

# ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

## ΑΝΑΦΟΡΑ

### Άσκηση 4

Αρετή Μέη 03120062  
Ίρις-Ελευθερία Παλιατσού 03120639

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

Ο πηγαίος κώδικας της άσκησης:

```
/*  
 * mmap.c  
 *  
 * Examining the virtual memory of processes.  
 *  
 * Operating Systems course, CSLab, ECE, NTUA  
 *  
 */  
  
#include <stdlib.h>  
#include <string.h>  
#include <stdio.h>  
  
#include <stdio.h>  
#include <sys/mman.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <errno.h>  
#include <stdint.h>  
#include <signal.h>  
#include <sys/wait.h>  
#include <inttypes.h>  
  
#include "help.h"  
  
#define RED      "\033[31m"  
#define RESET    "\033[0m"  
  
char *heap_private_buf;  
char *heap_shared_buf;  
  
char *file_shared_buf;  
  
uint64_t buffer_size;  
/*  
 * Child process' entry point.  
 */  
void child(void)  
{  
    /*  
     * Step 7 - Child  
     */  
    if (0 != raise(SIGSTOP))  
        die("raise(SIGSTOP)");  
    /*  
     * TODO: Write your code here to complete child's part of Step 7.  
     */
```

```

printf("Map of child process: \n"); //we expect that the child has inherited a copy of the VM of the father.
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 buffer from child=%ld \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.
 */

int i;
for(i=0; i<(int)buffer_size; i++) {
    heap_private_buf[i]=i;
}
printf("Physical address of private buffer from 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(i=0; i<(int)buffer_size; i++) {
    heap_shared_buf[i]=i;
}
printf("Physical address of shared buffer from 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.
 */

if (mprotect(heap_shared_buf, buffer_size, PROT_READ) == -1) {
    die("mprotect");
}

show_va_info((uint64_t)heap_shared_buf);

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

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

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

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

    /*
     * TODO: Write your code here to complete parent's part of Step 7.
     */
    printf("Map of parent process: \n");

```

```

show_maps();

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

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

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

printf("Physical address of private buffer from 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 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();

printf("Physical address of shared buffer from parent=%ld \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();

/*
 * TODO: Write your code here to complete parent's part of Step 10.
 */
printf("Physical address of private buffer from parent=%ld \n", get_physical_address((uint64_t)heap_shared_buf));

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

/*
 * Step 11: Disable writing on the shared buffer for the child
 * (hint: mprotect(2)).
 * Step 11 - Parent
 */
printf(RED "\nStep 11: Disable writing on the shared buffer for the "
       "child. Verify through the maps for the parent and the "
       "child.\n" RESET);
press_enter();

/*
 * TODO: Write your code here to complete parent's part of Step 11.
 */

show_va_info((uint64_t)heap_shared_buf);
if (-1 == kill(child_pid, SIGCONT))
    die("kill");
if (-1 == waitpid(child_pid, &status, 0))
    die("waitpid");

/*
 * Step 12: Free all buffers for parent and child.
 * Step 12 - Parent
*/

```

```

    * [TODO: Write your code here to complete parent's part of Step 12.
 */
munmap(heap_private_buf, buffer_size);
munmap(heap_shared_buf, buffer_size);
munmap(file_shared_buf, buffer_size);

}

int main(void)
{
    pid_t mypid, p;
    int fd = -1;

    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);

    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 of buffer = %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 address of buffer = %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);
//why shared, where is fd used, is this default
if(file_shared_buf == MAP_FAILED)
    die("mmap");

for(i=0; i<(int)buffer_size; i++) {
    if(file_shared_buf[i]!=EOF) //while we are not at the end of the file.txt
        putchar(file_shared_buf[i]);
    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_ANONYMOUS|MAP_SHARED, -1, 0);
if (heap_shared_buf == MAP_FAILED) {
    die("mmap");
}

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

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;

```

H έξοδος εκτέλεσης του προγράμματος:

1. Τυπώστε το χάρτη της εικονικής μνήμης της τρέχουσας διεργασίας

```
Step 1: Print the virtual address space map of this process [12676].
```

```
Virtual Memory Map of process [12676]:  
00400000-00403000 r-xp 00000000 fe:10 7604664 /store/  
homes/oslab/oslab25/exercise4/mmap  
00602000-00603000 rw-p 00002000 fe:10 7604664 /store/  
homes/oslab/oslab25/exercise4/mmap  
01584000-015a5000 rw-p 00000000 00:00 0 [heap]  
7fac64ae000-7fac6484f000 r-xp 00000000 fe:01 1045760 /lib/x8  
6_64-linux-gnu/libc-2.19.so  
7fac6484f000-7fac64a4f000 ---p 001a1000 fe:01 1045760 /lib/x8  
6_64-linux-gnu/libc-2.19.so  
7fac64a4f000-7fac64a53000 r--p 001a1000 fe:01 1045760 /lib/x8  
6_64-linux-gnu/libc-2.19.so  
7fac64a53000-7fac64a55000 rw-p 001a5000 fe:01 1045760 /lib/x8  
6_64-linux-gnu/libc-2.19.so  
7fac64a55000-7fac64a59000 rw-p 00000000 00:00 0  
7fac64a59000-7fac64a79000 r-xp 00000000 fe:01 1044705 /lib/x8  
6_64-linux-gnu/ld-2.19.so  
7fac64c6000-7fac64c6f000 rw-p 00000000 00:00 0  
7fac64c74000-7fac64c79000 rw-p 00000000 00:00 0  
7fac64c79000-7fac64c7a000 r--p 00020000 fe:01 1044705 /lib/x8  
6_64-linux-gnu/ld-2.19.so  
7fac64c7a000-7fac64c7b000 rw-p 00021000 fe:01 1044705 /lib/x8  
6_64-linux-gnu/ld-2.19.so  
7fac64c7b000-7fac64c7c000 rw-p 00000000 00:00 0  
7ffc927e2000-7ffc92803000 rw-p 00000000 00:00 0 [stack]  
7ffc929b7000-7ffc929b9000 r-xp 00000000 00:00 0 [vdsos]  
7ffc929b9000-7ffc929bb000 r--p 00000000 00:00 0 [vvar]  
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]  
11]
```

2. Με την κλήση συστήματος mmap() δεσμεύστε buffer (προσωρινή μνήμη) μεγέθους μίας σελίδας (page) και τυπώστε ξανά το χάρτη. Εντοπίστε στο χάρτη μνήμης τον χώρο εικονικών διευθύνσεων που δεσμεύσατε.

```
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 [12676]:  
00400000-00403000 r-xp 00000000 fe:10 7604664 /store/homes/oslab/oslab25/exercise4/mmap  
00602000-00603000 rw-p 00002000 fe:10 7604664 /store/homes/oslab/oslab25/exercise4/mmap  
01584000-015a5000 rw-p 00000000 00:00 0 [heap]  
7fac64ae000-7fac6484f000 r-xp 00000000 fe:01 1045760 /lib/x86_64-linux-gnu/libc-2.19.so  
7fac6484f000-7fac64a4f000 ---p 001a1000 fe:01 1045760 /lib/x86_64-linux-gnu/libc-2.19.so  
7fac64a4f000-7fac64a53000 r--p 001a1000 fe:01 1045760 /lib/x86_64-linux-gnu/libc-2.19.so  
7fac64a53000-7fac64a55000 rw-p 001a5000 fe:01 1045760 /lib/x86_64-linux-gnu/libc-2.19.so  
7fac64a55000-7fac64a59000 rw-p 00000000 00:00 0  
7fac64a59000-7fac64a79000 r-xp 00000000 fe:01 1044705 /lib/x86_64-linux-gnu/ld-2.19.so  
7fac64c6000-7fac64c6f000 rw-p 00000000 00:00 0  
7fac64c73000-7fac64c79000 rw-p 00000000 00:00 0  
7fac64c79000-7fac64c7a000 r--p 00020000 fe:01 1044705 /lib/x86_64-linux-gnu/ld-2.19.so  
7fac64c7a000-7fac64c7b000 rw-p 00021000 fe:01 1044705 /lib/x86_64-linux-gnu/ld-2.19.so  
7fac64c7b000-7fac64c7c000 rw-p 00000000 00:00 0  
7ffc927e2000-7ffc92803000 rw-p 00000000 00:00 0 [stack]  
7ffc929b7000-7ffc929b9000 r-xp 00000000 00:00 0 [vdsos]  
7ffc929b9000-7ffc929bb000 r--p 00000000 00:00 0 [vvar]  
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]  
----  
7fac64c73000-7fac64c79000 rw-p 00000000 00:00 0
```

Ο χώρος εικονικών διευθύνσεων που δεσμεύσαμε φαίνεται στην τελευταία γραμμή της εξόδου.

3. Προσπαθήστε να βρείτε και να τυπώσετε τη φυσική διεύθυνση μνήμης στην οποία απεικονίζεται η εικονική διεύθυνση του buffer (τη διεύθυνση όπου βρίσκεται αποθηκευμένος στη φυσική κύρια μνήμη). Τι παρατηρείτε και γιατί;

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

```
VA[0x7f13e8fb000] is not mapped; no physical memory allocated.  
Physical address of buffer = 0
```

Βλέπουμε ότι καλώντας την `get_physical_address()` για τον Buffer, μας λέει πως ο buffer δεν έχει γίνει mapped. Αυτό συμβαίνει διότι ενώ όπως βλέπουμε στον χάρτη μνήμης έχει δεσμευτεί η εικονική μνήμη, η φυσική μνήμη δεσμεύεται on demand από το λειτουργικό σύστημα όταν πάει να προσπελαστεί(On demand paging). Επομένως, δεν έχουν δημιουργηθεί ακόμη τα πλαίσια στην φυσική μνήμη που αντιστοιχούν στα pages του buffer, αφού αυτός δεν έχει ακόμη προσπελαστεί.

4. Γεμίστε με μηδενικά τον buffer και επαναλάβετε το Βήμα 3. Ποια αλλαγή παρατηρείτε;

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

```
Physical address of buffer = 7254425600
```

Τώρα που ο buffer έχει γεμίσει με μηδενικά, δηλαδή έχει γίνει η προσπέλασή του, ο buffer έχει φυσική διεύθυνση.

5. Χρησιμοποιείστε την `mmap()` για να απεικονίσετε (memory map) το αρχείο `file.txt` στον χώρο διευθύνσεων της διεργασίας σας και να τυπώσετε το περιεχόμενό του. Εντοπίστε 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 [609477]:  
00400000-00403000 r-xp 00000000 00:26 7604664  
00602000-00603000 rw-p 00002000 00:26 7604664  
01a32000-01a53000 rw-p 00000000 00:00 0  
7f13e8dba000-7f13e8ddc000 r--p 00000000 fe:01 144567  
7f13e8ddc000-7f13e8f35000 r--p 00022000 fe:01 144567  
7f13e8f35000-7f13e8f84000 r--p 0017b000 fe:01 144567  
7f13e8f84000-7f13e8f88000 r--p 001c9000 fe:01 144567  
7f13e8f88000-7f13e8f8a000 rw-p 001cd000 fe:01 144567  
7f13e8f8a000-7f13e8f90000 rw-p 00000000 00:00 0  
7f13e8f95000-7f13e8f96000 r--s 00000000 00:26 7620145  
7f13e8f96000-7f13e8f97000 r--p 00000000 fe:01 144563  
7f13e8f97000-7f13e8fb7000 r--p 00001000 fe:01 144563  
7f13e8fb7000-7f13e8fbff000 r--p 00021000 fe:01 144563  
7f13e8fbff000-7f13e8fc0000 rw-p 00000000 00:00 0  
7f13e8fc0000-7f13e8fc1000 r--p 00029000 fe:01 144563  
7f13e8fc1000-7f13e8fc2000 rw-p 0002a000 fe:01 144563  
7f13e8fc2000-7f13e8fc3000 rw-p 00000000 00:00 0  
7ffc7ed0000-7ffc7ed2e000 rw-p 00000000 00:00 0  
7ffc7edc6000-7ffc7edca000 r--p 00000000 00:00 0  
7ffc7edca000-7ffc7edcc000 r--p 00000000 00:00 0  
-----  
7f13e8f95000-7f13e8f96000 r--s 00000000 00:26 7620145  
-----  
/home/oslab/oslab25/exercise4/file.txt
```

Η απεικόνιση του αρχείου `file.txt` στον χώρο διευθύνσεων αναγράφεται στην τελευταία γραμμή, είναι δηλαδή:

```
7f13e8f95000-7f13e8f96000 r--s 00000000 00:26 7620145  
/home/oslab/oslab25/exercise4/file.txt
```

Βλέπουμε μάλιστα και το όνομα του αρχείου στα δεξιά.

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

Η νέα απεικόνιση στον 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.  
7f13e8f94000-7f13e8f95000 rw-s 00000000 00:01 12725  
/dev/zero (deleted)
```

Βλέπουμε το flag “s”, που σημαίνει shared.

7. Τυπώστε τον χάρτη της εικονικής μνήμης της διεργασίας πατέρα και της διεργασίας παιδιού.  
Τι παρατηρείτε?

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

Map of parent process:

Virtual Memory Map of process [609477]:
00400000-00403000 r-xp 00000000 00:26 7604664
00602000-00603000 rw-p 00002000 00:26 7604664
01a32000-01a53000 rw-p 00000000 00:00 0
7f13e8dba000-7f13e8ddc000 r--p 00000000 fe:01 144567
7f13e8ddc000-7f13e8f35000 r-xp 00022000 fe:01 144567
7f13e8f35000-7f13e8f84000 r--p 0017b000 fe:01 144567
7f13e8f84000-7f13e8f88000 r--p 001c9000 fe:01 144567
7f13e8f88000-7f13e8f8a000 rw-p 001cd000 fe:01 144567
7f13e8f8a000-7f13e8f90000 rw-p 00000000 00:00 0
7f13e8f94000-7f13e8f95000 rw-s 00000000 00:01 12725
7f13e8f95000-7f13e8f96000 r--s 00000000 00:26 7620145
7f13e8f96000-7f13e8f97000 r--p 00000000 fe:01 144563
7f13e8f97000-7f13e8fb7000 r-xp 00001000 fe:01 144563
7f13e8fb7000-7f13e8fbf000 r--p 00021000 fe:01 144563
7f13e8fbf000-7f13e8fc0000 rw-p 00000000 00:00 0
7f13e8fc0000-7f13e8fc1000 r--p 00029000 fe:01 144563
7f13e8fc1000-7f13e8fc2000 rw-p 0002a000 fe:01 144563
7f13e8fc2000-7f13e8fc3000 rw-p 00000000 00:00 0
7ffc7ed0d000-7ffc7ed2e000 rw-p 00000000 00:00 0
7ffc7edc6000-7ffc7edca000 r--p 00000000 00:00 0
7ffc7edca000-7ffc7edcc000 r-xp 00000000 00:00 0
-----

```

```
Map of child process:

Virtual Memory Map of process [609478]:
00400000-00403000 r-xp 00000000 00:26 7604664
00602000-00603000 rw-p 00002000 00:26 7604664
01a32000-01a53000 rw-p 00000000 00:00 0
7f13e8dba000-7f13e8ddc000 r--p 00000000 fe:01 144567
7f13e8ddc000-7f13e8f35000 r-xp 00022000 fe:01 144567
7f13e8f35000-7f13e8f84000 r--p 0017b000 fe:01 144567
7f13e8f84000-7f13e8f88000 r--p 001c9000 fe:01 144567
7f13e8f88000-7f13e8f8a000 rw-p 001cd000 fe:01 144567
7f13e8f8a000-7f13e8f90000 rw-p 00000000 00:00 0
7f13e8f94000-7f13e8f95000 rw-s 00000000 00:01 12725
7f13e8f95000-7f13e8f96000 r--s 00000000 00:26 7620145
7f13e8f96000-7f13e8f97000 r--p 00000000 fe:01 144563
7f13e8f97000-7f13e8fb7000 r-xp 00001000 fe:01 144563
7f13e8fb7000-7f13e8fbf000 r--p 00021000 fe:01 144563
7f13e8fbf000-7f13e8fc0000 rw-p 00000000 00:00 0
7f13e8fc0000-7f13e8fc1000 r--p 00029000 fe:01 144563
7f13e8fc1000-7f13e8fc2000 rw-p 0002a000 fe:01 144563
7f13e8fc2000-7f13e8fc3000 rw-p 00000000 00:00 0
7ffc7ed0d000-7ffc7ed2e000 rw-p 00000000 00:00 0
7ffc7edc6000-7ffc7edca000 r--p 00000000 00:00 0
7ffc7edca000-7ffc7edcc000 r-xp 00000000 00:00 0
-----

```

Η διεργασία παιδί κληρονομεί ένα αντίγραφο του πίνακα σελίδων και της εικονικής μνήμης από την διεργασία πατέρα, ωστόσο σε όλα τα private pages αφαιρούνται τα δικαιώματα ανάγνωσης από τις διεργασίες (COW).

8. Βρείτε και τυπώστε τη φυσική διεύθυνση στη κύρια μνήμη του private buffer (Βήμα 3) για τις διεργασίες πατέρα και παιδί. Τι συμβαίνει αμέσως μετά το fork?

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

Physical address of private buffer from parent=7254425600
Physical address of private buffer from child=7254425600

```

Παρατηρούμε πως αφού καμία από τις δύο δεν έχει γράψει στον buffer, απεικονίζονται στην ίδια διεύθυνση φυσικής μνήμης.

9. Γράψτε στον private buffer από τη διεργασία παιδί και επαναλάβετε το Βήμα 8. Τι αλλάζει και γιατί;

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

```
Physical address of shared buffer from parent=7254425600  
Physical address of private buffer from child=4504301568
```

Αφού έχουμε εγγραφή από τη διεργασία παιδί, οι νέες διευθύνσεις της φυσικής μνήμης δεν θα είναι οι ίδιες. Το Λ.Σ. θα δημιουργήσει ένα νέο πλαίσιο για τη διεργασία παιδί, θα αντιστοιχίσει εκεί τη νέα τροποποιημένη σελίδα στην οποία έγραψε η διεργασία παιδί, και θα ενημερώσει το page table(COW).

10. Γράψτε στον shared buffer (Βήμα 6) από τη διεργασία παιδί και τυπώστε τη φυσική του διεύθυνση για τις διεργασίες πατέρα και παιδί. Τι παρατηρείτε σε σύγκριση με τον private buffer?

```
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?  
Physical address of private buffer from parent=6974312448  
Physical address of shared buffer from child=6974312448
```

Εδώ ο buffer μας είναι shared, επομένως υπάρχει μια διεύθυνση φυσικής μνήμης για τη διεργασία πατέρα και παιδί.

11. Απαγορεύστε τις εγγραφές στον shared buffer για τη διεργασία παιδί. Εντοπίστε και τυπώστε την απεικόνιση του shared buffer στο χάρτη μνήμης των δύο διεργασιών για να επιβεβαιώσετε την απαγόρευση.

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

```
7f13e8f94000-7f13e8f95000 rw-s 00000000 00:01 12725 /dev/zero (deleted)  
7f13e8f94000-7f13e8f95000 r--s 00000000 00:01 12725 /dev/zero (deleted)  
oslab25@os-node1:~/exercises4$
```

Η απαγόρευση επιβεβαιώνεται με το ρ αντί για rw στα δικαιώματα της διεργασίας παιδιού, που σημαίνει ότι όντως δεν έχει πλέον το δικαίωμα εγγραφής, αλλά μόνο ανάγνωσης.

## 1.2 Παράλληλος υπολογισμός Mandelbrot με διεργασίες αντί για νήματα

### 1.2.1 Semaphores πάνω από διαμοιραζόμενη μνήμη

Ο πηγαίος κώδικας της άσκησης:

```

/*
 * 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 <semaphore.h>
#include <errno.h>
#include <signal.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <inttypes.h>
#include "help.h"
#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

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

 * Compile-time parameters  

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

 * Output at the terminal is is x_chars wide by y_chars long  

 */
int y_chars = 50;
int x_chars = 90;

sem_t *sem;
int NPROCS=3;

/*  

 * The part of the complex plane to be drawn:  

 * upper left corner is (xmin, ymax), lower right corner is (xmax, ymin)  

 */
double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;

/*  

 * Every character in the final output is  

 * xstep x ystep units wide on the complex plane.  

 */

double xstep;
double ystep;

/*
 * This function computes a line of output
 * as an array of x_char color values.
 */
void compute_mandel_line(int line, int color_val[])
{
    /*
     * x and y traverse the complex plane.
     */
    double x, y;

    int n;
    int val;

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

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

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

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

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

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

    for (i = 0; i < x_chars; i++) {
        /* Set the current color, then output the point */
        set_xterm_color(fd, color_val[i]);
    }
}

```

```
        if (write(fd, &point, 1) != 1) {
            perror("compute_and_output_mandel_line: write point");
            exit(1);
        }

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

void compute_and_output_mandel_line(int line, int NPROCS) {
    int line_number, color_val[x_chars];

    for(line_number = line; line_number < y_chars; line_number += NPROCS) {
        compute_mandel_line(line_number, color_val);
        sem_wait(&sem[line]);
        output_mandel_line(l, color_val);
        sem_post(&sem[(line_number + 1) % NPROCS]);
    }
}

/*
 * 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;
```

```

        addr = mmap(NULL, pages*sysconf(_SC_PAGE_SIZE) , PROT_READ | PROT_WRITE, MAP_ANONYMOUS|MAP_SHARED, -1, 0);
        if (addr == MAP_FAILED) {
            die("mmap");
        }
        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 i,line;
    if(argc>1) {
        NPROCS=atoi(argv[1]);
    }
    pid_t pid;

    // Create shared memory area
    sem = create_shared_memory_area(NPROCS*sizeof(sem_t));

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

    for (i = 1; i < NPROCS; i++) {
        sem_init(&sem[i], 1, 0); //value of 1 indicates that the semaphore can be shared among processes. This
                                //is used to increment the value of a semaphore
                                //from child process
        perror("sem_init");
    }

    if(sem_init(&sem[0],1,1)<0) { //used to increment the value of a semaphore
        perror("sem_post");
        exit(1);
    }

    int status;

    // Fork the NPROCS processes

    for(line=0; line<NPROCS; line++) {
        pid = fork();
        if (pid == -1) {
            perror("fork");
            exit(1);
        }
        if (pid == 0) {
            // Child process
            compute_and_output_mandel_line(line, NPROCS);
            exit(1);
        }
    }

    // Wait for all children processes to finish

    for(i=0; i<NPROCS; i++) {
        if (wait(&status) == -1) {
            perror("wait");
            exit(1);
        }
    }

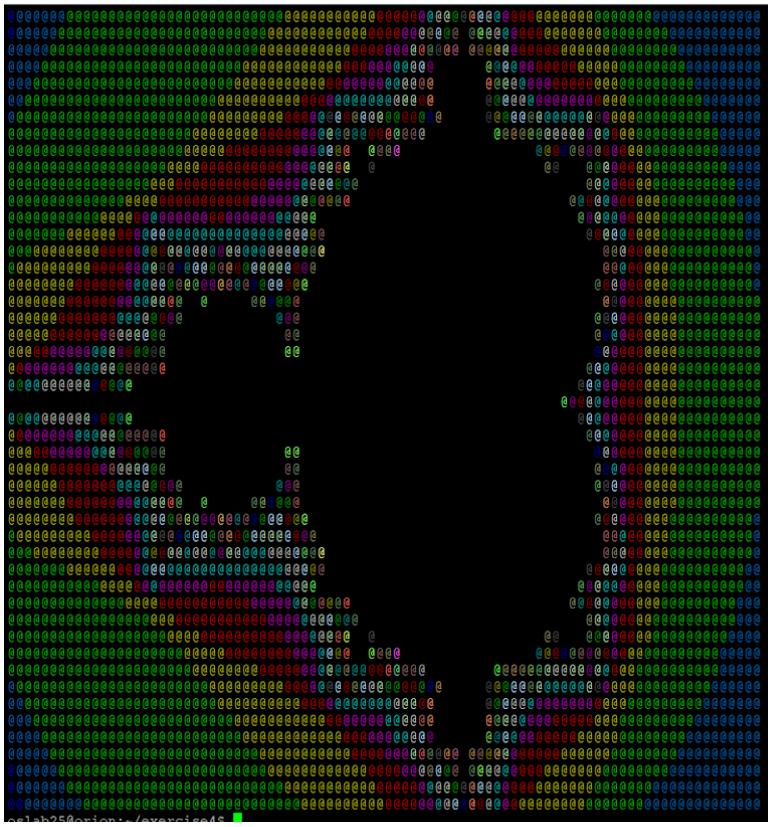
    // Clean up shared memory and semaphore

    for(i=0; i<NPROCS; i++)
        sem_destroy(&sem[i]);

    destroy_shared_memory_area(sem, NPROCS*sizeof(sem_t));
    reset_xterm_color(1);
    return 0;
}

```

Η έξοδος εκτέλεσης του προγράμματος:



```
celab250x1000000 /exercises12
```

### Ερωτήσεις:

- Τα νήματα είναι πιο γρήγορα και ελαφριά κατά τη δημιουργία και επικοινωνία μεταξύ τους, αφού έχουν κοινά δεδομένα και κοινή μνήμη, άρα είναι πιο γρήγορα, καθώς αποφεύγονται οι βαριές κλήσεις συστημάτων `mmap()` για τη δημιουργία των semaphores που χρησιμοποιούνται για το διαμοιρασμό μνήμης μεταξύ διεργασιών.

#### 1.2.2 Υλοποίηση χωρίς semaphores

Ο πηγαίος κώδικας της άσκησης:

```

/*
 * 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 <semaphore.h>
#include <errno.h>
#include <signal.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <inttypes.h>
#include "help.h"
#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

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

/*
 * Output at the terminal is is x_chars wide by y_chars long
 */
int y_chars = 50;
int x_chars = 90;

int NPROCS=3;

/*
 * 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, snewline, 1) != 1) {
        perror("compute_and_output_mandel_line: write newline");
        exit(1);
    }
}

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

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

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

    /* Create a shared, anonymous mapping for this number of pages */

    addr = mmap(NULL, pages*sysconf(_SC_PAGE_SIZE) , PROT_READ | PROT_WRITE, MAP_ANONYMOUS|MAP_SHARED, -1, 0);
    if (addr == MAP_FAILED) {
        die("mmap");
    }

    return addr;
}

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

```

        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 compute_and_output_mandel_line(int line, int NPROCS, int* shared_buffer) {
    int line_number, color_val[x_chars];

    for (line_number = line; line_number < y_chars; line_number += NPROCS) {
        compute_mandel_line(line_number, color_val);

        // Save the computed line in the shared buffer
        memcpy(&shared_buffer[line_number * x_chars], color_val, x_chars * sizeof(int));
    }
}

int main(int argc, char* argv[]) {
    int i, line;
    if (argc > 1) {
        NPROCS = atoi(argv[1]);
    }
    pid_t pid;

    // Create shared memory area
    int* shared_buffer = (int*)create_shared_memory_area(y_chars * x_chars * sizeof(int));

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

    // Fork the NPROCS processes
    for (i = 0; i < NPROCS; i++) {
        line = i;
        pid = fork();
        if (pid == -1) {
            perror("fork");
            exit(1);
        }
    }
}

int main(int argc, char* argv[]) {
    int i, line;
    if (argc > 1) {
        NPROCS = atoi(argv[1]);
    }
    pid_t pid;

    // Create shared memory area
    int* shared_buffer = (int*)create_shared_memory_area(y_chars * x_chars * sizeof(int));

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

    // Fork the NPROCS processes
    for (i = 0; i < NPROCS; i++) {
        line = i;
        pid = fork();
        if (pid == -1) {
            perror("fork");
            exit(1);
        }
        if (pid == 0) {
            // Child process
            compute_and_output_mandel_line(line, NPROCS, shared_buffer);
            exit(1);
        }
    }

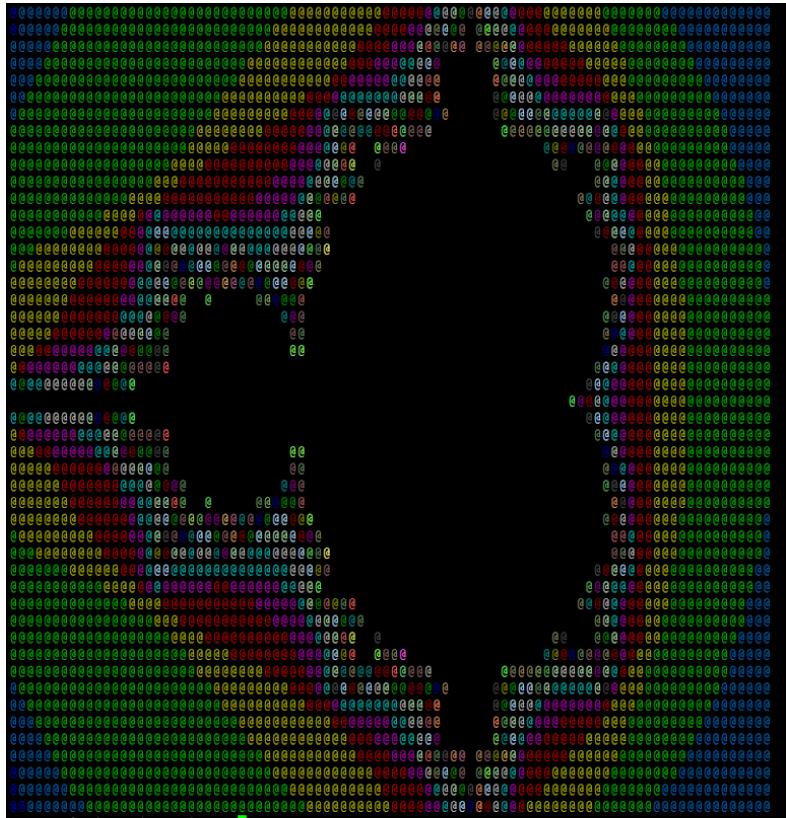
    // Wait for all children processes to finish
    for (i = 0; i < NPROCS; i++) {
        if (wait(NULL) == -1) {
            perror("wait");
            exit(1);
        }
    }

    // Print the buffer
    for (line = 0; line < y_chars; line++) {
        output_mandel_line(1, &shared_buffer[line * x_chars]);
    }

    // Clean up shared memory
    destroy_shared_memory_area(shared_buffer, y_chars * x_chars * sizeof(int));
    reset_xterm_color(1);
    return 0;
}

```

H έξοδος εκτέλεσης του προγράμματος:



1. Αυτή τη φορά δεν υπάρχει συγχρονισμός με την έννοια του να εκτελείται κάθε διεργασία περιμένοντας την σειρά της. Εδώ ο συγχρονισμός επιτυγχάνεται μέσω του προκαθορισμένου τρόπου με τον οποίο γράφουν οι διεργασίες στον shared buffer.

Στην περίπτωση που ο buffer είχε διαστάσεις NPROCS x x\_chars, δεν θα μπορούσαμε να τυπώσουμε τον buffer μια και έξω, αλλά η εκτύπωση γραμμών θα έπρεπε να γίνει ανά διαστήματα και αρκετές φορές, προκειμένου να εκτυπωθεί όλο το σχήμα. Μια λύση θα ήταν η χρήση των σημάτων SIGSTOP, προκειμένου η γονική διεργασία να περιμένει όλα της τα παιδιά να φτιάξουν την γραμμή τους και να την βάλουν στον buffer, και μετά να τα ενεργοποιήσει ένα ένα προκειμένου να τυπωθεί το σχήμα.