Λειτουργικά Συστήματα
4η Εργαστηριακή Αναφορά

Μηχανισμοί εικονικής μνήμης

6ο εξάμηνο, Ακαδημαϊκή περίοδος 2022-2023

Ομάδα: oslab35

Μέλη

Καμπουγέρης Χαράλαμπος | A.M: el20098

Κουστένης Χρίστος | A.M: el20227

1.1(Κλήσεις συστήματος και βασικοί μηχανισμοί του ΛΣ για την διαχείριση της εικονικής μνήμης)

Ο πηγαίος κώδικας της άσκησης είναι ο εξής:

```c
#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 <sys/mman.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 has the following vm map : ");
    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 address of private heap buffer on child: %lu\n",
get_physical_address((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;
    }
    show_va_info((uint64_t)heap_private_buf);
    printf("Physical address of private heap buffer on child: %lu\n",
get_physical_address((uint64_t)heap_private_buf));


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


    for (int i = 0; i < (int)buffer_size; i++)
    {
        heap_shared_buf[i] = 1;
    }
    show_va_info((uint64_t)heap_shared_buf);
    printf("Physical address of shared heap buffer on child: %lu\n",
get_physical_address((uint64_t)heap_shared_buf));


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

```
     */

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

    mprotect(heap_shared_buf, buffer_size, PROT_READ);
    printf("VM map of child:\n");
    show_maps();
    show_va_info((uint64_t)heap_shared_buf);
    /*
     * TODO: Write your code here to complete child's part of Step 11.
     */

    /*
     * 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);

    printf("Parent has the following vm map : ");
    show_maps();
    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();

    /*
     * TODO: Write your code here to complete parent's part of Step 8.
     */
    show_va_info(heap_private_buf);
    printf("Physical address of private heap buffer on father: %lu\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 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();

    for (int i = 0; i < (int)buffer_size; i++)
    {
        heap_private_buf[i] = 1;
    }
    show_va_info((uint64_t)heap_private_buf);
```

```
    printf("Physical address of private heap buffer on father: %lu\n",
get_physical_address((uint64_t)heap_private_buf));
    /*
     * TODO: Write your code here to complete parent's part of Step 9.
     */

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

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

    for (int i = 0; i < (int)buffer_size; i++)
    {
        heap_shared_buf[i] = 1;
    }
    show_va_info((uint64_t)heap_shared_buf);
    printf("Physical address of shared heap buffer on father: %lu\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(RED "\nStep 11: Disable writing on the shared buffer for the "
           "child. Verify through the maps for the parent and the "
           "child.\n" RESET);
press_enter();

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

if (-1 == kill(child_pid, SIGCONT))
    die("kill");
if (-1 == waitpid(child_pid, &status, 0))
```

```
        die("waitpid");

    /*
     * Step 12: Free all buffers for parent and child.
     * Step 12 - Parent
     */
    munmap(heap_shared_buf, buffer_size);
    munmap(heap_private_buf, buffer_size);
    munmap(file_shared_buf, buffer_size);
    /*
     * TODO: Write your code here to complete parent's part of Step 12.
     */
}

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.
     */
    int prot = PROT_WRITE | PROT_READ;
    int flags = MAP_ANONYMOUS | MAP_PRIVATE;
    heap_private_buf = mmap(NULL, buffer_size, prot, flags, 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.
     */
    pa = get_physical_address((uint64_t)heap_private_buf);
    printf("Physical address of private heap is: %lu\n", pa);

    /*********************************************************************
*********************************/

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

```
    pa = get_physical_address((uint64_t)heap_private_buf);
    printf("Physical address of private heap is: %lu\n", pa);

    /*****************************************************************
*************************************************/


    /*
     * 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");

    if (write(1, file_shared_buf, (size_t)buffer_size) !=
(ssize_t)buffer_size)
        die("write");
    show_maps();
```

```
    show_va_info((uint64_t)file_shared_buf);
    pa = get_physical_address((uint64_t)file_shared_buf);
    printf("Physical address  of file shared heap is : %lu", pa);
    /*****************************************************************
*****************************************************************/


    /*
     * 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.
     */
    prot = PROT_READ | PROT_WRITE;
    flags = MAP_ANONYMOUS | MAP_SHARED;
    heap_shared_buf = mmap(NULL, buffer_size, prot, flags, -1, 0);

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

    for (int i = 0; i < (int)buffer_size; i++)
    {
        heap_shared_buf[i] = 1;
    }
    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:

```
oslab35@os-node2:~/os-lab-exer4/mmap$ ./mmap

Step 1: Print the virtual address space map of this process [415459].


Virtual Memory Map of process [415459]:
55fd33d79000-55fd33d7a000 r--p 00000000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7a000-55fd33d7c000 r-xp 00001000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7c000-55fd33d7d000 r--p 00003000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7d000-55fd33d7e000 r--p 00003000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7e000-55fd33d7f000 rw-p 00004000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd35283000-55fd352a4000 rw-p 00000000 00:00 0                          [heap]
7f472bbb7000-7f472bbd9000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bbd9000-7f472bd32000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd32000-7f472bd81000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd81000-7f472bd85000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd85000-7f472bd87000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd87000-7f472bd8d000 rw-p 00000000 00:00 0
7f472bd92000-7f472bd93000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bd93000-7f472bdb3000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdb3000-7f472bdbb000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbc000-7f472bdbd000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbd000-7f472bdbe000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbe000-7f472bdbf000 rw-p 00000000 00:00 0
7fff6d440000-7fff6d461000 rw-p 00000000 00:00 0                          [stack]
7fff6d478000-7fff6d47c000 r--p 00000000 00:00 0                          [vvar]
7fff6d47c000-7fff6d47e000 r-xp 00000000 00:00 0                          [vdso]
-------------------------------------------------------
```

2. Δεσμεύουμε buffer μεγέθους μίας σελίδας (page) με την χρήση της mmap() (απεικόνιση σωρού) και τυπώνουμε ξανά το χάρτη:

```
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 [415459]:
55fd33d79000-55fd33d7a000 r--p 00000000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7a000-55fd33d7c000 r-xp 00001000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7c000-55fd33d7d000 r--p 00003000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7d000-55fd33d7e000 r--p 00003000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7e000-55fd33d7f000 rw-p 00004000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd35283000-55fd352a4000 rw-p 00000000 00:00 0                          [heap]
7f472bbb7000-7f472bbd9000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bbd9000-7f472bd32000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd32000-7f472bd81000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd81000-7f472bd85000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd85000-7f472bd87000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd87000-7f472bd8d000 rw-p 00000000 00:00 0
7f472bd92000-7f472bd93000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bd93000-7f472bdb3000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdb3000-7f472bdbb000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbb000-7f472bdbc000 rw-p 00000000 00:00 0
7f472bdbc000-7f472bdbd000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbd000-7f472bdbe000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbe000-7f472bdbf000 rw-p 00000000 00:00 0
7fff6d440000-7fff6d461000 rw-p 00000000 00:00 0                          [stack]
7fff6d478000-7fff6d47c000 r--p 00000000 00:00 0                          [vvar]
7fff6d47c000-7fff6d47e000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------

7f472bdbb000-7f472bdbc000 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:

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

VA[0x7f472bdbb000] is not mapped; no physical memory allocated.
Physical address of private heap is: 0
```

Παρατηρούμε ότι ενώ έχει δεσμευθεί η μνήμη δεν έχει γίνει ακόμα η απεικόνιση της εικονικής μνήμης στην φυσική. Αυτό συμβαίνει διότι η φυσική μνήμη δεσμεύεται on demand από το ΛΣ όταν γίνεται η προσπέλαση. Έτσι, όταν πάμε να προσπελάσουμε εικονική μνήμη που δεν έχει απεικονιστεί ακόμα θα γίνει page fault και αφού διαπιστωθεί ότι η εικονική μνήμη βρίσκεται στον χάρτη μνήμης τότε το ΛΣ επιλέγει ένα frame για να γίνει απεικόνιση.

4. Γεμίζουμε με μηδενικά τον buffer (επομένως τον προσπελάζουμε) και επαναλαμβάνουμε το προηγούμενο βήμα:

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

Physical address of private heap is: 7727460352
```

Τώρα έχει γίνει η απεικόνιση στη φυσική μνήμη όπως περιγράψαμε παραπάνω.

5. Απεικονίζουμε το αρχείο file.txt στο χώρο διευθύνσεων της διεργασίας με την χρήση της mmap()(memory-mapped file) και τυπώνουμε το περιεχόμενο του:

```
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 [415459]:
55fd33d79000-55fd33d7a000 r--p 00000000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7a000-55fd33d7c000 r-xp 00001000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7c000-55fd33d7d000 r--p 00003000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7d000-55fd33d7e000 r--p 00003000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7e000-55fd33d7f000 rw-p 00004000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd35283000-55fd352a4000 rw-p 00000000 00:00 0                          [heap]
7f472bbb7000-7f472bbd9000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bbd9000-7f472bd32000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd32000-7f472bd81000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd81000-7f472bd85000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd85000-7f472bd87000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd87000-7f472bd8d000 rw-p 00000000 00:00 0
7f472bd91000-7f472bd92000 r--s 00000000 00:26 8540002                    /home/oslab/oslab35/os-lab-exer4/mmap/file.txt
7f472bd92000-7f472bd93000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bd93000-7f472bdb3000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdb3000-7f472bdbb000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbb000-7f472bdbc000 rw-p 00000000 00:00 0
7f472bdbc000-7f472bdbd000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbd000-7f472bdbe000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbe000-7f472bdbf000 rw-p 00000000 00:00 0
7fff6d440000-7fff6d461000 rw-p 00000000 00:00 0                          [stack]
7fff6d478000-7fff6d47c000 r--p 00000000 00:00 0                          [vvar]
7fff6d47c000-7fff6d47e000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------------

7f472bd91000-7f472bd92000 r--s 00000000 00:26 8540002                    /home/oslab/oslab35/os-lab-exer4/mmap/file.txt
Physical address  of file shared heap is : 8406568960
```

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

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


Virtual Memory Map of process [415459]:
55fd33d79000-55fd33d7a000 r--p 00000000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7a000-55fd33d7c000 r-xp 00001000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7c000-55fd33d7d000 r--p 00003000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7d000-55fd33d7e000 r--p 00003000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7e000-55fd33d7f000 rw-p 00004000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd35283000-55fd352a4000 rw-p 00000000 00:00 0                          [heap]
7f472bbb7000-7f472bbd9000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bbd9000-7f472bd32000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd32000-7f472bd81000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd81000-7f472bd85000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd85000-7f472bd87000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd87000-7f472bd8d000 rw-p 00000000 00:00 0
7f472bd90000-7f472bd91000 rw-s 00000000 00:01 17129                      /dev/zero (deleted)
7f472bd91000-7f472bd92000 r--s 00000000 00:26 8540002                    /home/oslab/oslab35/os-lab-exer4/mmap/file.txt
7f472bd92000-7f472bd93000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bd93000-7f472bdb3000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdb3000-7f472bdbb000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbb000-7f472bdbc000 rw-p 00000000 00:00 0
7f472bdbc000-7f472bdbd000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbd000-7f472bdbe000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbe000-7f472bdbf000 rw-p 00000000 00:00 0
7fff6d440000-7fff6d461000 rw-p 00000000 00:00 0                          [stack]
7fff6d478000-7fff6d47c000 r--p 00000000 00:00 0                          [vvar]
7fff6d47c000-7fff6d47e000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------------

7f472bd90000-7f472bd91000 rw-s 00000000 00:01 17129                      /dev/zero (deleted)
```

Καλείται η fork()και δημιουργείται μία νέα διεργασία.

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

```
Step 7: Print parent's and child's map.
Parent has the following vm map :
Virtual Memory Map of process [415459]:
55fd33d79000-55fd33d7a000 r--p 00000000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7a000-55fd33d7c000 r-xp 00001000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7c000-55fd33d7d000 r--p 00003000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7d000-55fd33d7e000 r--p 00003000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7e000-55fd33d7f000 rw-p 00004000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd35283000-55fd352a4000 rw-p 00000000 00:00 0                          [heap]
7f472bbb7000-7f472bbd9000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bbd9000-7f472bd32000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd32000-7f472bd81000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd81000-7f472bd85000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd85000-7f472bd87000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd87000-7f472bd8d000 rw-p 00000000 00:00 0
7f472bd90000-7f472bd91000 rw-s 00000000 00:01 17129                      /dev/zero (deleted)
7f472bd91000-7f472bd92000 r--s 00000000 00:26 8540002                    /home/oslab/oslab35/os-lab-exer4/mmap/file.txt
7f472bd92000-7f472bd93000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bd93000-7f472bdb3000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdb3000-7f472bdbb000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbb000-7f472bdbc000 rw-p 00000000 00:00 0
7f472bdbc000-7f472bdbd000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbd000-7f472bdbe000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbe000-7f472bdbf000 rw-p 00000000 00:00 0
7fff6d440000-7fff6d461000 rw-p 00000000 00:00 0                          [stack]
7fff6d478000-7fff6d47c000 r--p 00000000 00:00 0                          [vvar]
7fff6d47c000-7fff6d47e000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------------


Child has the following vm map :
Virtual Memory Map of process [415487]:
55fd33d79000-55fd33d7a000 r--p 00000000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7a000-55fd33d7c000 r-xp 00001000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7c000-55fd33d7d000 r--p 00003000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7d000-55fd33d7e000 r--p 00003000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd33d7e000-55fd33d7f000 rw-p 00004000 00:26 8540007                    /home/oslab/oslab35/os-lab-exer4/mmap/mmap
55fd35283000-55fd352a4000 rw-p 00000000 00:00 0                          [heap]
7f472bbb7000-7f472bbd9000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bbd9000-7f472bd32000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd32000-7f472bd81000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd81000-7f472bd85000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd85000-7f472bd87000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f472bd87000-7f472bd8d000 rw-p 00000000 00:00 0
7f472bd90000-7f472bd91000 rw-s 00000000 00:01 17129                      /dev/zero (deleted)
7f472bd91000-7f472bd92000 r--s 00000000 00:26 8540002                    /home/oslab/oslab35/os-lab-exer4/mmap/file.txt
7f472bd92000-7f472bd93000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bd93000-7f472bdb3000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdb3000-7f472bdbb000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbb000-7f472bdbc000 rw-p 00000000 00:00 0
7f472bdbc000-7f472bdbd000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbd000-7f472bdbe000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f472bdbe000-7f472bdbf000 rw-p 00000000 00:00 0
7fff6d440000-7fff6d461000 rw-p 00000000 00:00 0                          [stack]
7fff6d478000-7fff6d47c000 r--p 00000000 00:00 0                          [vvar]
7fff6d47c000-7fff6d47e000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------------
```

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

8. Βρίσκουμε και τυπώνουμε τη φυσική διεύθυνση στην κύρια μνήμη του private buffer για τις διεργασίες πατέρα και παιδί:

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

7f472bdbb000-7f472bdbc000 rw-p 00000000 00:00 0
Physical address of private heap buffer on father: 7727460352
Physical address of private heap buffer on child: 7727460352
```

Απεικονίζονται στην ίδια φυσική μνήμη όπως αναφέραμε και παραπάνω.

9. Γράφουμε στον private buffer από τη διεργασία παιδί και επαναλαμβάνουμε το προηγούμενο βήμα:

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

7f472bdbb000-7f472bdbc000 rw-p 00000000 00:00 0
Physical address of private heap buffer on father: 7726936064
7f472bdbb000-7f472bdbc000 rw-p 00000000 00:00 0
Physical address of private heap buffer on child: 7727460352
```

Αυτή η τεχνική ονομάζεται Copy-on-Write (COW).
Αυτό που συμβαίνει είναι: Αντιγράφεται η εικονική μνήμη και ο πίνακας σελίδων (με την αφαίρεση των write δικαιωμάτων αν πρόκειται για private page) και μέχρι κάποια από τις διεργασίες προσπαθήσει να γράψει σε κάποια private σελίδα η απεικόνιση της εικονικής διεύθυνσης στην φυσική διεύθυνση είναι η ίδια. Όταν κάποια διεργασία προσπαθήσει να γράψει σε κάποια private σελίδα τότε το ΛΣ βρίσκει ένα νέο πλαίσιο για την απεικόνιση, αντιγράφεται το περιεχόμενο του page στην νέα φυσική διεύθυνση και ενημερώνεται ο πίνακας σελίδων. Επομένως, παρατηρούμε εδώ ότι όταν γράφουμε στον private buffer από τη διεργασία παιδί τότε οι φυσικές διευθύνσεις είναι διαφορετικές.

10. Γράφουμε στον shared buffer από τη διεργασία παιδί και τυπώνουμε τη φυσική 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?

7f472bd90000-7f472bd91000 rw-s 00000000 00:01 17129                    /dev/zero (deleted)
Physical address of shared heap buffer on father: 7726952448
7f472bd90000-7f472bd91000 rw-s 00000000 00:01 17129                    /dev/zero (deleted)
Physical address of shared heap buffer on child: 7726952448
```

Παρατηρούμε εδώ, σε σύγκριση με τον private buffer, ότι ο shared buffer απεικονίζεται στην ίδια φυσική διεύθυνση και για τις δύο διεργασίες. Αυτό γίνεται διότι η σελίδα τώρα είναι διαμοιραζόμενη (shared) μεταξύ των διεργασιών (MAP_SHARED flag στην mmap()) και επομένως αντιστοιχίζεται στην ίδια φυσική διεύθυνση. Αυτό δίνει την δυνατότητα για διαδιεργασιακή επικοινωνία αφού "μοιράζονται" την ίδια μνήμη (βλ. 1.2).

11. Απαγορεύουμε τις εγγραφές στον shared buffer για την διεργασία παιδί (αφαιρούμε το write permission) με χρήση της mprotect():

```
VM map of parent
Virtual Memory Map of process [415459]:
7f472bd90000-7f472bd91000 rw-s 00000000 00:01 17129                    /dev/zero (deleted)
```

```
VM map of child:

Virtual Memory Map of process [415487]:
7f472bd90000-7f472bd91000 r--s 00000000 00:01 17129                    /dev/zero (deleted)
```

Παρατηρούμε έχει αφαιρεθεί το write δικαίωμα από την διεργασία παιδί (αν πάει να γράψει θα έχουμε segfault).

12. Τέλος, αποδεσμεύουμε όλους τους buffers στις δύο διεργασίες με χρήση της munmap().

1.2 (Παράλληλος υπολογισμός συνόλου Mandelbrot με διεργασίες αντί για νήματα)

Ζητείται η τροποποίηση του προγράμματος υπολογισμού του Mandelbrot set, της άσκησης 3, ώστε αντί να χρησιμοποιεί threads (pthreads) για την παραλληλοποίηση του υπολογισμού του set, να χρησιμοποιεί διεργασίες (processes). Η κατανομή του υπολογιστικού φόρτου γίνεται ανά σειρά, ακριβώς όπως στην περίπτωση των threads: Για n διεργασίες, η διεργασία i (με i = 0, 1, 2, . . . , n − 1) αναλαμβάνει τις σειρές i, i + n, i + 2 × n, i + 3 × n, . . . Ο αριθμός των διεργασιών, NPROCS, όπως και ο αριθμός των threads στην προηγούμενη άσκηση, δίνεται ως όρισμα στη γραμμή εντολών.

1.2.1 Semaphores πάνω από διαμοιραζόμενη μνήμη

Ακολοθεί ο κώδικας του μέρους 1.2.1 :

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

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

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

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000
#define _GNU_SOURCE
/***************************
 * Compile-time parameters *
 ***************************/

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;
}

sem_t *S;

/*
 * 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;

void sig_handler(int signum)
{
    printf(" Color will be restored after signal : %d\n", signum);
    reset_xterm_color(1);
    printf("Color restored\n");
    printf("Exiting...\n");
    exit(EXIT_SUCCESS);
}
/*
 * This function computes a line of output
 * as an array of x_char color values.
 */
void compute_mandel_line(int line, int color_val[])
{
    /*
     * x and y traverse the complex plane.
     */
    double x, y;

    int n;
    int val;

    /* Find out the y value corresponding to this line */
    y = ymax - ystep * line;

    /* and iterate for all points on this line */
    for (x = xmin, n = 0; n < x_chars; x += xstep, n++)
    {

        /* Compute the point's color value */
        val = mandel_iterations_at_point(x, y, MANDEL_MAX_ITERATION);
        if (val > 255)
            val = 255;

        /* And store it in the color_val[] array */
        val = xterm_color(val);
        color_val[n] = val;
    }
}
```

```
/*
 * This function outputs an array of x_char color values
 * to a 256-color xterm.
 */
void output_mandel_line(int fd, int color_val[])
{
    int i;

    char point = '@';
    char newline = '\n';

    for (i = 0; i < x_chars; i++)
    {
        /* Set the current color, then output the point */
        set_xterm_color(fd, color_val[i]);
        if (write(fd, &point, 1) != 1)
        {
            perror("compute_and_output_mandel_line: write point");
            exit(1);
        }
    }

    /* Now that the line is done, output a newline character */
    if (write(fd, &newline, 1) != 1)
    {
        perror("compute_and_output_mandel_line: write newline");
        exit(1);
    }
}

void children_draw(int line, int numprocs)
{
    int color_val[x_chars];
    for (int index = line; index < y_chars; index += numprocs)
    {
        compute_mandel_line(index, color_val);
        if (sem_wait(&S[line]) < 0)
        {
            perror("sem_wait");
        }
        output_mandel_line(1, color_val);
        if (sem_post(&S[(index + 1) % numprocs]) < 0)
        {
```

```
            perror("sem_post");
        }
    }
    return;
}
/*
 * 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(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);
    }
}

int main(int argc, char *argv[])
{
    int numprocs;
    xstep = (xmax - xmin) / x_chars;
    ystep = (ymax - ymin) / y_chars;

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

    struct sigaction act;
    act.sa_handler = sig_handler;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    if (sigaction(SIGINT, &act, NULL) == -1)
    {
        perror("sigaction");
        exit(EXIT_FAILURE);
    }
```

```
    S = create_shared_memory_area(numprocs * sizeof(sem_t));

    // Initialiaze the semaphores
    sem_init(&S[0], 1, 1);
    for (int j = 1; j < numprocs; ++j)
    {
        if (sem_init(&S[j], 1, 0))
        {
            perror("semaphore init");
            exit(EXIT_FAILURE);
        }
    }

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

    // Let's make some children to draw
    pid_t pid;
    int status;
    for (int i = 0; i < numprocs; i++)
    {
        pid = fork();
        if (pid < 0)
        {
            perror("error with creation of child");
            exit(EXIT_FAILURE);
        }
        if (pid == 0)
        {
            children_draw(i, numprocs);
            exit(EXIT_FAILURE);
        }
    }
    for (int i = 0; i < numprocs; i++)
    {
        pid = wait(&status);
    }

    for (int i = 0; i < numprocs; i++)
    {
        sem_destroy(&S[i]);
    }
```

```
    destroy_shared_memory_area(S, numprocs * sizeof(sem_t));

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

Ερωτήσεις:

1. Ποια από τις δύο παραλληλοποιημένες υλοποιήσεις (threads vs processes) περιμένετε να έχει καλύτερη επίδοση και γιατί; Πώς επηρεάζει την επίδοση της υλοποίησης με διεργασίες το γεγονός ότι τα semaphores βρίσκονται σε διαμοιραζόμενη μνήμη μεταξύ διεργασιών;

   Η υλοποίηση των threads θα έχει καλύτερη επίδοση. Αυτό συμβαίνει γιατί τα threads δε δημιουργούν δικό τους PCB και δεν χρειάζονται να αντιγράψουνε όλες τις πληροφορίες της γονικής διεργασίας(αρχιτεκτονική κατάσταση, πίνακας ανοικτών αρχειών, δικαιώματα εγγραφών, PC κ.λ.π.). Παράλληλα, η εναλλαγή διεργασιών είναι πιο αργή από την εναλλαγή των threads, αφού απαιτείται context switch στην πρώτη περίπτωση για την εναλλαγή των διεργασιών στην CPU και της αλλαγής κατάστασης της προηγούμενης διεργασίας από εκτελούμενη σε έτοιμη και τοποθέτηση στην ουρά έτοιμων. Επιπλέον, τα threads έχουν μοιράζονται την κοινή μνήμη της διεργασίας στην οποία ανήκουν, ενώ στην περίπτωση των processes πρέπει να χρησιμοποιήσουμε την συνάρτηση create_shared_memory_area(), η οποία κάνει mmap().

1.2.2 Υλοποιήση χωρίς semaphores

Ακολουθεί ο κώδικας του μέρους 1.2.2 :

```
/*
 * 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 <errno.h>
#include <signal.h>
```

```c
#include <sys/mman.h>
#include <sys/wait.h>

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000


int **buff; //2D Mandelbrot set output

/*
 * 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);
        }
}

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 */
        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);
        }
}

/*
 * 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);
}

void children_compute(int line, int procnt)
{
        int line_num;
        //every process writes to the buffer
        for (line_num=line ; line_num<y_chars; line_num+=procnt) {
                compute_mandel_line(line_num, buff[line_num]);
        }
        return;
}


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

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

        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);
        }

         /*
         * 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);
        }
```

```
        buff=create_shared_memory_area(y_chars * sizeof(int)); //create
the initial 1d array
        for (i=0; i<y_chars; i++) {
          buff[i]=create_shared_memory_area(x_chars * sizeof(char));
          //go to every potition and make x_chars of memory
        }

        //create processes and call 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) {
                        children_compute(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;
}
```

Ερωτήσεις:

1. Με ποιο τρόπο και σε ποιο σημείο επιτυγχάνεται ο συγχρονισμός σε αυτή την υλοποίηση; Πώς θα επηρεαζόταν το σχήμα συγχρονισμού αν ο buffer είχε διαστάσεις NPROCS x x_chars;

Σε αυτό το σημείο, ο συγχρονισμός επιτυγχάνεται από το γεγονός πως η κάθε διεργασία γράφει στον buffer μόνο εκεί που της αναλογεί και έτσι δημιουργείται παράλληλα το output, χωρίς η μια διεργασία να επηρεάζει την άλλη. Πρακτικά, δεν υπάρχει κρίσιμο τμήμα, διότι την ευθύνη για το output την έχει η αρχική διεργασία. Εάν είχαμε μικρότερο πίνακα (NPROCS x x_chars αντί y_chars x x_chars) τότε δεν θα μπορούσαμε να τυπώναμε με την μία όλο το output και θα έπρεπε να γίνει σε «δόσεις». Μια λύση θα ήταν η χρήση σημάτων, δηλαδή όταν μια διεργασία υπολογίσει την γραμμή της θα κάνει raise(SIGSTOP), έπειτα η αρχική διεργασία θα περιμένει μέχρι όλες οι διεργασίες παιδιά της να την ενημερώσουν πως έχουν σταματήσει και άρα έχουν υπολογίσει την γραμμή που τους αντιστοιχεί, θα τυπώσει το αντίστοιχο buffer και θα τις ενεργοποιήσει μία μία, επαναλαμβάνοντας την παραπάνω διαδικασία μέχρι να υπολογιστεί η έξοδος. Έτσι, επιτυγχάνεται διαδιεργαστική επικοινωνία μεταξύ γονεϊκής διεργασίας και διεργασιών-παιδιών και έτσι επιτυγχάνεται ο συγχρονισμός. Στην πραγματικότητα βέβαια, αυτή η υλοποίηση θα είναι πολύ λιγότερο αποδοτική από τις προηγούμενες.

```
time ./mandel-fork-sem 10
```

```
real    0m0.249s
user    0m1.196s
sys     0m0.081s
```

```
time ./mandel-nosem 10
```

```
real    0m0.291s
user    0m1.623s
sys     0m0.057s
```