Λειτουργικά Συστήματα

4η αναφορά

Ονοματεπώνυμο	**Εμμανουηλίδης Εμμανουήλ**	**Λίτσος Ιωάννης**
Αριθμός Μητρώου	**03119435**	**03119135**
Ομάδα	**oslab61**	

Άσκηση 4:

Μηχανισμοί Εικονικής Μνήμης

ΑΣΚΗΣΕΙΣ

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

Step 1:

Τυπώνουμε το χάρτη της εικονικής μνήμης της τρέχουσας διεργασίας. Χρησιμοποιώντας τη συνάρτηση show_maps() παίρνουμε την ακόλουθη έξοδο:

```
oslaba61@os-node1:~/exercises/4th_exercise/mmap$ ./mmap

Step 1: Print the virtual address space map of this process [25126].


Virtual Memory Map of process [25126]:
00400000-00403000 r-xp 00000000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
008ee000-0090f000 rw-p 00000000 00:00 0                                  [heap]
7f09f0ca7000-7f09f0e48000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f0e48000-7f09f1048000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f1048000-7f09f104c000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104c000-7f09f104e000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104e000-7f09f1052000 rw-p 00000000 00:00 0
7f09f1052000-7f09f1073000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1265000-7f09f1268000 rw-p 00000000 00:00 0
7f09f126d000-7f09f1272000 rw-p 00000000 00:00 0
7f09f1272000-7f09f1273000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1273000-7f09f1274000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1274000-7f09f1275000 rw-p 00000000 00:00 0
7ffcd912f000-7ffcd9150000 rw-p 00000000 00:00 0                          [stack]
7ffcd91a6000-7ffcd91a9000 r--p 00000000 00:00 0                          [vvar]
7ffcd91a9000-7ffcd91ab000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

Η πρώτη στήλη του χάρτη μνήμης αντιστοιχεί στις διευθύνσεις , η δεύτερη μας πληροφορεί για τα δικαιώματα(read, write, execute) , ενώ στην τελευταία βλέπουμε τα paths, τη σωρό και τη στοίβα.

Step 2:

Με την κλήση συστήματος mmap() δεσμεύουμε buffer μεγέθους μίας σελίδας και τυπώνουμε ξανά τον χάρτη. Επίσης, καλούμε την συνάρτηση show_va_info() η οποία τυπώνει πληροφορίες για τη μνήμη που δεσμεύσαμε και παίρνουμε την ακόλουθη έξοδο:

```
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 [25126]:
00400000-00403000 r-xp 00000000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
008ee000-0090f000 rw-p 00000000 00:00 0                                  [heap]
7f09f0ca7000-7f09f0e48000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f0e48000-7f09f1048000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f1048000-7f09f104c000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104c000-7f09f104e000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104e000-7f09f1052000 rw-p 00000000 00:00 0
7f09f1052000-7f09f1073000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1265000-7f09f1268000 rw-p 00000000 00:00 0
7f09f126c000-7f09f1272000 rw-p 00000000 00:00 0
7f09f1272000-7f09f1273000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1273000-7f09f1274000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1274000-7f09f1275000 rw-p 00000000 00:00 0
7ffcd912f000-7ffcd9150000 rw-p 00000000 00:00 0                          [stack]
7ffcd91a6000-7ffcd91a9000 r--p 00000000 00:00 0                          [vvar]
7ffcd91a9000-7ffcd91ab000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------

7f09f126c000-7f09f1272000 rw-p 00000000 00:00 0
```

Ο χώρος εικονικών διευθύνσεων που δεσμεύσαμε είναι ο ακόλουθος:

7f09f126c000-7f09f1272000

Step 3:

Βρίσκουμε και τυπώνουμε τη φυσική διεύθυνση μνήμης στην οποία απεικονίζεται η εικονική διεύθυνση του buffer χρησιμοποιώντας την συνάρτηση get_physical_address() και παίρνουμε τη παρακάτω έξοδο:

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

VA[0x7f09f126d000] is not mapped; no physical memory allocated.
```

Παρατηρούμε ότι δεν έχει δεσμευτεί φυσική μνήμη διότι ο buffer που έχουμε δεσμεύσει είναι άδειος.

Step 4:

Γεμίζουμε με μηδενικά τον buffer μέσω της memset() και τυπώνουμε ξανά την εικονική και τη φυσική μνήμη του:

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

7f09f126c000-7f09f1272000 rw-p 00000000 00:00 0
Private buffer - Physical address
123e7c000
```

Παρατηρούμε ότι τώρα, εφόσον έχουμε αποθηκεύσει μηδενικά στον buffer, έχει δεσμευτεί φυσική μνήμη με διεύθυνση 123e7c000.

Step 5:

Στο βήμα αυτό ανοίγουμε το αρχείο file.txt μέσω της open και έπειτα μέσω της mmap() απεικονίζουμε το αρχείο στο χώρο διευθύνσεων της διεργασίας. Το περιεχόμενο του αρχείου έχει αποθηκευτεί στον buffer και μέσω αυτού τυπώνουμε το μήνυμα. Επίσης, εντοπίζουμε τη νέα απεικόνιση στο χάρτη μνήμης. Οι αντίστοιχες έξοδοι είναι οι εξής:

```
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 [25126]:
00400000-00403000 r-xp 00000000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
008ee000-0090f000 rw-p 00000000 00:00 0                                  [heap]
7f09f0ca7000-7f09f0e48000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f0e48000-7f09f1048000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f1048000-7f09f104c000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104c000-7f09f104e000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104e000-7f09f1052000 rw-p 00000000 00:00 0
7f09f1052000-7f09f1073000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1265000-7f09f1268000 rw-p 00000000 00:00 0
7f09f126b000-7f09f126c000 rw-p 00000000 00:00 0
7f09f126c000-7f09f126d000 r--s 00000000 00:21 1981593                    /home/oslab/oslaba61/exercises/4th_exercise/mmap/file.txt
7f09f126d000-7f09f1272000 rw-p 00000000 00:00 0
7f09f1272000-7f09f1273000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1273000-7f09f1274000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1274000-7f09f1275000 rw-p 00000000 00:00 0
7ffcd912f000-7ffcd9150000 rw-p 00000000 00:00 0                          [stack]
7ffcd91a6000-7ffcd91a9000 r--p 00000000 00:00 0                          [vvar]
7ffcd91a9000-7ffcd91ab000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------

7f09f126c000-7f09f126d000 r--s 00000000 00:21 1981593                    /home/oslab/oslaba61/exercises/4th_exercise/mmap/file.txt
File buffer - Physical address
117c99000
```

Step 6:

Δεσμεύουμε ένα νέο buffer , διαμοιραζόμενο αυτή τη φορά μεταξύ των διεργασιών , μέγεθος μιας σελίδας χρησιμοποιώντας στις παραμέτρους της mmap() το όρισμα MAP_SHARED αντί του MAP_PRIVATE . Η νέα απεικόνιση στο χάρτη μνήμης έχει ως εξής:

```
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 [25126]:
00400000-00403000 r-xp 00000000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
008ee000-0090f000 rw-p 00000000 00:00 0                                  [heap]
7f09f0ca7000-7f09f0e48000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f0e48000-7f09f1048000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f1048000-7f09f104c000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104c000-7f09f104e000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104e000-7f09f1052000 rw-p 00000000 00:00 0
7f09f1052000-7f09f1073000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1265000-7f09f1268000 rw-p 00000000 00:00 0
7f09f126a000-7f09f126b000 rw-p 00000000 00:00 0
7f09f126b000-7f09f126c000 rw-s 00000000 00:04 3870211                    /dev/zero (deleted)
7f09f126c000-7f09f126d000 r--s 00000000 00:21 1981593                    /home/oslab/oslaba61/exercises/4th_exercise/mmap/file.txt
7f09f126d000-7f09f1272000 rw-p 00000000 00:00 0
7f09f1272000-7f09f1273000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1273000-7f09f1274000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1274000-7f09f1275000 rw-p 00000000 00:00 0
7ffcd912f000-7ffcd9150000 rw-p 00000000 00:00 0                          [stack]
7ffcd91a6000-7ffcd91a9000 r--p 00000000 00:00 0                          [vvar]
7ffcd91a9000-7ffcd91ab000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------


7f09f126b000-7f09f126c000 rw-s 00000000 00:04 3870211                    /dev/zero (deleted)
Shared buffer - Physical address
1265bb000
```

Παρατηρούμε ότι στην στήλη με τα δικαιώματα υπάρχει το γράμμα «s» που μας πληροφορεί ότι η μνήμη είναι διαμοιραζόμενη.

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

Step 7:

Τυπώνουμε το χάρτη εικονικής μνήμης της διεργασίας πατέρα και της διεργασίας παιδιού:

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

Virtual memory map for Parent:

Virtual Memory Map of process [25126]:
00400000-00403000 r-xp 00000000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
008ee000-0090f000 rw-p 00000000 00:00 0                                  [heap]
7f09f0ca7000-7f09f0e48000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f0e48000-7f09f1048000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f1048000-7f09f104c000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104c000-7f09f104e000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104e000-7f09f1052000 rw-p 00000000 00:00 0
7f09f1052000-7f09f1073000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1265000-7f09f1268000 rw-p 00000000 00:00 0
7f09f126a000-7f09f126b000 rw-p 00000000 00:00 0
7f09f126b000-7f09f126c000 rw-s 00000000 00:04 3870211                    /dev/zero (deleted)
7f09f126c000-7f09f126d000 r--s 00000000 00:21 1981593                    /home/oslab/oslaba61/exercises/4th_exercise/mmap/file.txt
7f09f126d000-7f09f1272000 rw-p 00000000 00:00 0
7f09f1272000-7f09f1273000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1273000-7f09f1274000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1274000-7f09f1275000 rw-p 00000000 00:00 0
7ffcd912f000-7ffcd9150000 rw-p 00000000 00:00 0                          [stack]
7ffcd91a6000-7ffcd91a9000 r--p 00000000 00:00 0                          [vvar]
7ffcd91a9000-7ffcd91ab000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------

Virtual memory map for Child:

Virtual Memory Map of process [26184]:
00400000-00403000 r-xp 00000000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
008ee000-0090f000 rw-p 00000000 00:00 0                                  [heap]
7f09f0ca7000-7f09f0e48000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f0e48000-7f09f1048000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f1048000-7f09f104c000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104c000-7f09f104e000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104e000-7f09f1052000 rw-p 00000000 00:00 0
7f09f1052000-7f09f1073000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1265000-7f09f1268000 rw-p 00000000 00:00 0
7f09f126a000-7f09f126b000 rw-p 00000000 00:00 0
7f09f126b000-7f09f126c000 rw-s 00000000 00:04 3870211                    /dev/zero (deleted)
7f09f126c000-7f09f126d000 r--s 00000000 00:21 1981593                    /home/oslab/oslaba61/exercises/4th_exercise/mmap/file.txt
7f09f126d000-7f09f1272000 rw-p 00000000 00:00 0
7f09f1272000-7f09f1273000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1273000-7f09f1274000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1274000-7f09f1275000 rw-p 00000000 00:00 0
7ffcd912f000-7ffcd9150000 rw-p 00000000 00:00 0                          [stack]
7ffcd91a6000-7ffcd91a9000 r--p 00000000 00:00 0                          [vvar]
7ffcd91a9000-7ffcd91ab000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

Παρατηρούμε ότι ο χάρτης μνήμης είναι ίδιος και για τους δύο. Αυτό συμβαίνει διότι όταν καλούμε την fork() ο χάρτης μνήμης της διεργασίας πατέρα αντιγράφεται στη διεργασία παιδί.

Step 8:

Η φυσική διεύθυνση του private buffer στην κύρια μνήμη για τις διεργασίες πατέρα και παιδί είναι η ακόλουθη:

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

Parent private buffer - Virtual Address:
7f09f126d000-7f09f1272000 rw-p 00000000 00:00 0
Parent private buffer - Physical Address
123e7c000
Child private buffer - Virtual Address:
7f09f126d000-7f09f1272000 rw-p 00000000 00:00 0
Child private buffer - Physical Address
123e7c000
```

Αμέσως μετά το fork() η νέα διεργασία είναι αντίγραφο της παλιάς , κληρονομεί όλα τα ανοιχτά αρχεία και από εκεί και πέρα εκτελείται ανεξάρτητα. Γι’ αυτό , μέχρι το σημείο αυτό, η φυσική και εικονική μνήμη είναι ίδιες.

Step 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?

Parent private buffer - Virtual Address:
7f09f126d000-7f09f1272000 rw-p 00000000 00:00 0
Parent private buffer - Physical Address
123e7c000
CHild private buffer - Virtual Address:
7f09f126d000-7f09f1272000 rw-p 00000000 00:00 0
Child private buffer - Physical Address
afd3e000
```

Παρατηρούμε ότι η φυσική μνήμη του πατέρα διαφέρει από αυτή του παιδιού και αυτό οφείλεται στο copy-on-write. Όπως αναφέραμε νωρίτερα η γονική και θυγατρική διεργασία διαμοιράζονται αρχικά τις ίδιες σελίδες οι οποίες σημαίνονται ως σελίδες copy-on-write. Αυτό σημαίνει ότι αν οποιαδήποτε από τις δύο διεργασίες γράψει σε μία διαμοιραζόμενη σελίδα τότε δημιουργείται ένα αντίγραφο της διαμοιραζόμενης σελίδας και ότι πλέον η τροποποιημένη σελίδα αντιστοιχεί στην διεργασία που έκανε την αλλαγή(στο παράδειγμα μας η διεργασία παιδί) και η αρχική στην άλλη.

Step 10:

Κάνουμε την αντίστοιχη διαδικασία χρησιμοποιώντας τον shared_buffer αντί για τον private και παίρνουμε την ακόλουθη έξοδο:

```
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?

Parent shared buffer - Virtual Address:
7f09f126b000-7f09f126c000 rw-s 00000000 00:04 3870211                    /dev/zero (deleted)
Parent shared buffer - Physical Address:
1265bb000
Child shared buffer - Virtual Address:
7f09f126b000-7f09f126c000 rw-s 00000000 00:04 3870211                    /dev/zero (deleted)
CHild shared buffer - Physical Address:
1265bb000
```

Τώρα βλέπουμε ότι η φυσική μνήμη είναι και πάλι ίδια για το πατέρα και το παιδί, το οποίο οφείλεται στο γεγονός ότι η μνήμη είναι διαμοιραζόμενη.(/dev/zero (deleted))

Step 11-12:

Μέσω της mprotect() απαγορεύουμε τις εγγραφές στο buffer για τη διεργασία παιδί και τυπώνουμε την απεικόνιση του στο χάρτη μνήμης των διεργασιών :

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

Virtual memory map for Parent:

Virtual Memory Map of process [25126]:
00400000-00403000 r-xp 00000000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
008ee000-0090f000 rw-p 00000000 00:00 0                                  [heap]
7f09f0ca7000-7f09f0e48000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f0e48000-7f09f1048000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f1048000-7f09f104c000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104c000-7f09f104e000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104e000-7f09f1052000 rw-p 00000000 00:00 0
7f09f1052000-7f09f1073000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1265000-7f09f1268000 rw-p 00000000 00:00 0
7f09f126a000-7f09f126b000 rw-p 00000000 00:00 0
7f09f126b000-7f09f126c000 rw-s 00000000 00:04 3870211                    /dev/zero (deleted)
7f09f126c000-7f09f126d000 r--s 00000000 00:21 1981593                    /home/oslab/oslaba61/exercises/4th_exercise/mmap/file.txt
7f09f126d000-7f09f1272000 rw-p 00000000 00:00 0
7f09f1272000-7f09f1273000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1273000-7f09f1274000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1274000-7f09f1275000 rw-p 00000000 00:00 0
7ffcd912f000-7ffcd9150000 rw-p 00000000 00:00 0                          [stack]
7ffcd91a6000-7ffcd91a9000 r--p 00000000 00:00 0                          [vvar]
7ffcd91a9000-7ffcd91ab000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

```
7f09f126b000-7f09f126c000 rw-s 00000000 00:04 3870211                    /dev/zero (deleted)
Virtual memory map for Child:

Virtual Memory Map of process [26184]:
00400000-00403000 r-xp 00000000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 1981584                            /home/oslab/oslaba61/exercises/4th_exercise/mmap/mmap
008ee000-0090f000 rw-p 00000000 00:00 0                                  [heap]
7f09f0ca7000-7f09f0e48000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f0e48000-7f09f1048000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f1048000-7f09f104c000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104c000-7f09f104e000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f09f104e000-7f09f1052000 rw-p 00000000 00:00 0
7f09f1052000-7f09f1073000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1265000-7f09f1268000 rw-p 00000000 00:00 0
7f09f126a000-7f09f126b000 rw-p 00000000 00:00 0
7f09f126b000-7f09f126c000 ---s 00000000 00:04 3870211                    /dev/zero (deleted)
7f09f126c000-7f09f126d000 r--s 00000000 00:21 1981593                    /home/oslab/oslaba61/exercises/4th_exercise/mmap/file.txt
7f09f126d000-7f09f1272000 rw-p 00000000 00:00 0
7f09f1272000-7f09f1273000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1273000-7f09f1274000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f09f1274000-7f09f1275000 rw-p 00000000 00:00 0
7ffcd912f000-7ffcd9150000 rw-p 00000000 00:00 0                          [stack]
7ffcd91a6000-7ffcd91a9000 r--p 00000000 00:00 0                          [vvar]
7ffcd91a9000-7ffcd91ab000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------

7f09f126b000-7f09f126c000 ---s 00000000 00:04 3870211                    /dev/zero (deleted)
oslaba61@os-node1:~/exercises/4th_exercise/mmap$
```

Παρατηρούμε ότι στη στήλη των δικαιωμάτων στο χάρτη μνήμης της διεργασίας παιδί υπάρχει μόνο το s όσον αφορά τον shared_buffer πράγμα που επιβεβαιώνει την απαγόρευση που εφαρμόσαμε. Τέλος, χρησιμοποιούμε την συνάρτηση munmap() και αποδεσμεύουμε όλους τους buffers από τις διεργασίες.

Ο κώδικας της άσκησης είναι:

```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 <inttypes.h>

#include "help.h"

#define RED     "\033[31m"
```

```c
#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("Virtual memory map for Child:\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("Child private buffer - Virtual Address:\n");
    show_va_info((uint64_t) heap_private_buf);
    printf("Child private buffer - Physical Address\n");
    pa = get_physical_address((uint64_t)heap_private_buf);
    printf("%" PRIx64 "\n", pa);
```

```
    /*
     * Step 9 - Child
     */
    if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
    /*
     * TODO: Write your code here to complete child's part of Step 9.
     */
    memset(heap_private_buf, 0, buffer_size);

    printf("CHild private buffer - Virtual Address:\n");
    show_va_info((uint64_t) heap_private_buf);
    printf("Child private buffer - Physical Address\n");
    pa = get_physical_address((uint64_t)heap_private_buf);
    printf("%" PRIx64 "\n", pa);

    /*
     * Step 10 - Child
     */
    if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
    /*
     * TODO: Write your code here to complete child's part of Step 10.
     */

    memset(heap_shared_buf, 1, buffer_size);

    printf("Child shared buffer - Virtual Address:\n");
    show_va_info((uint64_t)heap_shared_buf);
    printf("CHild shared buffer - Physical Address:\n");
    pa = get_physical_address((uint64_t)heap_shared_buf);
    printf("%" PRIx64 "\n", pa);

    /*
     * Step 11 - Child
     */
    if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
    /*
     * TODO: Write your code here to complete child's part of Step 11.
     */
    mprotect(heap_shared_buf, buffer_size, PROT_NONE);
    printf("Virtual memory map for 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.
     */
     printf("Virtual memory map for Parent:\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("Parent private buffer - Virtual Address:\n");
    show_va_info((uint64_t) heap_private_buf);
    printf("Parent private buffer - Physical Address\n");
    pa = get_physical_address((uint64_t)heap_private_buf);
    printf("%" PRIx64 "\n", pa);

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


    /*
     * Step 9: Write to heap_private_buf. What happened?
     * Step 9 - Parent
     */
    printf(RED "\nStep 9: Write to the private buffer from the child and "
        "repeat step 8. What happened?\n" RESET);
    press_enter();

    /*
     * TODO: Write your code here to complete parent's part of Step 9.
     */
     printf("Parent private buffer - Virtual Address:\n");
     show_va_info((uint64_t) heap_private_buf);
     printf("Parent private buffer - Physical Address\n");
     pa = get_physical_address((uint64_t)heap_private_buf);
     printf("%" PRIx64 "\n", pa);

    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("Parent shared buffer - Virtual Address:\n");
   show_va_info((uint64_t)heap_shared_buf);
   printf("Parent shared buffer - Physical Address:\n");
   pa = get_physical_address((uint64_t)heap_shared_buf);
   printf("%" PRIx64 "\n", pa);

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


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

   /*
    * TODO: Write your code here to complete parent's part of Step 11.
    */
   printf("Virtual memory map for Parent:\n");
   show_maps();
   show_va_info((uint64_t)heap_shared_buf);

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

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

    /*
     * TODO: Write your code here to complete parent's part of Step 12.
     */
     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;
    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, -1, 0);
```

```c
 if(heap_private_buf == MAP_FAILED) {
     perror("erron on step 2\n");
     exit(-1);
 }
 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.
 */
 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.
 */
 memset(heap_private_buf, 0, buffer_size);
 show_va_info((uint64_t)heap_private_buf);
 printf("Private buffer - Physical address\n");
 pa = get_physical_address((uint64_t)heap_private_buf);
 printf("%" PRIx64 "\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) { perror("error opening file\n"); }

    file_shared_buf = mmap(NULL, buffer_size, PROT_READ, MAP_SHARED, fd, 0);
    if (file_shared_buf == MAP_FAILED) {
                perror("error on step 5\n");
                exit(-1);
    }

    int i;
    char c;
    for(i = 0; i < (int)buffer_size; i++) {
        c = *(file_shared_buf+i);
        if(c != EOF) printf("%c", c);
        else break;
    }

    show_maps();
    show_va_info((uint64_t)file_shared_buf);
    printf("File buffer - Physical address\n");
    pa = get_physical_address((uint64_t) file_shared_buf);
    printf("%" PRIx64 "\n", 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.
     */
    heap_shared_buf = mmap(NULL, buffer_size, PROT_READ|PROT_WRITE,
MAP_SHARED|MAP_ANONYMOUS, -1, 0);

    if (heap_private_buf == MAP_FAILED) {
        perror("error on step 6\n");
        exit(-1);
    }

    memset(heap_shared_buf, 5, buffer_size);
    show_maps();
```

```
    show_va_info((uint64_t)heap_shared_buf);
    printf("Shared buffer - Physical address\n");
    pa = get_physical_address((uint64_t) heap_shared_buf);
    printf("%" PRIx64 "\n", pa);

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

    parent(p);

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

1.2 Παράλληλος υπολογισμός Mandelbrot με διεργασίες αντί για νήματα

Στην άσκηση αυτή ζητείται η τροποποίηση του προγράμματος υπολογισμού του MandelBrot set, της άσκησης 3, ώστε να χρησιμοποιεί διεργασίες αντί για threads.

1.2.1 Semaphores πάνω από διαμοιραζόμενη μνήμη

Στο ερώτημα αυτό γίνεται και πάλι χρήση των semaphores, τα οποία όμως τώρα βρίσκονται σε κοινή μνήμη στην οποία έχουν πρόσβαση όλες οι διεργασίες. Ο κώδικας είναι παρόμοιος με αυτόν της προηγούμενης άσκησης με τις εξής διαφορές:

- Στην main δεσμεύουμε χώρο για τους semaphores μέσω της create_shared_memory_area , την οποία έχουμε συμπληρώσει με την εντολή :

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

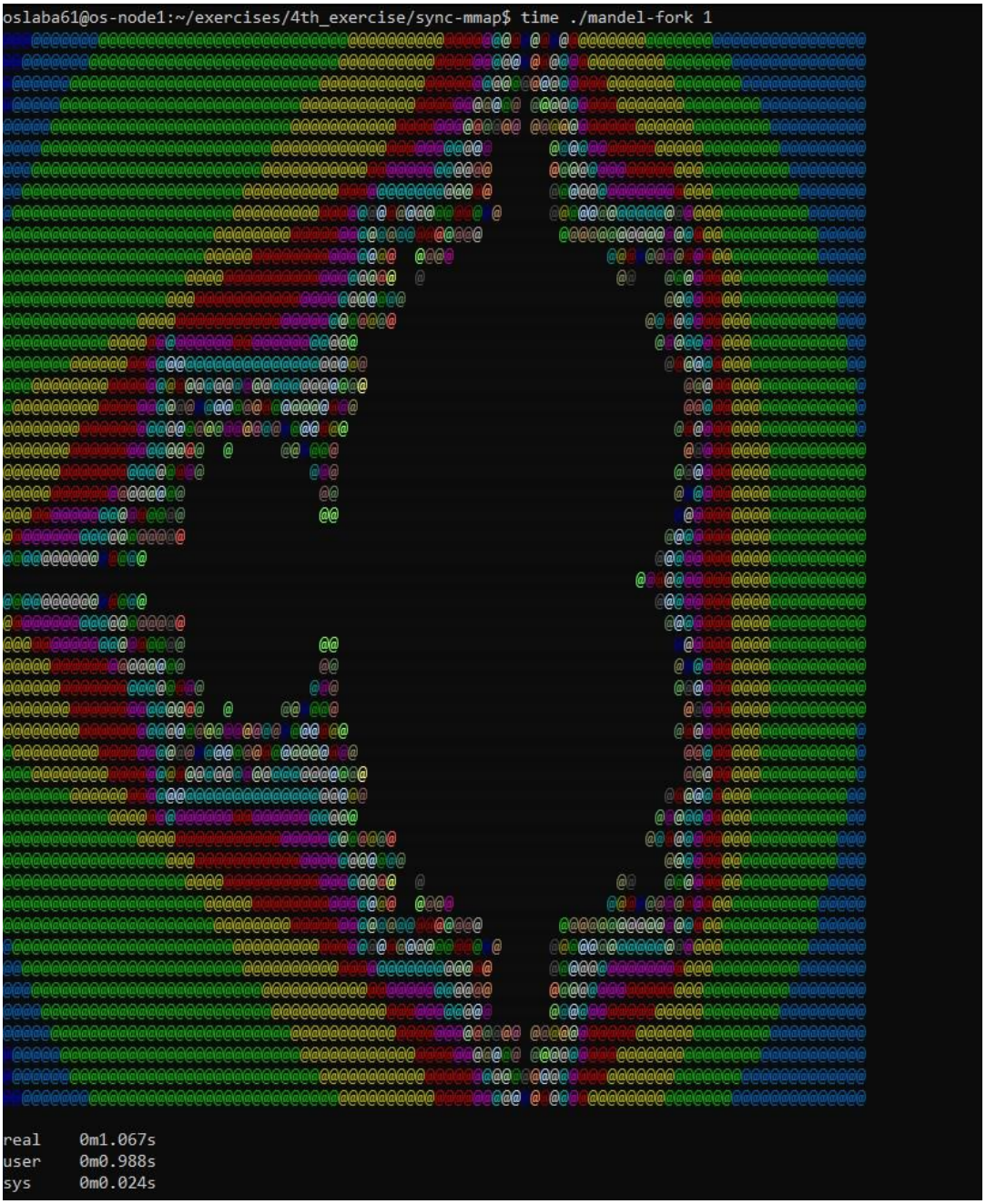
  Αφού δεσμεύσουμε χώρο με διαμοιραζόμενο τρόπο για τους semaphores τους αρχικοποιούμε όπως ακριβώς και στην προηγούμενη άσκηση.

- Κάνουμε fork() NPROCS φορές , μία για κάθε διεργασία και καλούμε για κάθε διεργασία την compute_and_output_mandel_line. Η compute_and_output_mandel_line

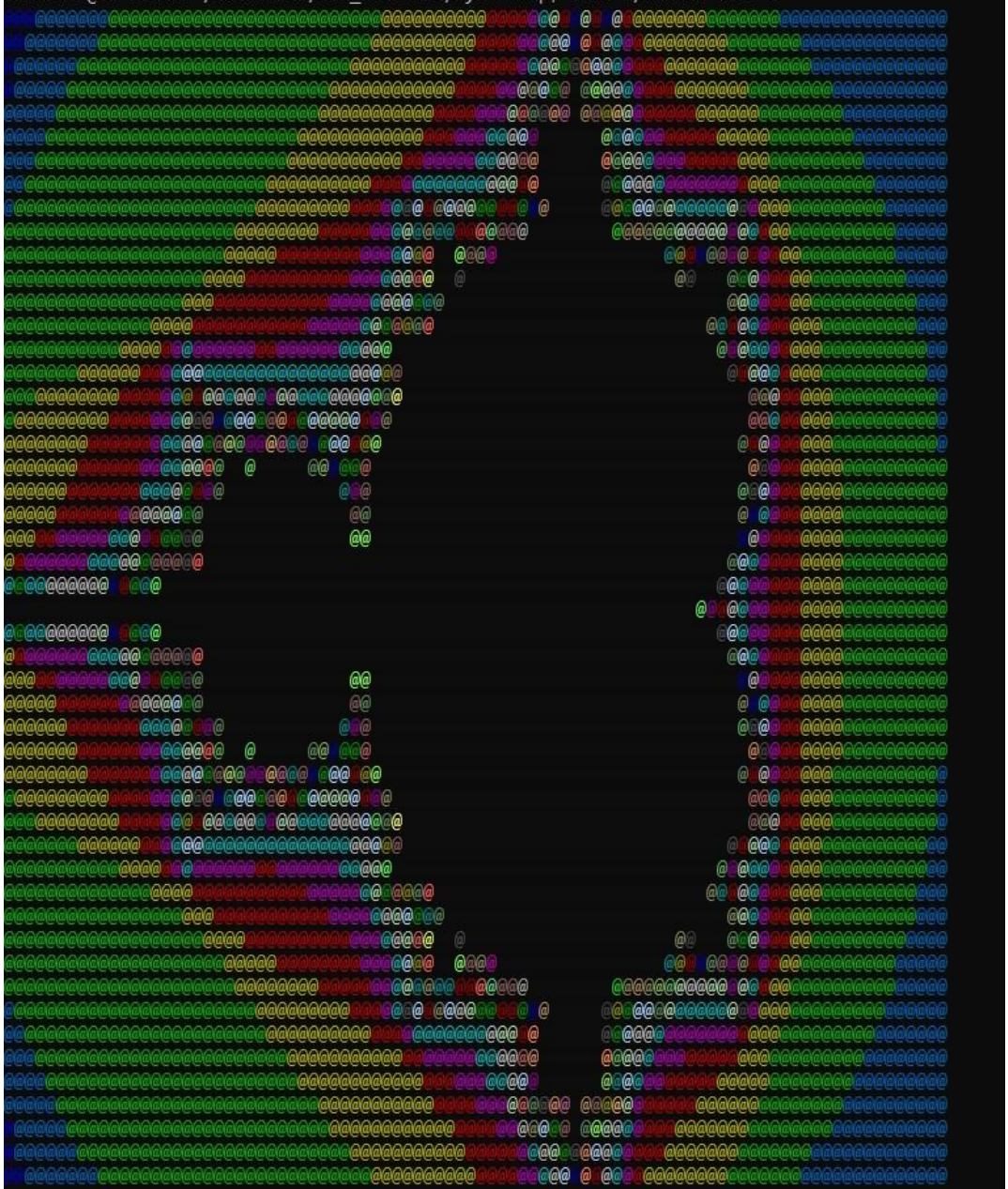
παραμένει ίδια, δηλαδή ο υπολογισμός των γραμμών γίνεται παράλληλα αλλά η εκτύπωση ελέγχεται μέσω των σημαφόρων ώστε να γίνεται με τη σειρά.

- Στην main μέσω της wait περιμένουμε να τερματίσουν όλες οι διεργασίες παιδιά
- Τέλος, καταστρέφουμε τους σημαφόρους με την sem_destroy και αποδεσμεύουμε την κοινόχρηστη μνήμη μέσω της destroy_shared_memory_area.

Έχουμε αντίστοιχες εξόδους και χρόνους εκτέλεσης με την προηγούμενη φορά. Ακολουθούν δύο ενδεικτικές έξοδοι με 1 και 8 διεργασίες :



```
oslaba61@os-node1:~/exercises/4th_exercise/sync-mmap$ time ./mandel-fork 8
```



```
real    0m0.163s
user    0m0.972s
sys     0m0.028s
```

Ο κώδικας του προγράμματος είναι ο εξής :

```c
/*
 * 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 <errno.h>

#include <semaphore.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

sem_t *semaphores;
int NPROCS;

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

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

/*
 * The part of the complex plane to be drawn:
 * upper left corner is (xmin, ymax), lower right corner is (xmax, ymin)
*/
double xmin = -1.8, xmax = 1.0;
```

```
double ymin = -1.0, ymax = 1.0;

/*
 * Every character in the final output is
 * xstep x ystep units wide on the complex plane.
 */
double xstep;
double ystep;

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

    int n;
    int val;

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

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

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

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

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

```
{
    int i;

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

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

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


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

    int i;
    for(i = line; i < y_chars; i += NPROCS) {
        compute_mandel_line(i, color_val);
        sem_wait(&semaphores[i % NPROCS]);
        // critical section
        output_mandel_line(fd, color_val);
        sem_post(&semaphores[(i+1) % NPROCS]);
    }
    exit(1);
    return NULL;
}

/*
 * 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, PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1,0);

    //

    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 usage(char *argv0)
{
    fprintf(stderr, "Usage: %s wrong input!\n\n"
        "Exactly one argument required:\n"
        "    NTHREADS: The number of threads to create.\n",
        argv0);
    exit(1);
}


void *safe_malloc(size_t size)
{
    void *p;

    if ((p = malloc(size)) == NULL) {
        fprintf(stderr, "Out of memory, failed to allocate %zd bytes\n",
            size);
        exit(1);
    }

    return p;
}


int safe_atoi(char *s, int *val)
{
    long l;
    char *endp;

    l = strtol(s, &endp, 10);
    if (s != endp && *endp == '\0') {
        *val = l;
        return 0;
    } else
        return -1;
}


int main(int argc, char *argv[])
{
```

```
    int i;
    pid_t *pid;

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

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

    pid = safe_malloc(sizeof(pid_t));

    unsigned int addr_size = NPROCS * sizeof(sem_t);
    semaphores = create_shared_memory_area(addr_size);

    /* initialize semaphores */

    // #semaphores = NPROCS

    /* 1st semaphore needs to start with value = 1 */
    sem_init(&semaphores[0],1,1);

    // the rest of them start with value = 0
    for(i = 1; i< NPROCS; i++)
        sem_init(&semaphores[i],1,0);

    //create processes
    for(i = 0; i < NPROCS; i++) {
        pid[i] = fork();
        if(pid[i] < 0) {
            perror("fork failed\n");
        }
        if(pid[i] == 0) {
            /* IN child process */
            compute_and_output_mandel_line(1, i);
        }
    }

    int status;
```

```
    /*
    Wait for all processes to terminate
    */

    for(i = 0; i < NPROCS; i++) {
        pid[i] = wait(&status);
    }

    // destroy semaphores
    for(i = 0; i < NPROCS; i++)
        sem_destroy(&semaphores[i]);

    // destroy shared memory area
    destroy_shared_memory_area(semaphores, addr_size);
    reset_xterm_color(1);
    return 0;
}
```

Ερωτήσεις

1. Η υλοποίηση με threads περιμένουμε να έχει καλύτερη απόδοση από ότι η υλοποίηση με processes. Αρχικά, η δημιουργία και η χρήση των threads απαιτεί λιγότερους πόρους σε σχέση με μία διεργασία. Επίσης, τα νήματα έχουν εξ ορισμού κοινή μνήμη μεταξύ τους (τα νήματα που ανήκουν στην ίδια διεργασία), ενώ οι διεργασίες δεν έχουν και για αυτό έπρεπε στο πρόγραμμα να υλοποιήσουμε εμείς τους σημαφόρους με τέτοιο τρόπο ώστε να μοιράζονται μεταξύ των διεργασιών. Τέλος, το context switch μεταξύ των νημάτων είναι ταχύτερη σε σχέση με τις διεργασίες.

   Η χρήση διαμοιραζόμενης μνήμης στους σημαφόρους βελτιώνει την επίδοση της υλοποίησης. Εναλλακτική επιλογή θα ήταν η χρήση σωληνώσεων η οποία όμως είναι αρκετά πιο αργή υλοποίηση.
2. Για να διαμοιράσει το mmap interface μνήμη μεταξύ των διεργασιών πρέπει πρώτα μία διεργασία να ορίσει έναν εικονικό κοινό χώρο διευθύνσεων μέσω της mmap. Αρχικά, αυτός ο χώρος δεν διαμοιράζεται με κανέναν έως ότου να πραγματοποιηθεί μία fork(). Τότε το παιδί μπορεί να χρησιμοποιήσει αυτό το κοινό χώρο διευθύνσεων. Για να διαμοιράζονται μνήμη δύο διεργασίες πρέπει να έχουν τον ίδιο εικονικό χώρο διευθύνσεων και αυτό μπορεί να συμβαίνει μόνο σε διεργασίες που έχουν κοινό πρόγονο. Επομένως, διεργασίες που δεν έχουν κοινό πρόγονο δεν μπορούν να διαμοιράζονται μνήμη μεταξύ τους.

1.2.2 Υλοποίηση χωρίς semaphores

Στο ερώτημα αυτό τροποποιούμε την υλοποίηση του προηγούμενου ερωτήματος ώστε να αποφευχθεί η χρήση semaphores. Συγκεκριμένα, οι διεργασίες παιδιά υπολογίζουν τις γραμμές του set όπως και πριν , αλλά αντί να τυπώνουν το αποτέλεσμα το γράφουν στην αντίστοιχη γραμμή ενός διαμοιραζόμενου buffer διαστάσεων x_chars × y_chars . Οι αλλαγές στον κώδικα έχουν ως εξής:

- Μέσω της create_shared_memory_area δημιουργούμε τον διαμοιραζόμενο διδιάστατο buffer και για κάθε διεργασία καλούμε την compute_and_output_mandel_line .
- Στην compute_and_output_mandel_line καλούμε μόνο την compute_mandel_line με παράμετρο τον αριθμό γραμμής i και την αντίστοιχη γραμμή του buffer buff[i], ώστε να γραφεί η αντίστοιχη γραμμή του set στον buffer.
- Στην main μέσω της wait περιμένουμε να τερματίσουν όλες οι διεργασίες όπως και πριν και στη συνέχεια για κάθε γραμμή από 0 έως y_chars -1 καλούμε την output_mandel_line χρησιμοποιώντας τον buffer για να τυπώσει τις γραμμές και στο τέλος καταστρέφουμε την διαμοιραζόμενη μνήμη.

Ο κώδικας της άσκησης είναι ο ακόλουθος:

```
/*
 * 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 <errno.h>

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

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

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000
```

```
int NPROCS;
int **buff;

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

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

/*
 * The part of the complex plane to be drawn:
 * upper left corner is (xmin, ymax), lower right corner is (xmax, ymin)
*/
double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;

/*
 * Every character in the final output is
 * xstep x ystep units wide on the complex plane.
 */
double xstep;
double ystep;

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

    int n;
    int val;

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

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

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

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

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

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

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

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


void *compute_and_output_mandel_line(int fd, int line)
{
    /*
```

```
     * A temporary array, used to hold color values for the line being drawn
     */
    int i;
    for(i = line; i < y_chars; i += NPROCS) {
        compute_mandel_line(i, buff[i]);
    }
        exit(1);
    return NULL;
}


/*
 * 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, PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1,0);

    //

    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 usage(char *argv0)
{
    fprintf(stderr, "Usage: %s wrong input!\n\n"
        "Exactly one argument required:\n"
        "    NTHREADS: The number of threads to create.\n",
        argv0);
    exit(1);
}


void *safe_malloc(size_t size)
{
    void *p;

    if ((p = malloc(size)) == NULL) {
        fprintf(stderr, "Out of memory, failed to allocate %zd bytes\n",
            size);
        exit(1);
    }

    return p;
}


int safe_atoi(char *s, int *val)
{
```

```
    long l;
    char *endp;

    l = strtol(s, &endp, 10);
    if (s != endp && *endp == '\0') {
        *val = l;
        return 0;
    } else
        return -1;
}


int main(int argc, char *argv[])
{
    int i;
    pid_t *pid;

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

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

    pid = create_shared_memory_area(NPROCS*sizeof(pid_t));

    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(int));
    }

    //create processes
    for(i = 0; i < NPROCS; i++) {
        pid[i] = fork();
        if(pid[i] < 0) {
            perror("fork failed\n");
        }
        if(pid[i] == 0) {
            /* IN child process */
```

```
            compute_and_output_mandel_line(1, i);
        }
    }

    int status;

    /*
    Wait for all processes to terminate
    */

    for(i = 0; i < NPROCS; i++) {
        pid[i] = wait(&status);
    }

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

    // destroy shared memoery area

    for(i = 0; i < y_chars; i++) {
        destroy_shared_memory_area(buff, x_chars * sizeof(int));
    }
    destroy_shared_memory_area(buff, y_chars * sizeof(int));

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

Η έξοδος και οι χρόνοι εκτέλεσης είναι ανάλογοι με την προηγούμενη άσκηση.

Ερώτηση

1. Σε αυτή την υλοποίηση ο συγχρονισμός επιτυγχάνεται με τον τρόπο που περιγράφεται στην συνέχεια. Αρχικά όλες οι διεργασίες διαμοιράζονται τον buffer και έτσι κάθε μία έχει το δικαίωμα να γράψει σε αυτόν. Κάθε διεργασία υπολογίζει τις γραμμές που της έχουν ανατεθεί και τις γράφει στις αντίστοιχες γραμμές-θέσεις του buffer . Με τον τρόπο που έχει υλοποιηθεί το πρόγραμμα είμαστε βέβαιοι ότι καμία διεργασία δεν θα γράψει ξανά στην ίδια θέση του buffer. Στο τέλος , περιμένουμε όλες οι διεργασίες να ολοκληρώσουν την εκτέλεση τους, δηλαδή να γράψουν τις γραμμές τους στον buffer , και αφού συμβεί αυτό εκτυπώνουν το περιεχόμενο του buffer και παίρνουμε το σωστό αποτέλεσμα.

Αν ο buffer είχε διαστάσεις NPROCS × x_chars θα έπρεπε κάθε φορά, αφού γινόταν ο υπολογισμός και η εγγραφή των NPROCS διαδοχικών γραμμών στον buffer , να εκτυπώναμε το περιεχόμενο του και έπειτα να γινόταν ο υπολογισμός και η εγγραφή των επόμενων NPROCS γραμμών κ.ο.κ . Διαφορετικά, θα γινόταν overwrite πάνω στον buffer και θα εκτυπώνονταν μόνο οι τελευταίες NPROCS γραμμές.