
/*
 * 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 <semaphore.h>
#include <sys/types.h>
/*TODO header file for m(un)map*/

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

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

sem_t *mutex;
int NPROCS;
//int * buffer;
int (*buffer) [90]; //90 = x_chars (SECOND SOLUTION WITH BUFFER)

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;
}

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 array_size\n\n"
                "Exactly two argument required:\n"
                "    thread_count: The number of threads to create.\n"
                "    array_size: The size of the array to run with.\n",
                argv0);
        exit(1);
}




/*
 * 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);
	}
}

void compute_and_output_mandel_line(int id)
{
	/*
```

```
         * A temporary array, used to hold color values for the line being drawn
         */
        int color_val[x_chars];

        int i = 0;
        int k = 0;
        for(i = id; i<y_chars; i+=NPROCS){
//                compute_mandel_line(i, color_val);
                /*for(k = 0; k<x_chars; k++){
                        buffer[i*x_chars + k] = color_val[k];
                }*/ //part 2 with buffer
                compute_mandel_line(i, buffer[i]);//for second solution with buffer

//                compute_mandel_line(i, color_val);
//                printf("inside function id is %d\n", id);
//                sem_wait(&mutex[(i % NPROCS)]); //part 2 with semas
//                compute_mandel_line(i, color_val);
//                output_mandel_line(1, color_val); //part 2 with semas
//                sem_post(&mutex[((i+1) % NPROCS)]); //part 2 with semas


        }
}

/*
 * 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 */
        /* TODO:
                addr = mmap(...)
        */

        addr = mmap(NULL, numbytes, PROT_READ|PROT_WRITE, MAP_ANONYMOUS|
MAP_SHARED, -1, 0);
        if(addr == MAP_FAILED){
```

```
                printf("mapping failed..\n");
                exit(2);
        }

        //printf("addr is %lu \n", addr);

        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);
        }
}

int main(int argc, char** argv)
{
        int line;
        //int NPROCS;

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

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

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

        printf("NPROCS = %d\n", NPROCS);

//      mutex = create_shared_memory_area(NPROCS*sizeof(sem_t)); //part2 with semas
```

```
//      buffer = create_shared_memory_area((x_chars*y_chars)*sizeof(int)); //part 2 with buffer
        buffer = (int (*)[x_chars])create_shared_memory_area((x_chars*y_chars)*sizeof(int)); //part
2 with buffer second solution

        //printf("mutex addr is %lu \n", mutex);
        //printf("NPROCS*sizeof(sem_t) = %lu\n", NPROCS*sizeof(sem_t));

        //long unsigned int  size = sizeof(sem_t);

        //printf("size of sem_t is: %lu\n", size);

        int i;

/*      for(i = 0; i<NPROCS; i++){
                if(i == 0){
                 sem_init(&mutex[i], 1, 1);
           }
           else
                 sem_init(&mutex[i], 1, 0);
                //printf("%d\n", mutex[i])
                //printf("size of mutex[%lu] is: %lu\n", i, sizeof(mutex[i]));
        }*/ //part2 with semas

        pid_t pid[NPROCS];
        for(i = 0; i < NPROCS; i++){
        /*      if(i == 0){
                        sem_init(&mutex[i], 0, 1);
                }
                else
                        sem_init(&mutex[i], 0, 0);*/

                pid[i] = fork();
                if(pid[i]<0){
                        perror("main: fork\n");
                        exit(1);

                }

                if(pid[i] == 0){
//                      printf("pid of child is %d\n", pid[i]);
                        compute_and_output_mandel_line(i);
//                      printf("hey\n");
                        exit(1);
                }

        }


        int j;
        int status;
        int p;
        for(j = 0; j<NPROCS; j++){
```

```
                p = wait(&status);
                if(p < 0){
                        printf("something went wrong with childer %d \n", j);
                }
        }

        //for (line = 0; line < y_chars; line++) {
//              compute_and_output_mandel_line(1, line);
//}
        /*int color_val[x_chars];
        int k;
        for(j = 0; j<y_chars; j++){
                for(k = 0; k<x_chars; k++){
                        color_val[k] = buffer[j*x_chars + k];
                }
                output_mandel_line(1, color_val);
        }*/ //part 2 with buffer

        for(i = 0; i < y_chars; i++){
                output_mandel_line(1, buffer[i]);
        }//part 2 with buffer second solution

//      destroy_shared_memory_area(mutex, sizeof(sem_t)*NPROCS);
        destroy_shared_memory_area(buffer, x_chars*y_chars*sizeof(int)); //part 2 with buffer

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

Ερωτήσεις

2.1.1) Αναμένουμε τα threads να είναι γρηγορότερα. Αυτό θα συμβαίνει καθώς το fork() και mmap() είναι χρονοβόρες διαδικασίες ενώ αντιθέτως τα threads είναι αρκετά ελαφριά.

2.1.1)

Όχι κάτι τέτοιο είναι αδύνατο, αφού διεργασίες που δεν έχουν κοινό πατέρα έχουν εντελώς διαφορετικό χάρτη μνήμης επομένως δεν υπάρχει η κοινή αντιστοίχιση vm – pm που θα οδηγήσει τη διεργασία στον διαμοιραζόμενο buffer.

2.2)

1) Οι διεργασίες υπολογίζουν τις γραμμές που τους αντιστοιχούν και τις αποθηκεύουν στην αντίστοιχη γραμμή του δισδιάστατου buffer και αφού τελειώσουν όλες, τότε τυπώνεται ο buffer γραμμή – γραμμή, ώστε οι γραμμές να τυπωθούν με τη σωστή σειρά. Αν ο buffer είχε διαστάσεις NPROCS x x_chars, τότε η διαδικασία θα ήταν περίπου ίδια με το προηγούμενο ερώτημα, αφού οι διεργασίες θα υπολόγιζαν από μία γραμμή η κάθε μια, θα τις αποθήκευαν στο σωστό σημείο του buffer και σε αυτό το σημείο θα έπρεπε να τυπωθεί ολόκληρος ο buffer για να μπορέσουν οι διεργασίες να συνεχίσουν στην τύπωση των επόμενων γραμμών (με σημαιοφόρους θα μπορούσε να γίνει ακόμα πιο γρήγορο αλλά λόγο μεγάλου φόρτου εργασιών δεν παραδόθηκε ο αντίστοιχος κώδικας).