3Η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΠΑΠΑΝΔΡΙΚΟΠΟΥΛΟΣ ΓΡΗΓΟΡΗΣ-ΠΑΝΑΓΙΩΤΗΣ ΑΜ 03121136
ΝΤΑΒΕΑΣ ΣΤΑΣΙΝΟΣ ΑΜ 03121076

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

Παρακάτω παραθέτω τον κώδικα της άσκησης:

```c
/*
 * 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)");
        /*
         * TODO: Write your code here to complete child's part of Step 7.
         */
        printf("child's memory map is the following:\n");
        show_maps();
```

```
 */
printf("child's memory map is the following:\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.
 */
uint64_t phys_addr_frst_page_child = get_physical_address((uint64_t)heap_private_buf);
printf("The physical address of the buffer in main memory AT CHILD'S PROCCESS is : %ld\n",phys_addr_frst_page_child);
printf("The VA info is the following : \n");
show_va_info((uint64_t) heap_private_buf);

/*
 * Step 9 - Child
 */
if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
/*
 * TODO: Write your code here to complete child's part of Step 9.
 */
for(int i =0;i<(int)buffer_size;i++){
                heap_private_buf[i] = 0;
                }

uint64_t phys_addr_frst_page_child_pvt = get_physical_address((uint64_t)heap_private_buf);
printf("The physical address of the private_buffer in main memory AT CHILD'S PROCCESS is : %ld\n",phys_addr_frst_page_child_pvt);
printf("The VA info is the following : \n");
show_va_info((uint64_t) heap_private_buf);



/*
 * Step 10 - Child
 */
if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
/*
 * TODO: Write your code here to complete child's part of Step 10.
 */
uint64_t phys_addr_frst_page_child_shared = get_physical_address((uint64_t)heap_shared_buf);
printf("The physical address of the buffer in main memory AT CHILD'S PROCCESS is : %ld\n",phys_addr_frst_page_child_shared);
printf("The VA info is the following : \n");
show_va_info((uint64_t) heap_shared_buf);

/*
 * 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);
```

```
 mprotect(heap_shared_buf, buffer_size, PROT_READ);
 /*
 Η συνάρτηση mprotect() χρησιμοποιείται για την αλλαγή των δικαιωμάτων πρόσβασης σε μια περιοχή μνήμης.
 Ορίζει τις προστασίες που θα ισχύουν για μια συγκεκριμένη περιοχή μνήμης στην οποία έχει ήδη γίνει
 απεικόνιση ένας χώρος διευθύνσεων μέσω της συνάρτησης mmap()
 */
 printf("VM map of child's proccess is:\n");
 show_maps();
 show_va_info((uint64_t)heap_shared_buf);

 /*
  * Step 12 - Child
  */
 /*
  * TODO: Write your code here to complete child's part of Step 12.
  */
 //deallocate the buffers of child's proccess through munmap function which takes as arguments the address and buffer's size
 if (munmap(heap_shared_buf, buffer_size) == -1) {
         perror("munmap");
         exit(EXIT_FAILURE);
 }
 if (munmap(heap_private_buf, buffer_size) == -1) {
         perror("munmap");
         exit(EXIT_FAILURE);
 }if (munmap(file_shared_buf, buffer_size) == -1) {
         perror("munmap");
         exit(EXIT_FAILURE);




ent process' entry point.

arent(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("map of the parent's process is the following\n");
 show_maps();

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

```
        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.
     */
    uint64_t phys_addr_frst_page_parent = get_physical_address((uint64_t)heap_private_buf);
    printf("The physical address of the buffer in main memory IN PARENT'S PROCCESS is : %ld\n",phys_addr_frst_page_parent);
    printf("The VA info is the following : \n");
    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();

    /*
     * TODO: Write your code here to complete parent's part of Step 9.
     */
    uint64_t phys_addr_frst_page_parent_pvt = get_physical_address((uint64_t)heap_private_buf);
    printf("The physical address of the private_buffer in main memory AT PARENT'S PROCCESS is : %ld\n",phys_addr_frst_page_parent_pvt);
    printf("The VA info is the following : \n");
    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 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.
        */
       uint64_t phys_addr_frst_page_child_shared = get_physical_address((uint64_t)heap_shared_buf);
       printf("The physical address of the buffer in main memory AT CHILD'S PROCCESS is : %ld\n",phys_addr_frst_page_child_shared);
       printf("The VA info is the following : \n");
       show_va_info((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();

       /*
        * TODO: Write your code here to complete parent's part of Step 11.
        */
       printf("VM map of parent's process is:\n");
       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
        */

       /*
        * TODO: Write your code here to complete parent's part of Step 12.
        */
//deallocate the buffers of parent's proccess through munmap function which takes as arguments the address and buffer's size
       if (munmap(heap_shared_buf,buffer_size) == -1) {
              perror("munmap");
```

```
 */
//deallocate the buffers of parent's proccess through munmap function which takes as arguments the address and buffer's size
        if (munmap(heap_shared_buf,buffer_size) == -1) {
                perror("munmap");
                exit(EXIT_FAILURE);
        }
        if (munmap(heap_private_buf,buffer_size) == -1) {
                perror("munmap");
                exit(EXIT_FAILURE);
        }if (munmap(file_shared_buf,buffer_size) == -1) {
                perror("munmap");
                exit(EXIT_FAILURE);
    }
}

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

                if (heap_private_buf == MAP_FAILED) {
                           perror("mmap failed");
                           exit(EXIT_FAILURE);
                        }

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

                show_maps();

        /*
         * Step 2: Use mmap to allocate a buffer of 1 page and print the map
```

```
        /*
         * 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.
         */
                //Δημιουργία απεικόνισης μνήμης
//το πρώτο όρισμα addr ειναι η αρχική διέυυθυνση του χάρτη .
//Το αρχικοποιούμε στην τιμή NULL προκειμένου ο πυρήνας να επιλέξει την(page-aligned)διέυθυνση στην οποία θα δημιουργηθεί η απεικόνιση,
//Επειτα ως length βάζω το buffer_size,στο int prot που περιγράφει την επιθυμητή προστασία μνήμης της απεικόνισης βάζω την ετικέτα PROT_READ|PROT_WRITE,
//μετά αρχικοποιώ τις σημαίες σε MAP_PRIVATE(καθορίζει τον τρόπο χαρτογράφησης ο οποίος ειναι private),
//έπειτα το περιγραφητή αρχείου fd και τέλος offset  = 0(προκειτα για την αρχική θέση από την οποία θα ξεκινησει η χαρτογράφηση)
                 heap_private_buf = mmap(NULL,buffer_size,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,fd,0);

                if (heap_private_buf == MAP_FAILED) {
                           perror("mmap failed");
                           exit(EXIT_FAILURE);
                        }

                show_maps();
                show_va_info((uint64_t)heap_private_buf);
                //Απελευθέρωση της απεικόνισης μνήμης
                /*
                if(munmap(heap_private_buf,buffer_size)== -1){
                        perror("munmap Failed");
                        exit(EXIT_FAILURE);
                }
                */
        /*
         * 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.
         */
        //καλώ την get_physical_address για να βρώ την φυσική διέυθυνση της πρώτης σελίδας του buffer
        //Ως όρισμα παίρνει την εικονικλη διέυθυνση μνήμης τύπου uint64_t για αυτό και μετατρέπω τον τύπο του heap_private_buf καταλλήλως
                uint64_t phys_addr_frst_page = get_physical_address((uint64_t)heap_private_buf);
                printf("The physical address of the buffer in main memory is : %ld\n",phys_addr_frst_page);


        /*
         * 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();
```

```c
        /*
         * TODO: Write your code here to complete Step 4.
         */
         for(int i = 0;i<(int)buffer_size;i++){
                heap_private_buf[i] = 0;
                }

        uint64_t phys_addr_frst_page_zeros = get_physical_address((uint64_t)heap_private_buf);
                printf("The physical address of the buffer INITIALIZED WITH ZEROS in main memory is : %ld\n",phys_addr_frst_page_zeros);


        /*
         * 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.
         */


                //τυπωση του περιεχομένου του αρχείου
                FILE *fp;
                char c;

                fp = fopen("file.txt","r");
                if(fp ==NULL){
                        printf("error while opening");
                        exit(0);}

                while((c = fgetc(fp))!=EOF)
                        printf("%c",c);
                fclose(fp);
                printf("\n");
        int fd_new = open("file.txt", O_RDONLY);//ανοίγουμε το αρχείο file.txt για διάβασμα και
                                                //μέσω της open επιστρέφεται ο περιγραφητής του αρχείου
        if (fd_new == -1) {
                perror("open");

                }

        file_shared_buf = mmap(NULL,buffer_size,PROT_READ,MAP_SHARED,fd_new,0);

        if (file_shared_buf == MAP_FAILED) {
                perror("mmap failed");
                exit(EXIT_FAILURE);
               }

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

```
        /*
         * Step 6: Use mmap(2) to allocate a shared buffer of 1 page. Use
         * heap_shared_buf.
         */
        printf(RED "\nStep 6: Use mmap(2) to allocate a shared buffer of size "
                "equal to 1 page. Initialize the buffer and print the new "
                "mapping information that has been created.\n" RESET);
        press_enter();
        /*
         * TODO: Write your code here to complete Step 6        */

        heap_shared_buf=mmap(NULL, buffer_size, PROT_READ| PROT_WRITE,MAP_SHARED| MAP_ANONYMOUS,-1,0);
        if (heap_shared_buf == MAP_FAILED) {
                perror("mmap failed");
                exit(EXIT_FAILURE);
                }
        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 τυπώσαμε τον χάρτη εικονικής μνήμης μέσω της εντολής show_maps()

```
Step 1: Print the virtual address space map of this process [1084106].


Virtual Memory Map of process [1084106]:
55f5612e6000-55f5612e7000 r--p 00000000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e7000-55f5612e8000 r-xp 00001000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e8000-55f5612e9000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e9000-55f5612ea000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612ea000-55f5612eb000 rw-p 00003000 00:27 7620005                    /home/oslab/oslab025/mmap
55f561bd1000-55f561bf2000 rw-p 00000000 00:00 0                          [heap]
7f186699d000-7f18669bf000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f18669bf000-7f1866b18000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b18000-7f1866b67000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b67000-7f1866b6b000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6b000-7f1866b6d000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6d000-7f1866b73000 rw-p 00000000 00:00 0
7f1866b79000-7f1866b7a000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b7a000-7f1866b9a000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b9a000-7f1866ba2000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba2000-7f1866ba3000 rw-p 00000000 00:00 0
7f1866ba3000-7f1866ba4000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba4000-7f1866ba5000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba5000-7f1866ba6000 rw-p 00000000 00:00 0
7fffacf6c000-7fffacf8d000 rw-p 00000000 00:00 0                          [stack]
7fffacfd5000-7fffacfd9000 r--p 00000000 00:00 0                          [vvar]
7fffacfd9000-7fffacfdb000 r-xp 00000000 00:00 0                          [vdso]
----------------------------------------------------------
```

Από την απεικόνιση παρατηρούμε πληροφορίες για την μνήμη του συστήματος η οποία είναι χωρισμένη σε frames . Ειδικότερα, σε κάθε σειρά που απεικονίζεται οι δύο πρώτοι αριθμοί αποτελούν την virtual memory του κάθε πλαισίου. Αμέσως ,μετά απεικονίζεται η πληροφορία σχετικά με τα δικαιώματα πρόσβασης που έχουμε στο αρχείο w,r,x (write,read και execute αντίστοιχα).Έπειτα, απεικονίζεται το offset του κάθε πλαισίου και τέλος το path του κάθε αρχείου.

2) Μετά την κλήση του συστήματος mmap() δεσμεύσαμε τον buffer μεγέθους μίας σελίδας και το τυπώσαμε ξανά στον χρήστη

```
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 [1084106]:
55f5612e6000-55f5612e7000 r--p 00000000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e7000-55f5612e8000 r-xp 00001000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e8000-55f5612e9000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e9000-55f5612ea000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612ea000-55f5612eb000 rw-p 00003000 00:27 7620005                    /home/oslab/oslab025/mmap
55f561bd1000-55f561bf2000 rw-p 00000000 00:00 0                          [heap]
7f186699d000-7f18669bf000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f18669bf000-7f1866b18000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b18000-7f1866b67000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b67000-7f1866b6b000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6b000-7f1866b6d000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6d000-7f1866b73000 rw-p 00000000 00:00 0
7f1866b78000-7f1866b79000 rw-p 00000000 00:00 0
7f1866b79000-7f1866b7a000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b7a000-7f1866b9a000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b9a000-7f1866ba2000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba2000-7f1866ba3000 rw-p 00000000 00:00 0
7f1866ba3000-7f1866ba4000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba4000-7f1866ba5000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba5000-7f1866ba6000 rw-p 00000000 00:00 0
7fffacf6c000-7fffacf8d000 rw-p 00000000 00:00 0                          [stack]
7fffacfd5000-7fffacfd9000 r--p 00000000 00:00 0                          [vvar]
7fffacfd9000-7fffacfdb000 r-xp 00000000 00:00 0                          [vdso]
-------------------------------------------------------

7f1866b78000-7f1866b79000 rw-p 00000000 00:00 0
```

Η mmap πήρε ως ορίσματα αρχικά το NULL καθώς παίρνει default τιμές από το σύστημα, έπειτα βάζουμε ως όρισμα το μέγεθος του χώρου που θα δεσμέυσιυμε, μετά τα permissions flags, τέτοια ώστε το αρχείο να μπορεί να εγγραφεί και αναγνωστεί. Στην συνέχεια την θέτουμε MAP_PRIVATE προκειμένου να μην έχουν πρόσβαση πολλές διεργασίες και ο χώρος να είναι ιδιωτικός και την θέτουμε και ANONYMOUS για να δείξουμε ότι ο χώρος αυτός δεν αναφέρεται σε κάποιο αρχείο ,Επειτα βάζουμε τον περιγραφητή αρχείου να παίρνει την τιμή fd δηλαδή -1 ,προκειμένου να δηλώσουμε ότι ο χώρος που δημιουργήθηκε δεν αναφέρεται σε κάποιο αρχείο και τέλος offset 0.Ελέγχουμε για τυχόν σφάλμα και τυπώνουμε τον χάρτη μνήμης συνοδευόμενο από τις πληροφορίες .Παρατηρούμε λοιπόν ότι στον χάρτη εικονικής μνήμης προστέθηκε μία ακόμη γραμμή με τα στοιχεία που της δώσαμε πχ offset = 0 δυνατότητα εγγραφης και ανάγνωσης.

3)Σε αυτό το βήμα προσπαθούμε να βοούμε και να τυπώσουμε τη φυσική διεύθυνση μνήμης στην οποία απεικονίζεται η εικονική διεύθυνση του buffer ΄μεσω της συνάρτησης get_physical_address:

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

VA[0x7f1866b78000] is not mapped; no physical memory allocated.
The physical address of the buffer in main memory is : 0
```

Παρατηρούμε, ότι παρά το γεγονός ότι η μνήμη έχει δεσμευτεί, η απεικόνιση της εικονικής μνήμης στη φυσική δεν έχει ακόμη πραγματοποιηθεί. Αυτό συμβαίνει καθώς, η φυσική μνήμη δεσμεύεται ON DEMAND από το λειτουργικό σύστημα όταν χρειαστεί να προσπελάσει τα δεδομένα. Με αυτόν τον τρόπο το ΛΣ εξοικονομεί χώρο καθώς δεν τον δεσμεύει μέχρι να τον αξιοποιήσουμε.

4) Γεμίσμα με μηδενικά στον buffer και επανάληψη του τρίτου βήματος .

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

The physical address of the buffer INITIALIZED WITH ZEROS in main memory is : 5891350528
```

Παρατηρούμε ότι έχει γίνει απεικόνιση στην φυσική μνήμη για τους λόγους που εξηγήσαμε προηγουμένως.

5) Χρησιμοποιούμε την mmap() για να απεικονίσουμε το αρχείο file.txt στον χώρο διευθύνσεων της διεργασίας και τυπώνουμε το περιεχόμενό του. Η μόνη διαφορά στα ορίσματα σε σχέση με αυτά της προηγούμενης χρήσης mmap είναι ότι αντί για MAP_PRIVATE χρησιμοποιήσαμε MAP_SHARED προκειμένου όλες οι διεργασίες να έχουν πλέον πρόσβαση στο αρχείο. Ακόμη, αξίζει να σημειωθεί πως ανοίξαμε το αρχείο για ανάγνωση και τυπώσαμε το περιεχόμενο του το οποίο απεικονίζεται και στο cmd (hello world!)

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


Virtual Memory Map of process [1084106]:
55f5612e6000-55f5612e7000 r--p 00000000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e7000-55f5612e8000 r-xp 00001000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e8000-55f5612e9000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e9000-55f5612ea000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612ea000-55f5612eb000 rw-p 00003000 00:27 7620005                    /home/oslab/oslab025/mmap
55f561bd1000-55f561bf2000 rw-p 00000000 00:00 0                          [heap]
7f186699d000-7f18669bf000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f18669bf000-7f1866b18000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b18000-7f1866b67000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b67000-7f1866b6b000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6b000-7f1866b6d000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6d000-7f1866b73000 rw-p 00000000 00:00 0
7f1866b77000-7f1866b78000 r--s 00000000 00:27 7620996                    /home/oslab/oslab025/file.txt
7f1866b78000-7f1866b79000 rw-p 00000000 00:00 0
7f1866b79000-7f1866b7a000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b7a000-7f1866b9a000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b9a000-7f1866ba2000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba2000-7f1866ba3000 rw-p 00000000 00:00 0
7f1866ba3000-7f1866ba4000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba4000-7f1866ba5000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba5000-7f1866ba6000 rw-p 00000000 00:00 0
7fffacf6c000-7fffacf8d000 rw-p 00000000 00:00 0                          [stack]
7fffacfd5000-7fffacfd9000 r--p 00000000 00:00 0                          [vvar]
7fffacfd9000-7fffacfdb000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------------

7f1866b77000-7f1866b78000 r--s 00000000 00:27 7620996                    /home/oslab/oslab025/file.txt
```

Τέλος, παρατηρούμε στην τελική γραμμή τις πληροφορίες(τα permissions offset κ.α.) της εικονικής μνήμης για το αρχείο που δώσαμε.

6)Χρησιμοποιούμε την mmap() για να δεσμεύσουμε έναν νέο buffer, διαμοιραζόμενο (shared) αυτή τη φορά μεταξύ διεργασιών και anonymous για να δείξουμε ότι δεν αναφέρεται σε κάποιο αρχειό .Απεικονίζοντας παρακάτω την εικονική μνήμη παρατηρούμε ότι ο χώρος που δεσμεύσαμε δείχνει ότι το path του αρχείο είναι deleted όπως και αναμέναμε καθώς ο χώρος αυτός θα ήταν διαμοιραζόμενος από τις διεργασίες ,όμως αφού η διεργασία πατέρας έχει τερματιστεί, ο χώρος αυτός δεν έχει λόγο ύπαρξης. Έτσι το ΛΣ εξοικονομεί χώρο.

```
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.


Virtual Memory Map of process [1084106]:
55f5612e6000-55f5612e7000 r--p 00000000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e7000-55f5612e8000 r-xp 00001000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e8000-55f5612e9000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e9000-55f5612ea000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612ea000-55f5612eb000 rw-p 00003000 00:27 7620005                    /home/oslab/oslab025/mmap
55f561bd1000-55f561bf2000 rw-p 00000000 00:00 0                          [heap]
7f186699d000-7f18669bf000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f18669bf000-7f1866b18000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b18000-7f1866b67000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b67000-7f1866b6b000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6b000-7f1866b6d000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6d000-7f1866b73000 rw-p 00000000 00:00 0
7f1866b76000-7f1866b77000 rw-s 00000000 00:01 9001                       /dev/zero (deleted)
7f1866b77000-7f1866b78000 r--s 00000000 00:27 7620996                    /home/oslab/oslab025/file.txt
7f1866b78000-7f1866b79000 rw-p 00000000 00:00 0
7f1866b79000-7f1866b7a000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b7a000-7f1866b9a000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b9a000-7f1866ba2000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba2000-7f1866ba3000 rw-p 00000000 00:00 0
7f1866ba3000-7f1866ba4000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba4000-7f1866ba5000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba5000-7f1866ba6000 rw-p 00000000 00:00 0
7fffacf6c000-7fffacf8d000 rw-p 00000000 00:00 0                          [stack]
7fffacfd5000-7fffacfd9000 r--p 00000000 00:00 0                          [vvar]
7fffacfd9000-7fffacfdb000 r-xp 00000000 00:00 0                          [vdso]
-------------------------------------------------------

7f1866b76000-7f1866b77000 rw-s 00000000 00:01 9001                       /dev/zero (deleted)
```

7) Στο σημείο που έχουμε σημειώσει η fork() καλείται και δημιουργείται μία νέα διεργασία. Τυπώνουμε τους χάρτες της εικονικής μνήμης της γονικής διεργασίας και της διεργασίας παιδιού

```
Step 7: Print parent's and child's map.
^[
map of the parent's process is the following

Virtual Memory Map of process [1084106]:
55f5612e6000-55f5612e7000 r--p 00000000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e7000-55f5612e8000 r-xp 00001000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e8000-55f5612e9000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e9000-55f5612ea000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612ea000-55f5612eb000 rw-p 00003000 00:27 7620005                    /home/oslab/oslab025/mmap
55f561bd1000-55f561bf2000 rw-p 00000000 00:00 0                          [heap]
7f186699d000-7f18669bf000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f18669bf000-7f1866b18000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b18000-7f1866b67000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b67000-7f1866b6b000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6b000-7f1866b6d000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6d000-7f1866b73000 rw-p 00000000 00:00 0
7f1866b76000-7f1866b77000 rw-s 00000000 00:01 9001                       /dev/zero (deleted)
7f1866b77000-7f1866b78000 r--s 00000000 00:27 7620996                    /home/oslab/oslab025/file.txt
7f1866b78000-7f1866b79000 rw-p 00000000 00:00 0
7f1866b79000-7f1866b7a000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b7a000-7f1866b9a000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b9a000-7f1866ba2000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba2000-7f1866ba3000 rw-p 00000000 00:00 0
7f1866ba3000-7f1866ba4000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba4000-7f1866ba5000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba5000-7f1866ba6000 rw-p 00000000 00:00 0
7fffacf6c000-7fffacf8d000 rw-p 00000000 00:00 0                          [stack]
7fffacfd5000-7fffacfd9000 r--p 00000000 00:00 0                          [vvar]
7fffacfd9000-7fffacfdb000 r-xp 00000000 00:00 0                          [vdso]
-------------------------------------------------------
```

```
child's memory map is the following:

Virtual Memory Map of process [1084134]:
55f5612e6000-55f5612e7000 r--p 00000000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e7000-55f5612e8000 r-xp 00001000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e8000-55f5612e9000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e9000-55f5612ea000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612ea000-55f5612eb000 rw-p 00003000 00:27 7620005                    /home/oslab/oslab025/mmap
55f561bd1000-55f561bf2000 rw-p 00000000 00:00 0                          [heap]
7f186699d000-7f18669bf000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f18669bf000-7f1866b18000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b18000-7f1866b67000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b67000-7f1866b6b000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6b000-7f1866b6d000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6d000-7f1866b73000 rw-p 00000000 00:00 0
7f1866b76000-7f1866b77000 rw-s 00000000 00:01 9001                       /dev/zero (deleted)
7f1866b77000-7f1866b78000 r--s 00000000 00:27 7620996                    /home/oslab/oslab025/file.txt
7f1866b78000-7f1866b79000 rw-p 00000000 00:00 0
7f1866b79000-7f1866b7a000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b7a000-7f1866b9a000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b9a000-7f1866ba2000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba2000-7f1866ba3000 rw-p 00000000 00:00 0
7f1866ba3000-7f1866ba4000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba4000-7f1866ba5000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba5000-7f1866ba6000 rw-p 00000000 00:00 0
7fffacf6c000-7fffacf8d000 rw-p 00000000 00:00 0                          [stack]
7fffacfd5000-7fffacfd9000 r--p 00000000 00:00 0                          [vvar]
7fffacfd9000-7fffacfdb000 r-xp 00000000 00:00 0                          [vdso]
```

Παρατηρούμε, ότι η διεργασία παιδί μου δημιουργήθηκε κατά την εκτέλεση της fork() αποτελεί αντίγραφο του parent και κληρονομεί και την εικονική μνήμη του. Ακόμη κληρονομεί και ένα αντίγραφο σελίδων (page table) του parent ενώ παράλληλα 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 (Βήμα 3) για τις διεργασίες γονέα και παιδιού.

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

The physical address of the buffer in main memory IN PARENT'S PROCCESS is : 5891350528
The VA info is the following :
7f1866b78000-7f1866b79000 rw-p 00000000 00:00 0
The physical address of the buffer in main memory AT CHILD'S PROCCESS is : 5891350528
The VA info is the following :
7f1866b78000-7f1866b79000 rw-p 00000000 00:00 0
```

Αμέσως μετά το fork παρατηρούμε ότι απεικονίζονται στην ίδια φυσική μνήμη καθώς παιδί και πατέρας μοιράζονται κοινή μνήμη αφού δεν έχει γίνει ακόμη κάποα εγγραφή είτε στον πατέρα είτε στο παιδί (COW).

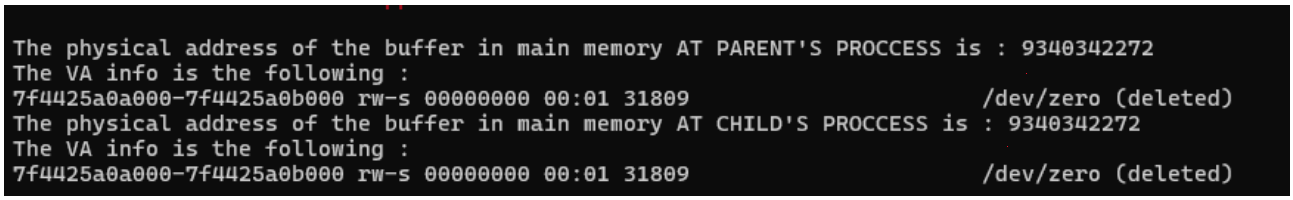
9)Γράφουμε στον private buffer από τη διεργασία παιδί και επαναλαμβάνουμε το Βήμα 8.

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

The physical address of the private_buffer in main memory AT PARENT'S PROCCESS is : 5891350528
The VA info is the following :
7f1866b78000-7f1866b79000 rw-p 00000000 00:00 0
The physical address of the private_buffer in main memory AT CHILD'S PROCCESS is : 4921712640
The VA info is the following :
7f1866b78000-7f1866b79000 rw-p 00000000 00:00 0
```

Όταν μία διεργασία παιδί δημιουργείται μέσω fork() αντιγράφεται η εικονική μνήμη και πίνακας σελίδων με την ταυτόχρονη αφαίρεση του δικαιώματος write στις private pages και η απεικόνιση της εικονικής διεύθυνσης στην φυσική διεύθυνση είναι ίδια μέχρι μία διεργασία να προσπαθήσει να

κάνει write σε μια private page. Όταν συμβεί αυτό το λειτουργικό σύστημα βρίσκει νέο frame για την απεικόνιση αντιγράφεται το περιεχόμενο του page στην νέα φυσική διεύθυνση και ενήμερώνεται ο πίνακας σελίδων (COW).Για αυτό τον λόγο επομένως οι διευθύνσεις στην προκειμένη περίπτωση είναι διαφορετικές.

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



Σε αντίθεση με τον private buffer ο shared buffer απεικονίζει την ίδια φυσική διεύθυνση και για την διεργασία παιδί και την διεργασία πατέρα. Αυτό συμβαίνει επειδή η page δεν είναι πλέον private αλλά shared μεταξύ των διεργασιών.

11)Απαγορεύουμε τις εγγραφές στον shared buffer για τη διεργασία παιδί αφαιρώντας το write permission μέσω της εντολής mprotect().

```
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's process is:

Virtual Memory Map of process [1084106]:
55f5612e6000-55f5612e7000 r--p 00000000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e7000-55f5612e8000 r-xp 00001000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e8000-55f5612e9000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e9000-55f5612ea000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612ea000-55f5612eb000 rw-p 00003000 00:27 7620005                    /home/oslab/oslab025/mmap
55f561bd1000-55f561bf2000 rw-p 00000000 00:00 0                          [heap]
7f186699d000-7f18669bf000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f18669bf000-7f1866b18000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b18000-7f1866b67000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b67000-7f1866b6b000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6b000-7f1866b6d000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6d000-7f1866b73000 rw-p 00000000 00:00 0
7f1866b76000-7f1866b77000 rw-s 00000000 00:01 9001                       /dev/zero (deleted)
7f1866b77000-7f1866b78000 r--s 00000000 00:27 7620996                    /home/oslab/oslab025/file.txt
7f1866b78000-7f1866b79000 rw-p 00000000 00:00 0
7f1866b79000-7f1866b7a000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b7a000-7f1866b9a000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b9a000-7f1866ba2000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba2000-7f1866ba3000 rw-p 00000000 00:00 0
7f1866ba3000-7f1866ba4000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba4000-7f1866ba5000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba5000-7f1866ba6000 rw-p 00000000 00:00 0
7fffacf6c000-7fffacf8d000 rw-p 00000000 00:00 0                          [stack]
7fffacfd5000-7fffacfd9000 r--p 00000000 00:00 0                          [vvar]
7fffacfd9000-7fffacfdb000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------

7f1866b76000-7f1866b77000 rw-s 00000000 00:01 9001                       /dev/zero (deleted)
VM map of child's proccess is:
```

```
7f1866b76000-7f1866b77000 rw-s 00000000 00:01 9001                       /dev/zero (deleted)
VM map of child's proccess is:

Virtual Memory Map of process [1084134]:
55f5612e6000-55f5612e7000 r--p 00000000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e7000-55f5612e8000 r-xp 00001000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e8000-55f5612e9000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612e9000-55f5612ea000 r--p 00002000 00:27 7620005                    /home/oslab/oslab025/mmap
55f5612ea000-55f5612eb000 rw-p 00003000 00:27 7620005                    /home/oslab/oslab025/mmap
55f561bd1000-55f561bf2000 rw-p 00000000 00:00 0                          [heap]
7f186699d000-7f18669bf000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f18669bf000-7f1866b18000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b18000-7f1866b67000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b67000-7f1866b6b000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6b000-7f1866b6d000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1866b6d000-7f1866b73000 rw-p 00000000 00:00 0
7f1866b76000-7f1866b77000 r--s 00000000 00:01 9001                       /dev/zero (deleted)
7f1866b77000-7f1866b78000 r--s 00000000 00:27 7620996                    /home/oslab/oslab025/file.txt
7f1866b78000-7f1866b79000 rw-p 00000000 00:00 0
7f1866b79000-7f1866b7a000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b7a000-7f1866b9a000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866b9a000-7f1866ba2000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba2000-7f1866ba3000 rw-p 00000000 00:00 0
7f1866ba3000-7f1866ba4000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba4000-7f1866ba5000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1866ba5000-7f1866ba6000 rw-p 00000000 00:00 0
7fffacf6c000-7fffacf8d000 rw-p 00000000 00:00 0                          [stack]
7fffacfd5000-7fffacfd9000 r--p 00000000 00:00 0                          [vvar]
7fffacfd9000-7fffacfdb000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------

7f1866b76000-7f1866b77000 r--s 00000000 00:01 9001                       /dev/zero (deleted)
```

Και παρατηρούμε ότι έχει αφαιρεθεί το δικαίωμα write από την διεργασία παιδί ενώ στον πατέρα το δικαίωμα εγγραφής έχει διατηρηθεί

12)Μέσω της munmap() αποδεσμεύουμε τους buffers

(Το παραπάνω πρόγραμμα γίνεται executable μέσω του Makefile2)

1.2.1 Semaphores πάνω από διαμοιραζόμενη μνήμη

Αρχικά ορίσαμε την global variable pprocesses στην οποία θα αποθηκευτεί μέσω του cmd πόσες διεργασίες θα εκτελέσουν το πρόγραμμα .Επιπλέον δημιουργούμε έναν δείκτη σε σημαφορο στον οποίο στην συνέχεια θα δεσμέυσουμε χώρο ώστε κάθε διεργασία να έχει και ένα σημαφόρο. Η process_function παίρνει σαν όρισμα την start_line που καθορίζει την αρχική γραμμή για την επεξεργασία και share_buffer ένα δείκτη σε πίνακα. Βρόγχος for ξεκινά από το start_line και προχωρά με ίσα βήματα pprocesses.Αυτό εξασφαλίζει ότι κάθε διεργασία επεξεργάζεται γραμμές σε διαστήματα που είναι καθορισμένα από ένα πλήθος των διεργασιών (pprocesses).Η sem_wait(&semaphores[i]) περιμένει να αποκτήσει τον σημαφόρο για την i-οστή γραμμή.Με αυτό τον τρόπο διασφαλίζουμε ότι καμία άλλη διεργασία δεν επεξεργάζεται ή γράφει ταυτόχρονα στην i-οστή γραμμή του shared_buffer. Αφού αποκτήσει πρόσβαση του σημαφόρου η memcpy της string.h η διεργασία αντιγράφει τις τιμές του πίνακα color_val στον αντίστοιχο χώρο στη γραμμή i του shared_buffer. Τέλος μέσω της sem_post(&semaphores[i]) η διεργασία απελευθερώνει τον σημαφόρο για τη γραμμή i επιτρέποντας έτσι άλλες διεργασίες να αποκτήσουν πρόσβαση σε αυτή την γραμμή. Στην συνέχεια χρησιμοποιούμε την συνάρτηση create_shared_memory_area και δημιουργούμε μια περιοχή κοινόχρηστης μνήμης με μέγεθος που καθορίζεται από τον αριθμό των bytes που παρέχεται ως είσοδος. Αρχικά, ελέγχουμε αν η είσοδος numbytes είναι μηδενική και, αν ναι, εκτυπώνεουμε μήνυμα σφάλματος και τερματίζουμε το πρόγραμμα. Στη συνέχεια, υπολογίζουμε, τον αριθμό των σελίδων μνήμης που απαιτούνται για να καλύψουν το μέγεθος numbytes, στρογγυλοποιώντας προς τα επάνω. Στην συνέχεια χρησιμοποιώντας τη συνάρτηση mmap για να δημιουργήσουμε μια ανώνυμη και κοινόχρηστη αντιστοίχιση μνήμης (MAP_SHARED | MAP_ANONYMOUS) με δικαιώματα ανάγνωσης και εγγραφής (PROT_READ | PROT_WRITE). Εάν η κλήση της mmap αποτύχει, εκτυπώνεται μήνυμα σφάλματος και το πρόγραμμα τερματίζεται. Τέλος, επιστρέφουμε τη διεύθυνση της περιοχής μνήμης που δημιουργήσαμε.Στην συνέχεια δημιουργούμε την destroy_shared_memory για να αποδεσμέυσουμε περιοχή μνήμης που δεσμέυσαμε μέσω της προηγούμενης συνάρτησης .Για αυτό μέσα στην συνάρτηση καλούμε την munmap προκειμένου να αποδεσμέυσουμε τον χώρο μνήμης που θέλουμε. Στην main αρχικά δημιουργούμε την κοινόχρηστη μνήμη για τον πίνακα με τη συνάρτηση create_shared_memory_area, και στη συνέχεια δημιουργούμε και αρχικοποιούμε μια περιοχή μνήμης για τους σημαφόρους, κάθε ένας από τους οποίους τίθεται στην αρχική τιμή 1 (παίρνουν στα δύο τελευταία ορίσματα 1 προκειμένου τώρα σε αντίθεση με τις προηγούμενες mandel διαχειριζόμαστε διεργασίες αντί για νήματα και επίσης θέλουμε να είναι ξεκλείδωτες). Ακολουθεί η δυναμική δέσμευση μνήμης για την αποθήκευση των PID των παιδικών διεργασιών. Σε έναν βρόχο, χρησιμοποιούμε τη fork για να δημιουργήσουμε τις παιδικές διεργασίες. Κάθε παιδική διεργασία καλεί τη συνάρτηση process_function για να επεξεργαστεί τις γραμμές της εικόνας και να αποθηκεύσει τα αποτελέσματα στον κοινόχρηστο πίνακα, χρησιμοποιώντας σημαφόρους για να διασφαλιστεί ο συγχρονισμός. Αν οποιαδήποτε από αυτές τις ενέργειες αποτύχει, εκτυπώνεται μήνυμα σφάλματος και το πρόγραμμα τερματίζεται.Τέλος μέσω της destroyed-shared_memory απδεσμέυουμε τον χώρο που δεσμέυσαμε για την δημιουργία των αντίστοιχων χώρων μνήμης και με την free(child_pids) αποδεσμέυουμε χώρο για τις διεργασίες παιδιά που δημιουργήσαμε.

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

#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"
/* TODO: Include necessary headers for mmap and munmap */

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

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

/*
 * Output at the terminal 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;

int PPROCESSES; // Number of processes

// Handler to reset xterm color and exit on CTRL+C
```

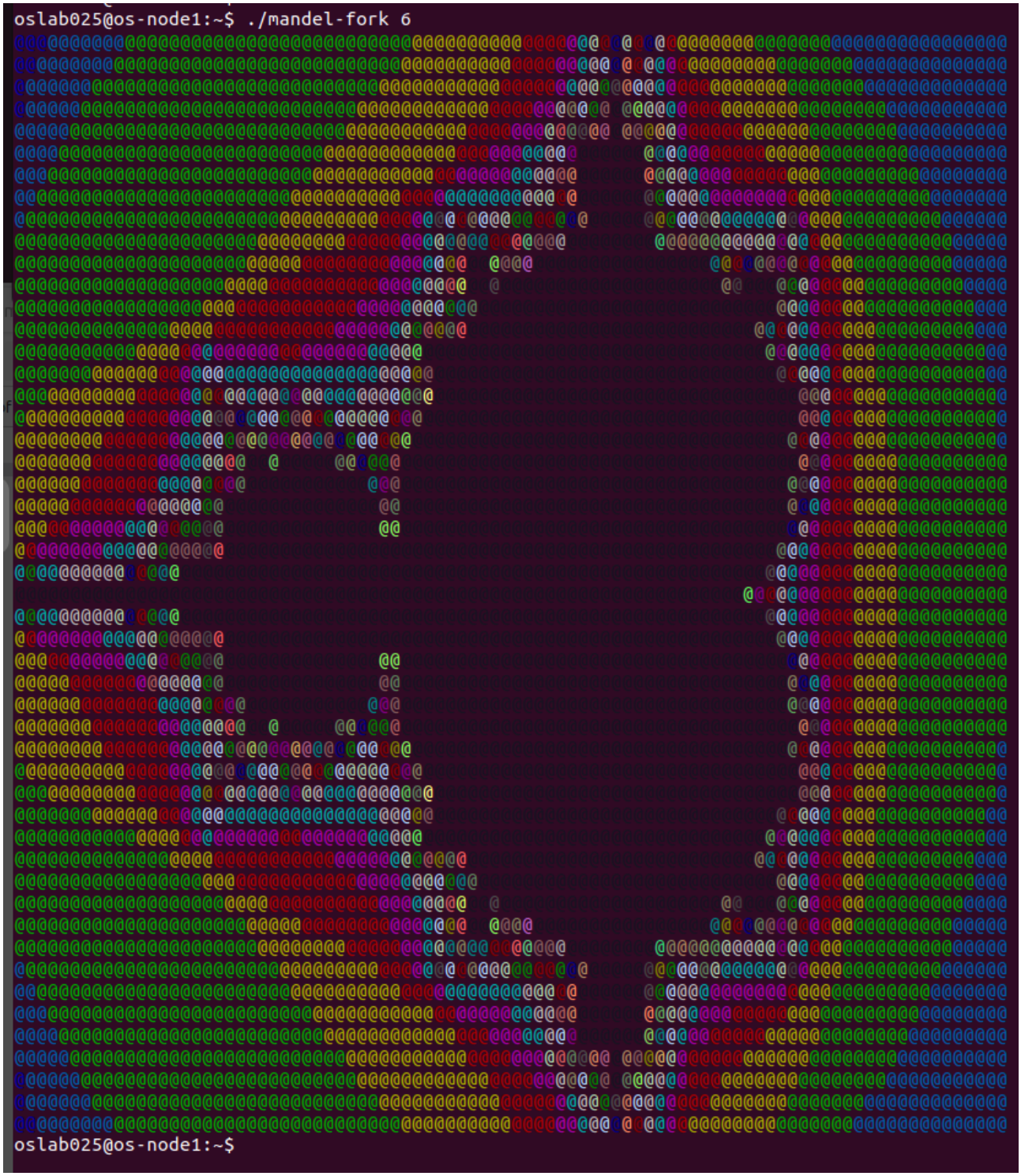
```c
// Handler to reset xterm color and exit on CTRL+C
void sigint_handler(int signum) {
    reset_xterm_color(1);
    exit(1);
}

/*
 * This function computes a line of output
 * as an array of x_char color values.
 */
void compute_mandel_line(int line, int color_val[])
{
    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");
```

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

void process_function(int start_line, int (*shared_buffer)[x_chars]) {
    for (int i = start_line; i < y_chars; i += PPROCESSES) {
        int color_val[x_chars];
        compute_mandel_line(i, color_val);
        memcpy(shared_buffer[i], color_val, x_chars * sizeof(int)); // Copy color_val to shared_buffer
    }
}

/*
 * 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_SHARED | MAP_ANONYMOUS, -1, 0);
    if (addr == MAP_FAILED) {
        perror("mmap Failed");
        exit(EXIT_FAILURE);
    }
    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[])
{
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <number_of_processes>\n", argv[0]);
        exit(1);
    }

    PPROCESSES = atoi(argv[1]);

    if (signal(SIGINT, sigint_handler) == SIG_ERR) {
        perror("signal");
        exit(1);
    }

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

    int (*shared_buffer)[x_chars] = create_shared_memory_area(y_chars * x_chars * sizeof(int));

    pid_t *child_pids = malloc(PPROCESSES * sizeof(pid_t));
    if (child_pids == NULL) {
        perror("Error in allocating memory for child processes");
        exit(1);
    }

    for (int i = 0; i < PPROCESSES; i++) {
        child_pids[i] = fork();
        if (child_pids[i] < 0) {
            perror("Error in creating child process");
            exit(1);
        }
        if (child_pids[i] == 0) {
            process_function(i, shared_buffer);
            exit(0);
        }
    }

    int status;
    for (int i = 0; i < PPROCESSES; i++) {
        wait(&status);
    }

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

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

destroy_shared_memory_area(shared_buffer, y_chars * x_chars * sizeof(int));
free(child_pids);
reset_xterm_color(1);
return 0;
```

Ενδεικτικό output για 6 processes



ΕΡΩΤΗΣΗ 1

Παρατηρούμε ότι όσο αυξάνεται ο αριθμός των διεργασιών, αυξάνεται και η ταχύτητα εκτέλεσής τους. Αυτό είναι αναμενόμενο καθώς η εκτέλεση του κώδικα γίνεται παράλληλα από τις διεργασίες παιδία. Σχετικά με τη σύγκριση μεταξύ νημάτων και διεργασιών, παρατηρούμε ότι ο χρόνος για τις διεργασίες είναι ελαφρώς μεγαλύτερος. Αυτό είναι αναμενόμενο, καθώς όταν κάνουμε fork(), πρέπει να γίνουν πολλαπλές αντιγραφές για να δημιουργήσουμε κάθε διεργασία. Παρόλο που το λειτουργικό σύστημα εφαρμόζει κάποιες βελτιστοποιήσεις, όπως το copy on write, εξακολουθεί να χάνεται χρόνος κατά τη δημιουργία μιας νέας διεργασίας.

1.2.2 Υλοποίηση χωρίς semaphores

Επεξήγηση του κώδικα/ Ερώτηση 1:

Με αυτή την υλοποίηση, δεν χρειάζονται ουσιαστικά σημαφόροι και κλειδιά για τη σειριοποίηση. Δημιουργούμε έναν κοινόχρηστο πίνακα όπου κάθε διεργασία, ανεξάρτητα και χωρίς να εκτελείται σειριακά με τις άλλες, υπολογίζει και καταγράφει τις τιμές στις αντίστοιχες θέσεις. Έτσι, κάθε διεργασία λειτουργεί αυτόνομα, αποφεύγοντας πιθανά σφάλματα στη σειριοποίηση. Αυτή η μέθοδος έχει το μειονέκτημα ότι απαιτεί αρκετό χώρο, καθώς δεσμεύουμε τον χώρο για ολόκληρο τον πίνακα εξ αρχής. Αντίθετα, με σημαφόρους μπορούμε να δεσμεύσουμε πολύ λιγότερο χώρο (συγκεκριμένα Ν πίνακες, αντί για y_chars πίνακες όπως τώρα), αλλά με το κόστος επιπλέον χρόνου λόγω των εντολών κλειδώματος και των αναμονών μεταξύ των διεργασιών-νημάτων. Εάν ο buffer είχε διαστάσεις NPROCS x x_chars, τότε θα ήταν απαραίτητη η σειριοποίηση της εκτύπωσης όπως και πριν. Κάθε διεργασία-νήμα θα υπολόγιζε μια γραμμή, θα περίμενε να εκτυπωθεί και μετά θα προχωρούσε στην επόμενη, και αυτή η σειριοποίηση θα απαιτούσε σημαφόρους ή mutex κτλ.

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

#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"
/* TODO: Include necessary headers for mmap and munmap */

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

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

/*
 * Output at the terminal 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;

int PPROCESSES; // Number of processes

// Handler to reset xterm color and exit on CTRL+C
```

```c
void sigint_handler(int signum) {
    reset_xterm_color(1);
    exit(1);
}

/*
 * This function computes a line of output
 * as an array of x_char color values.
 */
void compute_mandel_line(int line, int color_val[])
{
    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 process_function(int start_line, int (*shared_buffer)[x_chars]) {
    for (int i = start_line; i < y_chars; i += PPROCESSES) {
        int color_val[x_chars];
        compute_mandel_line(i, color_val);
        memcpy(shared_buffer[i], color_val, x_chars * sizeof(int)); // Copy color_val to shared_buffer
    }
}

/*
 * 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_SHARED | MAP_ANONYMOUS, -1, 0);
    if (addr == MAP_FAILED) {
        perror("mmap Failed");
        exit(EXIT_FAILURE);
    }
    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;
```

```
}

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[])
{
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <number_of_processes>\n", argv[0]);
        exit(1);
    }

    PPROCESSES = atoi(argv[1]);

    if (signal(SIGINT, sigint_handler) == SIG_ERR) {
        perror("signal");
        exit(1);
    }

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

    int (*shared_buffer)[x_chars] = create_shared_memory_area(y_chars * x_chars * sizeof(int));

    pid_t *child_pids = malloc(PPROCESSES * sizeof(pid_t));
    if (child_pids == NULL) {
        perror("Error in allocating memory for child processes");
        exit(1);
    }

    for (int i = 0; i < PPROCESSES; i++) {
        child_pids[i] = fork();
        if (child_pids[i] < 0) {
            perror("Error in creating child process");
            exit(1);
        }
        if (child_pids[i] == 0) {
```

```
    }

    for (int i = 0; i < PPROCESSES; i++) {
        child_pids[i] = fork();
        if (child_pids[i] < 0) {
            perror("Error in creating child process");
            exit(1);
        }
        if (child_pids[i] == 0) {
            process_function(i, shared_buffer);
            exit(0);
        }
    }

    int status;
    for (int i = 0; i < PPROCESSES; i++) {
        wait(&status);
    }

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

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

MAKEFILE MANDEL-FORK.C
MAKEFILE2 MMAP.C
MAKEFILE4 MANDEL-FORK-SEM.C