Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
6^{Ο} Εξάμηνο: Λειτουργικά Συστήματα 2022-23
Δανάη Σπέντζου (03120237)
Νεκτάριος Μπούμπαλος (03120441)

Άσκηση 4.1: Κλήσεις συστήματος και βασικοί μηχανισμοί του ΛΣ για τη διαχείριση της εικονικής μνήμης (Virtual Memory – VM)

```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.
         */
        if(0!=raise(SIGSTOP))
                die("raise(SIGSTOP)");
        printf("VM map of child process:\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.
         */
        printf("Physical Adress of private buffer requested by child:%ld\n",get_physical_address((uint64_t)heap_private_buf));

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


        /*
         * Step 9 - Child
         */
        if (0 != raise(SIGSTOP))
                die("raise(SIGSTOP)");
        /*
         * TODO: Write your code here to complete child's part of Step 9.
         */
        int j;
        for(j=0;j<(int)buffer_size;j++){
                heap_private_buf[j]=0;
        }
        printf("VA info of private buffer in child: ");
        show_va_info((uint64_t) heap_private_buf);

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


         /*
          * Step 10 - Child
          */
         if (0 != raise(SIGSTOP))
                 die("raise(SIGSTOP)");
         /*
          * TODO: Write your code here to complete child's part of Step 10.
          */
         for(j=0;j<(int)buffer_size;j++){
                 heap_shared_buf[j]=0;
         }
         printf("VA info of shared buffer in child: ");
         show_va_info((uint64_t) heap_shared_buf);

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

         /*
          * Step 11 - Child
          */
         if (0 != raise(SIGSTOP))
                 die("raise(SIGSTOP)");
         /*
          * TODO: Write your code here to complete child's part of Step 11.
          */
         mprotect(heap_shared_buf, buffer_size, PROT_READ);
         printf("VM map of child:\n");
         show_maps();
         show_va_info((uint64_t)heap_shared_buf);
```

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

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

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

        /*
         * Step 7: Print parent's and child's maps. What do you see?
         * Step 7 - Parent
         */
        printf(RED "\nStep 7: Print parent's and child's map.\n" RESET);
        press_enter();

        /*
         * TODO: Write your code here to complete parent's part of Step 7.
         */

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


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

        printf("Physical address of private buffer requested by parent: %ld\n" ,get_physical_address((uint64_t)heap_private_buf));
        /*
         * TODO: Write your code here to complete parent's part of Step 8.
         */
        printf("VA info of private buffer in parent: ");
        show_va_info((uint64_t)heap_private_buf);

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


        /*
         * 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.
         */

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

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

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


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

        press_enter();

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

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



        /*
         * TODO: Write your code here to complete parent's part of Step 10.
         */

        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: ");
        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.
         */
        munmap(heap_shared_buf,buffer_size);
        munmap(heap_private_buf,buffer_size);
        munmap(file_shared_buf,buffer_size);
}
```

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

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

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

        /*
         * Step 2: Use mmap to allocate a buffer of 1 page and print the map
         * again. Store buffer in heap_private_buf.
         */
        printf(RED "\nStep 2: Use mmap(2) to allocate a private buffer of "
                "size equal to 1 page and print the VM map again.\n" RESET);
        press_enter();
        /*
         * TODO: Write your code here to complete Step 2.
         */
        heap_private_buf=mmap(NULL,buffer_size,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,fd,0);

        if(heap_private_buf==MAP_FAILED)
                die("mmap");

        show_maps();
        show_va_info((uint64_t)heap_private_buf);
        /*
         * Step 3: Find the physical address of the first page of your buffer
         * in main memory. What do you see?
         */
        printf(RED "\nStep 3: Find and print the physical address of the "
                "buffer in main memory. What do you see?\n" RESET);
        press_enter();
        /*
         * TODO: Write your code here to complete Step 3.
         */
        printf("Physical address:%ld\n",get_physical_address((uint64_t)heap_private_buf));
        /*
         * Step 4: Write zeros to the buffer and repeat Step 3.
         */
        printf(RED "\nStep 4: Initialize your buffer with zeros and repeat "
                "Step 3. What happened?\n" RESET);
        press_enter();
        /*
         * TODO: Write your code here to complete Step 4.
         */
        int i;
        for(i=0;i<(int)buffer_size;i++)
        {
                heap_private_buf[i]=0;
        }
        printf("Physical adress:%ld\n",get_physical_address((uint64_t)heap_private_buf));


        /*
         * Step 5: Use mmap(2) to map file.txt (memory-mapped files) and print
         * its content. Use file_shared_buf.
         */
        printf(RED "\nStep 5: Use mmap(2) to read and print file.txt. Print "
                "the new mapping information that has been created.\n" RESET);
        press_enter();
        /*
         * TODO: Write your code here to complete Step 5.
         */

        fd=open("file.txt",O_RDONLY);
        if(fd==-1)
                die("open");

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

        if(file_shared_buf==MAP_FAILED)
                die("mmap");

        char c;

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

        show_maps();
        show_va_info((uint64_t)file_shared_buf);
        /*
         * Step 6: Use mmap(2) to allocate a shared buffer of 1 page. Use
         * heap_shared_buf.
         */
        printf(RED "\nStep 6: Use mmap(2) to allocate a shared buffer of size "
                "equal to 1 page. Initialize the buffer and print the new "
                "mapping information that has been created.\n" RESET);
        press_enter();
        /*
         * 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_private_buf==MAP_FAILED)
                die("mmap");

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

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



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

        parent(p);

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

1. Τυπώνουμε το χάρτη της εικονικής μνήμης της τρέχουσας διεργασίας.

```
oslab162@orion:~/exercise4/mmap$ ./mmap
Step 1: Print the virtual address space map of this process [12600].

Virtual Memory Map of process [12600]:
00400000-00403000 r-xp 00000000 fe:10 8923153                            /store/homes/oslab/oslab162/exercise4/mmap/mmap
00602000-00603000 rw-p 00002000 fe:10 8923153                            /store/homes/oslab/oslab162/exercise4/mmap/mmap
02249000-0226a000 rw-p 00000000 00:00 0                                  [heap]
7f592f90f000-7f592fab0000 r-xp 00000000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f592fab0000-7f592fcb0000 ---p 001a1000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f592fcb0000-7f592fcb4000 r--p 001a1000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f592fcb4000-7f592fcb6000 rw-p 001a5000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f592fcb6000-7f592fcba000 rw-p 00000000 00:00 0
7f592fcba000-7f592fcda000 r-xp 00000000 fe:01 1044705                    /lib/x86_64-linux-gnu/ld-2.19.so
7f592fecd000-7f592fed0000 rw-p 00000000 00:00 0
7f592fed5000-7f592feda000 rw-p 00000000 00:00 0
7f592feda000-7f592fedb000 r--p 00020000 fe:01 1044705                    /lib/x86_64-linux-gnu/ld-2.19.so
7f592fedb000-7f592fedc000 rw-p 00021000 fe:01 1044705                    /lib/x86_64-linux-gnu/ld-2.19.so
7f592fedc000-7f592fedd000 rw-p 00000000 00:00 0
7ffea29d2000-7ffea29f3000 rw-p 00000000 00:00 0                          [stack]
7ffea29fa000-7ffea29fc000 r-xp 00000000 00:00 0                          [vdso]
7ffea29fc000-7ffea29fe000 r--p 00000000 00:00 0                          [vvar]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

2. Με την κλήση συστήματος `mmap()` δεσμεύουμε buffer(προσωρινή μνήμη) μεγέθους μίας σελίδας (page) και τυπώνουμε ξανά το χάρτη. Εντοπίζουμε στον χάρτη μνήμης τον χώρο εικονικών διευθύνσεων που δεσμεύσαμε.

```
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 [12600]:
00400000-00403000 r-xp 00000000 fe:10 8923153                            /store/homes/oslab/oslab162/exercise4/mmap/mmap
00602000-00603000 rw-p 00002000 fe:10 8923153                            /store/homes/oslab/oslab162/exercise4/mmap/mmap
02249000-0226a000 rw-p 00000000 00:00 0                                  [heap]
7f592f90f000-7f592fab0000 r-xp 00000000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f592fab0000-7f592fcb0000 ---p 001a1000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f592fcb0000-7f592fcb4000 r--p 001a1000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f592fcb4000-7f592fcb6000 rw-p 001a5000 fe:01 1045760                    /lib/x86_64-linux-gnu/libc-2.19.so
7f592fcb6000-7f592fcba000 rw-p 00000000 00:00 0
7f592fcba000-7f592fcda000 r-xp 00000000 fe:01 1044705                    /lib/x86_64-linux-gnu/ld-2.19.so
7f592fecd000-7f592fed0000 rw-p 00000000 00:00 0
7f592fed4000-7f592feda000 rw-p 00000000 00:00 0
7f592feda000-7f592fedb000 r--p 00020000 fe:01 1044705                    /lib/x86_64-linux-gnu/ld-2.19.so
7f592fedb000-7f592fedc000 rw-p 00021000 fe:01 1044705                    /lib/x86_64-linux-gnu/ld-2.19.so
7f592fedc000-7f592fedd000 rw-p 00000000 00:00 0
7ffea29d2000-7ffea29f3000 rw-p 00000000 00:00 0                          [stack]
7ffea29fa000-7ffea29fc000 r-xp 00000000 00:00 0                          [vdso]
7ffea29fc000-7ffea29fe000 r--p 00000000 00:00 0                          [vvar]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------

7f592fed4000-7f592feda000 rw-p 00000000 00:00 0
```

3. Προσπαθούμε να βρούμε και να τυπώσουμε τη φυσική διεύθυνση μνήμης στην οποία απεικονίζεται η εικονική διεύθυνση του buffer (τη διεύθυνση όπου βρίσκεται αποθηκευμένος στη φυσική κύρια μνήμη).

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

VA[0x7fc634c8c000] is not mapped; no physical memory allocated.
Physical address:0
```

Παρατηρούμε, πως παρ' όλο που η μνήμη έχει δεσμευθεί, όπως φαίνεται και στον χάρτη μνήμης, η απεικόνιση της εικονικής μνήμης στην φυσική δεν έχει γίνει ακόμα αφού η φυσική δεσμεύεται **on demand** από το ΛΣ όταν πάει να προσπελαστεί. Έτσι, σε περίπτωση προσπέλασης της εικονικής μνήμης γίνεται **page fault** και αφού διαπιστωθεί ότι η εικονική μνήμη βρίσκεται στον χάρτη μνήμης τότε το ΛΣ επιλέγει ένα frame για την απεικόνιση του.

4. Γεμίζουμε με μηδενικά τον buffer και επαναλαμβάνουμε το Βήμα 3. Παρατηρούμε ότι η απεικόνιση γίνεται στην φυσική μνήμη όπως αναλύσαμε και παραπάνω.

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

Physical adress:5663563776
```

5. Χρησιμοποιούμε την `mmap()` για να απεικονίσουμε (memorymap) το αρχείο file.txt στον χώρο διευθύνσεων της διεργασίας μας και να τυπώσουμε το περιεχόμενό του. Εντοπίζουμε τη νέα απεικόνιση (mapping) στον χάρτη μνήμης.

```
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 [575347]:
00400000-00403000 r-xp 00000000 00:26 8914372                            /home/oslab/oslab162/exercise4/mmap/mmap
00602000-00603000 rw-p 00002000 00:26 8914372                            /home/oslab/oslab162/exercise4/mmap/mmap
02529000-0254a000 rw-p 00000000 00:00 0                                  [heap]
7efec4019000-7efec403b000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efec403b000-7efec4194000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efec4194000-7efec41e3000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efec41e3000-7efec41e7000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efec41e7000-7efec41e9000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efec41e9000-7efec41ef000 rw-p 00000000 00:00 0
7efec41f4000-7efec41f5000 r--s 00000000 00:26 8923147                    /home/oslab/oslab162/exercise4/mmap/file.txt
7efec41f5000-7efec41f6000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efec41f6000-7efec4216000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efec4216000-7efec421e000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efec421e000-7efec421f000 rw-p 00000000 00:00 0
7efec421f000-7efec4220000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efec4220000-7efec4221000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efec4221000-7efec4222000 rw-p 00000000 00:00 0
7ffcf3185000-7ffcf31a6000 rw-p 00000000 00:00 0                          [stack]
7ffcf31f8000-7ffcf31fc000 r--p 00000000 00:00 0                          [vvar]
7ffcf31fc000-7ffcf31fe000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------------

7efec41f4000-7efec41f5000 r--s 00000000 00:26 8923147                    /home/oslab/oslab162/exercise4/mmap/file.txt
```

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

```
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 [575347]:
00400000-00403000 r-xp 00000000 00:26 8914372                            /home/oslab/oslab162/exercise4/mmap/mmap
00602000-00603000 rw-p 00002000 00:26 8914372                            /home/oslab/oslab162/exercise4/mmap/mmap
02529000-0254a000 rw-p 00000000 00:00 0                                  [heap]
7efec4019000-7efec403b000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efec403b000-7efec4194000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efec4194000-7efec41e3000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efec41e3000-7efec41e7000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efec41e7000-7efec41e9000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efec41e9000-7efec41ef000 rw-p 00000000 00:00 0
7efec41f3000-7efec41f4000 rw-s 00000000 00:01 1822                       /dev/zero (deleted)
7efec41f4000-7efec41f5000 r--s 00000000 00:26 8923147                    /home/oslab/oslab162/exercise4/mmap/file.txt
7efec41f5000-7efec41f6000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efec41f6000-7efec4216000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efec4216000-7efec421e000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efec421e000-7efec421f000 rw-p 00000000 00:00 0
7efec421f000-7efec4220000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efec4220000-7efec4221000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efec4221000-7efec4222000 rw-p 00000000 00:00 0
7ffcf3185000-7ffcf31a6000 rw-p 00000000 00:00 0                          [stack]
7ffcf31f8000-7ffcf31fc000 r--p 00000000 00:00 0                          [vvar]
7ffcf31fc000-7ffcf31fe000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------------

7efec41f3000-7efec41f4000 rw-s 00000000 00:01 1822                       /dev/zero (deleted)
```

Στο σημείο αυτό καλείται η συνάρτηση `fork()` και δημιουργείται μια νέα διεργασία.

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

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

VM map of parent process:

Virtual Memory Map of process [319777]:
00400000-00403000 r-xp 00000000 00:26 8914402                            /home/oslab/oslab162/exercise4/mmap/mmap
00602000-00603000 rw-p 00002000 00:26 8914402                            /home/oslab/oslab162/exercise4/mmap/mmap
01c24000-01c45000 rw-p 00000000 00:00 0                                  [heap]
7f9b48f6d000-7f9b48f8f000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b48f8f000-7f9b490e8000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b490e8000-7f9b49137000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b49137000-7f9b4913b000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b4913b000-7f9b4913d000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b4913d000-7f9b49143000 rw-p 00000000 00:00 0
7f9b49146000-7f9b49147000 rw-s 00000000 00:01 12518                      /dev/zero (deleted)
7f9b49147000-7f9b49148000 r--s 00000000 00:26 8923147                    /home/oslab/oslab162/exercise4/mmap/file.txt
7f9b49148000-7f9b49149000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49149000-7f9b49169000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49169000-7f9b49171000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49171000-7f9b49172000 rw-p 00000000 00:00 0
7f9b49172000-7f9b49173000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49173000-7f9b49174000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49174000-7f9b49175000 rw-p 00000000 00:00 0
7ffe5c392000-7ffe5c3b3000 rw-p 00000000 00:00 0                          [stack]
7ffe5c3b9000-7ffe5c3bd000 r--p 00000000 00:00 0                          [vvar]
7ffe5c3bd000-7ffe5c3bf000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------------

VM map of child process:

Virtual Memory Map of process [319778]:
00400000-00403000 r-xp 00000000 00:26 8914402                            /home/oslab/oslab162/exercise4/mmap/mmap
00602000-00603000 rw-p 00002000 00:26 8914402                            /home/oslab/oslab162/exercise4/mmap/mmap
01c24000-01c45000 rw-p 00000000 00:00 0                                  [heap]
7f9b48f6d000-7f9b48f8f000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b48f8f000-7f9b490e8000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b490e8000-7f9b49137000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b49137000-7f9b4913b000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b4913b000-7f9b4913d000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b4913d000-7f9b49143000 rw-p 00000000 00:00 0
7f9b49146000-7f9b49147000 rw-s 00000000 00:01 12518                      /dev/zero (deleted)
7f9b49147000-7f9b49148000 r--s 00000000 00:26 8923147                    /home/oslab/oslab162/exercise4/mmap/file.txt
7f9b49148000-7f9b49149000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49149000-7f9b49169000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49169000-7f9b49171000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49171000-7f9b49172000 rw-p 00000000 00:00 0
7f9b49172000-7f9b49173000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49173000-7f9b49174000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49174000-7f9b49175000 rw-p 00000000 00:00 0
7ffe5c392000-7ffe5c3b3000 rw-p 00000000 00:00 0                          [stack]
7ffe5c3b9000-7ffe5c3bd000 r--p 00000000 00:00 0                          [vvar]
7ffe5c3bd000-7ffe5c3bf000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------------
```

Παρατηρούμε, πως η νέα διεργασία που δημιουργείται μέσω της fork() ως αντίγραφο της παλιάς, κληρονομεί ένα αντίγραφο της **εικονικής μνήμης** και ένα του πίνακα σελίδων (***page table***) της αρχικής διεργασίας (και από τα δύο αφαιρούνται τα write δικαιώματα στις σελίδες private-COW)

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

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

Physical Address of private buffer requested by parent: 5754757120
VA info of private buffer in parent: 7f9b49171000-7f9b49172000 rw-p 00000000 00:00 0
Physical Address of private buffer requested by child: 5754757120
VA info of private buffer in child: 7f9b49171000-7f9b49172000 rw-p 00000000 00:00 0
```

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?

VA info of private buffer in parent: 7efec421e000-7efec421f000 rw-p 00000000 00:00 0
Physical Address of private buffer requested by parent: 7595442176
Physical Adress of private buffer requested by child:7595442176
VA info of private buffer in child: 7efec421e000-7efec421f000 rw-p 00000000 00:00 0
```

Σε συστήματα εικονικής μνήμης, όταν δημιουργείται μία διεργασία μέσω fork(), αντιγράφεται μόνο η εικονική μνήμη και ο πίνακας σελίδων χωρίς τα write δικαιώματα σε private pages και μόνο σε περίπτωση προσπάθειας write σε private σελίδα η απεικόνιση της εικονικής διεύθυνσης στην φυσική διεύθυνση αλλάζει και το ΛΣ βρίσκει ένα νέο πλαίσιο για την απεικόνιση της, στο οποίο αντιγράφεται όλο το περιεχόμενο του page και ενημερώνεται ο πίνακας σελίδων [**Copy-on- Write (COW)**]. Επομένως, όταν γράφουμε στον private buffer από τη διεργασία παιδί τότε οι φυσικές διευθύνσεις είναι διαφορετικές.

10. Γράφουμε στον sharedbuffer (Βήμα6) από τη διεργασία παιδί και τυπώνουμε τη φυσική του διεύθυνση για τις διεργασίες πατέρα και παιδί.

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

VA info of shared buffer in parent: 7f9b49146000-7f9b49147000 rw-s 00000000 00:01 12518                    /dev/zero (deleted)
Physical Address of shared buffer requested by parent: 5124722688
VA info of shared buffer in child: 7f9b49146000-7f9b49147000 rw-s 00000000 00:01 12518                    /dev/zero (deleted)
Physical Address of shared buffer requested by child: 5124722688
```

Σε σύγκριση με τον private buffer, ο shared buffer απεικονίζεται στην ίδια φυσική διεύθυνση και για τις δύο διεργασίες, αφού η σελίδα είναι shared μεταξύ των διεργασιών (**MAP_SHARED** flag στην mmap()) και άρα αντιστοιχίζεται στην ίδια φυσική διεύθυνση. Αυτό επιτρέπει την διαδιεργασιακή επικοινωνία αφού "μοιράζονται" την ίδια μνήμη.

11. Απαγορεύουμε τις εγγραφές στον shared buffer για τη διεργασία παιδί. Εντοπίζουμε και τυπώνουμε την απεικόνιση του shared buffer στο χάρτη μνήμης των δύο διεργασιών για να επιβεβαιώσουμε την απαγόρευση και παρατηρούμε πως έχει αφαιρεθεί το write δικαίωμα από την διεργασία παιδί (**segfault**).

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

VM map of parent
Virtual Memory Map of process [319777]:
00400000-00403000 r-xp 00000000 00:26 8914402                            /home/oslab/oslab162/exercise4/mmap/mmap
00602000-00603000 rw-p 00002000 00:26 8914402                            /home/oslab/oslab162/exercise4/mmap/mmap
01c24000-01c45000 rw-p 00000000 00:00 0                                  [heap]
7f9b48f6d000-7f9b48f8f000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b48f8f000-7f9b490e8000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b490e8000-7f9b49137000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b49137000-7f9b4913b000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b4913b000-7f9b4913d000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b4913d000-7f9b49143000 rw-p 00000000 00:00 0
7f9b49146000-7f9b49147000 rw-s 00000000 00:01 12518                      /dev/zero (deleted)
7f9b49147000-7f9b49148000 r--s 00000000 00:26 8923147                    /home/oslab/oslab162/exercise4/mmap/file.txt
7f9b49148000-7f9b49149000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49149000-7f9b49169000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49169000-7f9b49171000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49171000-7f9b49172000 rw-p 00000000 00:00 0
7f9b49172000-7f9b49173000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49173000-7f9b49174000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49174000-7f9b49175000 rw-p 00000000 00:00 0
7ffe5c392000-7ffe5c3b3000 rw-p 00000000 00:00 0                          [stack]
7ffe5c3b9000-7ffe5c3bd000 r--p 00000000 00:00 0                          [vvar]
7ffe5c3bd000-7ffe5c3bf000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------------

7f9b49146000-7f9b49147000 rw-s 00000000 00:01 12518                      /dev/zero (deleted)
VM map of child:

Virtual Memory Map of process [319778]:
00400000-00403000 r-xp 00000000 00:26 8914402                            /home/oslab/oslab162/exercise4/mmap/mmap
00602000-00603000 rw-p 00002000 00:26 8914402                            /home/oslab/oslab162/exercise4/mmap/mmap
01c24000-01c45000 rw-p 00000000 00:00 0                                  [heap]
7f9b48f6d000-7f9b48f8f000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b48f8f000-7f9b490e8000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b490e8000-7f9b49137000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b49137000-7f9b4913b000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b4913b000-7f9b4913d000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9b4913d000-7f9b49143000 rw-p 00000000 00:00 0
7f9b49146000-7f9b49147000 r--s 00000000 00:01 12518                      /dev/zero (deleted)
7f9b49147000-7f9b49148000 r--s 00000000 00:26 8923147                    /home/oslab/oslab162/exercise4/mmap/file.txt
7f9b49148000-7f9b49149000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49149000-7f9b49169000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49169000-7f9b49171000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49171000-7f9b49172000 rw-p 00000000 00:00 0
7f9b49172000-7f9b49173000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49173000-7f9b49174000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9b49174000-7f9b49175000 rw-p 00000000 00:00 0
7ffe5c392000-7ffe5c3b3000 rw-p 00000000 00:00 0                          [stack]
7ffe5c3b9000-7ffe5c3bd000 r--p 00000000 00:00 0                          [vvar]
7ffe5c3bd000-7ffe5c3bf000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------------

7f9b49146000-7f9b49147000 r--s 00000000 00:01 12518                      /dev/zero (deleted)
```

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

Άσκηση 4.2: Παράλληλος υπολογισμός Mandelbrot με διεργασίες αντί για νήματα

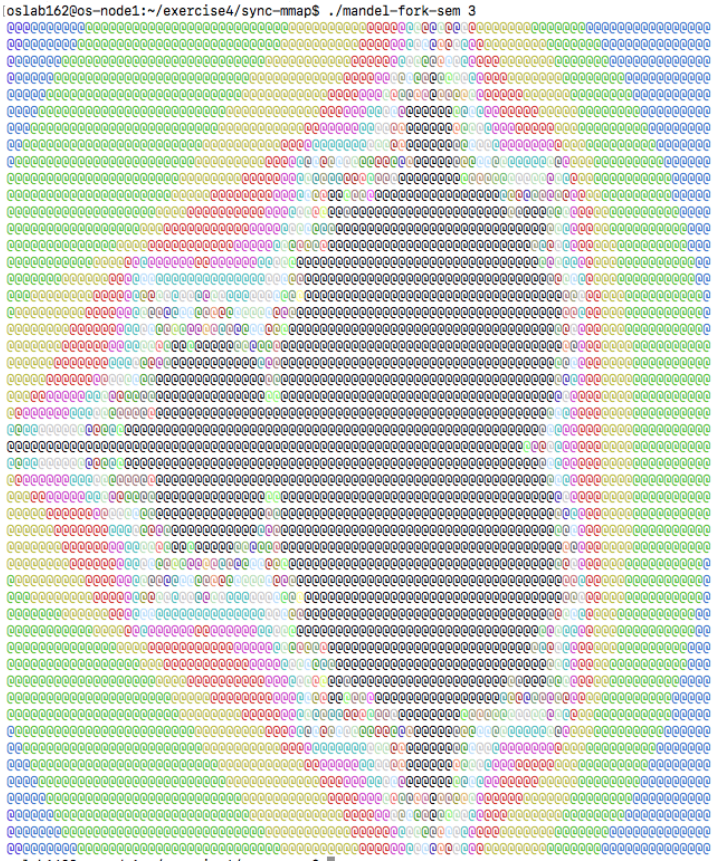
4.2.1: Semaphores πάνω από διαμοιραζόμενη μνήμη

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

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

#include "../helpers/mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

sem_t *sem;

/*
 * 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;

//helping functions
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;
}

/*
 * 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);
        }
}
```

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


/*
 * Catch SIGINT (Ctrl-C) with the sigint_handler to ensure the prompt is not
 * drawn in a funny colour if the user "terminates" the execution with Ctrl-C.
 */
void sigint_handler(int signum)
{
        reset_xterm_color(1);
        exit(1);
}

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

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

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

        /* Create a shared, anonymous mapping for this number of pages */
        addr=mmap(NULL, pages * sysconf(_SC_PAGE_SIZE), PROT_READ | PROT_WRITE,
                 MAP_SHARED | MAP_ANONYMOUS, -1,0);
        if(addr==MAP_FAILED) {
                perror("mmap");
                exit(1);
        }
        return addr;
}

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

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

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

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


void fork_execute(int line, int procnt)
{       //same as ex3
        int line_num;
        int color_val[x_chars];
        for (line_num=line; line_num<y_chars; line_num+=procnt) {
                compute_mandel_line(line_num, color_val);
                if(sem_wait(&sem[line])<0) {
                        perror("sem_wait");
                        exit(1);
                }
                output_mandel_line(1, color_val);
                if(sem_post(&sem[(line_num+1) % procnt])<0) {
                        perror("sem_post");
                        exit(1);
                }
        }
}

int main(int argc, char *argv[])
{
        int i, procnt, status;

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

        /*
         * signal handling
         */
        struct sigaction sa;
        sa.sa_handler=sigint_handler;
        sa.sa_flags=0;
        sigemptyset(&sa.sa_mask);
        if(sigaction(SIGINT, &sa, NULL)<0) {
                perror("sigaction");
                exit(1);
        }

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

        sem=create_shared_memory_area(procnt * sizeof(sem_t)); //allocate shared mem to store
                                                               //semaphore array
```

```
    for (i=0; i<procnt; i++) {
            if(sem_init(&sem[i],1,0)<0) {
                    perror("sem_init");
                    exit(1);
            }
    }

    if(sem_post(&sem[0])<0) {
            perror("sem_post");
            exit(1);
    }

    /*create the processes and call the execution function*/
    pid_t child_pid;

     for(i=0; i<procnt; i++) {
             child_pid=fork();
             if(child_pid< 0) {
                     perror("error with creation of child");
                     exit(1);
             }
             if(child_pid== 0) {
                     fork_execute(i, procnt);
                     exit(1);
             }
     }
     for(i=0; i<procnt ; i++) {
             child_pid= wait(&status);
     }

     for(i=0; i<procnt ; i++) {
           sem_destroy(&sem[i]);
    }

     destroy_shared_memory_area(sem, procnt * sizeof(sem_t));

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

```
oslab162@os-node1:~/exercise4/sync-mmap$ ./mandel-fork-sem 3
```



Περιμένουμε καλύτερη απόδοση να έχει η υλοποίηση με τα threads, διότι η δημιουργία διεργασιών λόγω των απαραίτητων ενεργειών που απαιτούνται, όπως η δημιουργία των PCB's, η αφαίρεση δικαιώματος write από τα pages που έχει δεσμεύσει η γονική διεργασία για να υποστηριχτεί το Copy-on-Write κ.λπ. απαιτούν περισσότερο χρόνο. Παράλληλα, διαφορά εντοπίζεται και στην εναλλαγή, αφού στις διεργασίες για να γίνει το context switch πρέπει να αντικατασταθεί το PCB εκτός από την αποθήκευση του PC και των registers. Τέλος, τα threads έχουν κοινή μνήμη, ενώ τα processes πρέπει χρησιμοποιούν την συνάρτηση create_shared_memory_area() για το mmap().

4.2.2: Υλοποίηση χωρίς semaphores

```
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include<errno.h>
#include<signal.h>
#include<sys/mman.h>
#include<sys/wait.h>


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

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

/***************************
 * Compile-time parameters *
 ***************************/
int **buff;

/*
 * 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);
        }
}
```

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

        compute_mandel_line(line, color_val);
        output_mandel_line(fd, color_val);
}

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

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

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

        /*
         * Determine the number of pages needed, round up the requested number of

         * pages
         */

        pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;

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


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

        if(addr==MAP_FAILED){
                perror("mmap");
                exit(1);
        }

        return addr;
}

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

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

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

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

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

void fork_execute(int line, int procnt)
{
        int line_num;
        for(line_num=line;line_num<y_chars;line_num+=procnt){
                compute_mandel_line(line_num,buff[line_num]);
        }
        return;
}

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;
}

int main(int argc,char *argv[])
{
        int i , procnt, status;

        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],&procnt)<0||procnt<=0){
                fprintf(stderr,"%s is not valid for 'processes_count'\n",argv[1]);
                exit(1);
        }

        struct sigaction sa;
        sa.sa_handler=sigint_handler;
        sa.sa_flags=0;
        sigemptyset(&sa.sa_mask);
        if(sigaction(SIGINT, &sa, NULL)<0){
                perror("sigaction");
                exit(1);
        }

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

        pid_t child_pid;
        for(i=0;i<procnt;i++){
                child_pid=fork();
                if(child_pid<0){
                        perror("error with creation of child");
                        exit(1);
                }
                if(child_pid==0){
                        fork_execute(i,procnt);
                        exit(1);
                }
        }
        for(i=0;i<procnt;i++){
                child_pid=wait(&status);
        }
        for(i=0;i<y_chars;i++){
                output_mandel_line(1,buff[i]);
        }
        for(i=0;i<y_chars;i++){
                destroy_shared_memory_area(buff[i],sizeof(buff[i]));
        }
        destroy_shared_memory_area(buff,sizeof(buff));
        reset_xterm_color(1);
        return 0;
}
```

```
oslab162@os-node1:~/exercise4/sync-mmap$ ./mandel-fork 3
```

Ο συγχρονισμός επιτυγχάνεται μέσα από το γεγονός ότι κάθε διεργασία είναι ανεξάρτητη και γράφει στον buffer μόνο στα σημεία που της αναλογούν δημιουργώντας ταυτόχρονα το output. Έτσι, δεν υπάρχει κάποιο κρίσιμο τμήμα, αφού για το output είναι υπεύθυνη η αρχική διεργασία.

Αν ο buffer είχε μικρότερες διαστάσεις **NPROCS x x_chars** αντί για **y_chars x x_chars** τότε δεν θα μπορούσαμε να τυπώσουμε όλο το output αλλά θα κάναμε τμηματική εκτύπωση με μικρότερα outputs το οποίο θα μπορούσε να επιτευχθεί με τη χρήση σημάτων. Συγκεκριμένα, κάθε φορά που μια διεργασία υπολογίζει την γραμμή της θα μπορούσε να κάνει *raise(SIGSTOP)* και αυτό να επαναλαμβάνεται για όλες τις διεργασίες παιδιά. Όταν αυτά τελειώσουν η αρχική διεργασία να τυπώνει το αντίστοιχο buffer και να τις ενεργοποιηεί μία μία, επαναλαμβάνοντας την παραπάνω διαδικασία μέχρι να υπολογιστεί η έξοδος.