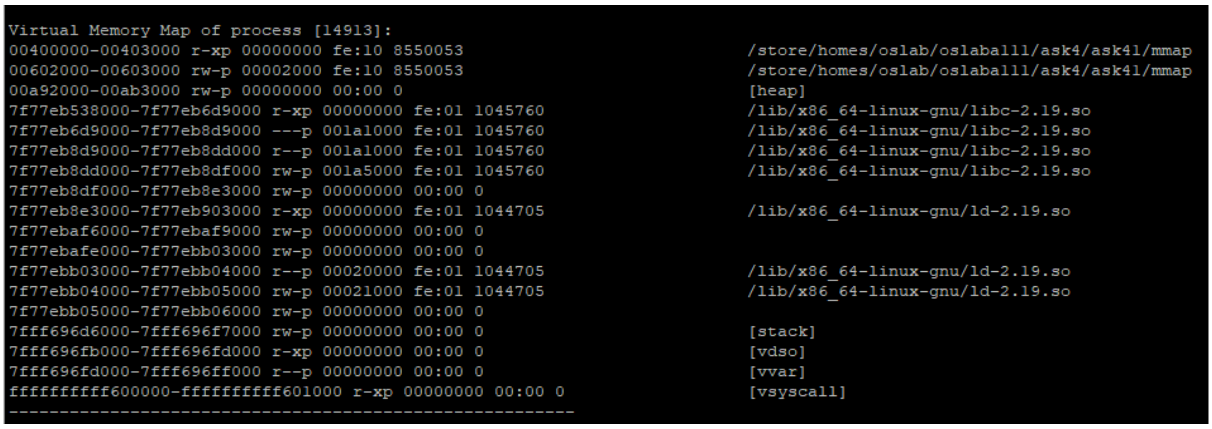
ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΗΛΙΑΣ ΝΤΟΝΤΟΡΟΣ
ΘΟΔΩΡΗΣ- ΙΩΑΝΝΗΣ ΚΙΤΣΗΣ

ΑΝΑΦΟΡΑ 4ης ΣΕΙΡΑΣ ΑΣΚΗΣΕΩΝ

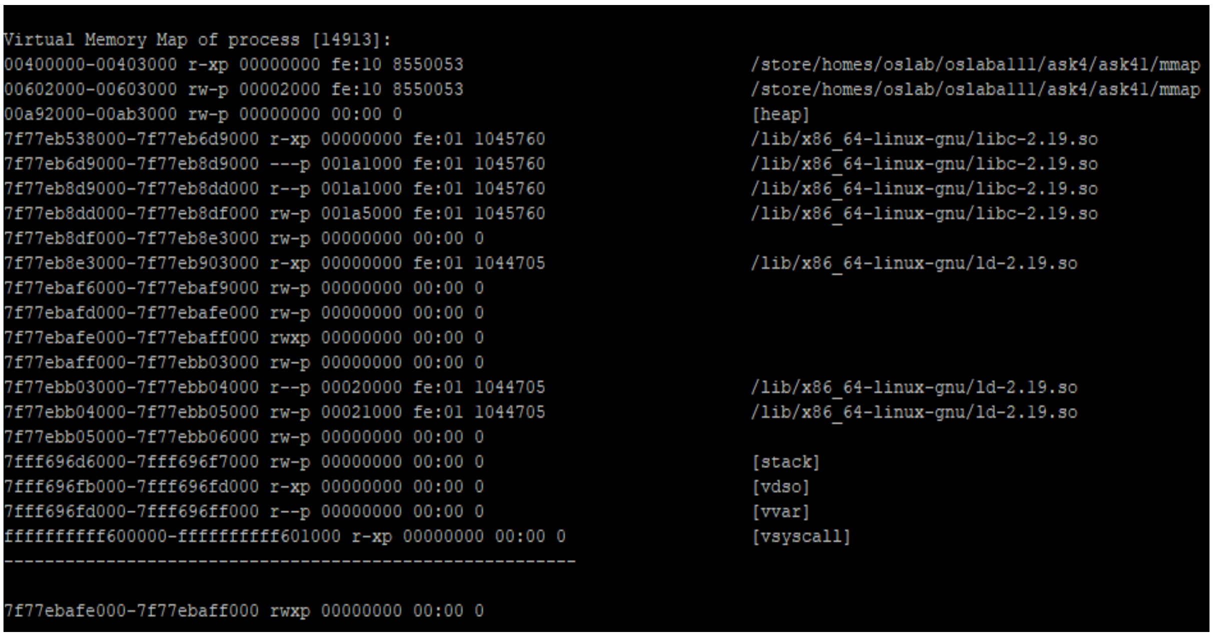
ΑΣΚΗΣΗ 1Η

4.1.1 Print the virtual address space map of this process [9283].
Με την εντολή show_maps() τυπώνουμε το χάρτη της εικονικής μνήμης της τρέχουσας διεργασίας.



```
Virtual Memory Map of process [14913]:
00400000-00403000 r-xp 00000000 fe:10 8550053                          /store/homes/oslab/oslaba11/ask4/ask41/mmap
00602000-00603000 rw-p 00002000 fe:10 8550053                          /store/homes/oslab/oslaba11/ask4/ask41/mmap
00a92000-00ab3000 rw-p 00000000 00:00 0                                [heap]
7f77eb538000-7f77eb6d9000 r-xp 00000000 fe:01 1045760                  /lib/x86_64-linux-gnu/libc-2.19.so
7f77eb6d9000-7f77eb8d9000 ---p 001a1000 fe:01 1045760                  /lib/x86_64-linux-gnu/libc-2.19.so
7f77eb8d9000-7f77eb8dd000 r--p 001a1000 fe:01 1045760                  /lib/x86_64-linux-gnu/libc-2.19.so
7f77eb8dd000-7f77eb8df000 rw-p 001a5000 fe:01 1045760                  /lib/x86_64-linux-gnu/libc-2.19.so
7f77eb8df000-7f77eb8e3000 rw-p 00000000 00:00 0
7f77eb8e3000-7f77eb903000 r-xp 00000000 fe:01 1044705                  /lib/x86_64-linux-gnu/ld-2.19.so
7f77ebaf6000-7f77ebaf9000 rw-p 00000000 00:00 0
7f77ebafe000-7f77ebb03000 rw-p 00000000 00:00 0
7f77ebb03000-7f77ebb04000 r--p 00020000 fe:01 1044705                  /lib/x86_64-linux-gnu/ld-2.19.so
7f77ebb04000-7f77ebb05000 rw-p 00021000 fe:01 1044705                  /lib/x86_64-linux-gnu/ld-2.19.so
7f77ebb05000-7f77ebb06000 rw-p 00000000 00:00 0
7fff696d6000-7fff696f7000 rw-p 00000000 00:00 0                        [stack]
7fff696fb000-7fff696fd000 r-xp 00000000 00:00 0                        [vdso]
7fff696fd000-7fff696ff000 r--p 00000000 00:00 0                        [vvar]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                [vsyscall]
--------------------------------------------------------
```

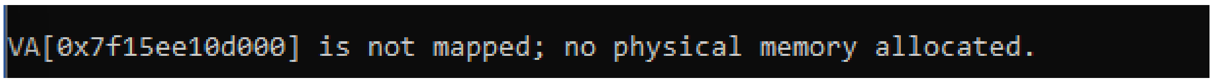
4.1.2 Step 2: Use mmap(2) to allocate a private buffer of size equal to 1 page and print the VM map again.
Παρατηρούμε ότι στον χάρτη που τυπώσαμε, στην γραμμή 10 έχει συγχωνευτεί μία σελίδα την οποία δεσμεύσαμε με την mmap. Το ΛΣ επέκτεινε την απεικόνιση αυτής της γραμμής κατά 1 σελίδα από την αρχή της επειδή σηματοδοτήσαμε την νέα απεικόνιση ως private για τη διεργασία αυτή.



```
Virtual Memory Map of process [14913]:
00400000-00403000 r-xp 00000000 fe:10 8550053                     /store/homes/oslab/oslaball/ask4/ask41/mmap
00602000-00603000 rw-p 00002000 fe:10 8550053                     /store/homes/oslab/oslaball/ask4/ask41/mmap
00a92000-00ab3000 rw-p 00000000 00:00 0                           [heap]
7f77eb538000-7f77eb6d9000 r-xp 00000000 fe:01 1045760             /lib/x86_64-linux-gnu/libc-2.19.so
7f77eb6d9000-7f77eb8d9000 ---p 001a1000 fe:01 1045760             /lib/x86_64-linux-gnu/libc-2.19.so
7f77eb8d9000-7f77eb8dd000 r--p 001a1000 fe:01 1045760             /lib/x86_64-linux-gnu/libc-2.19.so
7f77eb8dd000-7f77eb8df000 rw-p 001a5000 fe:01 1045760             /lib/x86_64-linux-gnu/libc-2.19.so
7f77eb8df000-7f77eb8e3000 rw-p 00000000 00:00 0
7f77eb8e3000-7f77eb903000 r-xp 00000000 fe:01 1044705             /lib/x86_64-linux-gnu/ld-2.19.so
7f77ebaf6000-7f77ebaf9000 rw-p 00000000 00:00 0
7f77ebafd000-7f77ebafe000 rw-p 00000000 00:00 0
7f77ebafe000-7f77ebaff000 rwxp 00000000 00:00 0
7f77ebaff000-7f77ebb03000 rw-p 00000000 00:00 0
7f77ebb03000-7f77ebb04000 r--p 00020000 fe:01 1044705             /lib/x86_64-linux-gnu/ld-2.19.so
7f77ebb04000-7f77ebb05000 rw-p 00021000 fe:01 1044705             /lib/x86_64-linux-gnu/ld-2.19.so
7f77ebb05000-7f77ebb06000 rw-p 00000000 00:00 0
7fff696d6000-7fff696f7000 rw-p 00000000 00:00 0                   [stack]
7fff696fb000-7fff696fd000 r-xp 00000000 00:00 0                   [vdso]
7fff696fd000-7fff696ff000 r--p 00000000 00:00 0                   [vvar]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0           [vsyscall]
--------------------------------------------------------

7f77ebafe000-7f77ebaff000 rwxp 00000000 00:00 0
```

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

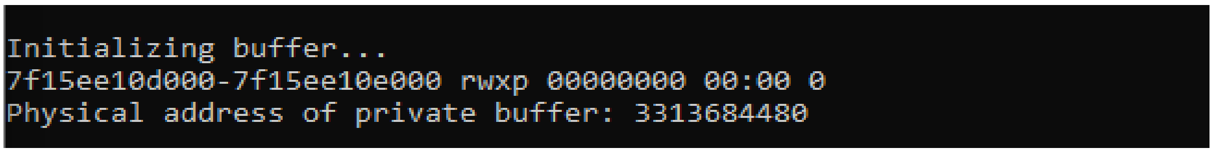
Αφού δεν αρχικοποιήσαμε τον buffer η δεσμευμένη εικονική μνήμη δεν αντιστοιχεί ακόμα σε κάποιο πεδίο διευθύνσεων στην πραγματική μνήμη. Αυτό συμβαίνει λόγω Demand Paging όταν μια διεργασία χρειαστεί να αποθηκεύσει δεδομένα σε μία σελίδα δημιουργείται το αντίστοιχο πλαίσιο στη φυσική μνήμη.



```
VA[0x7f15ee10d000] is not mapped; no physical memory allocated.
```

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

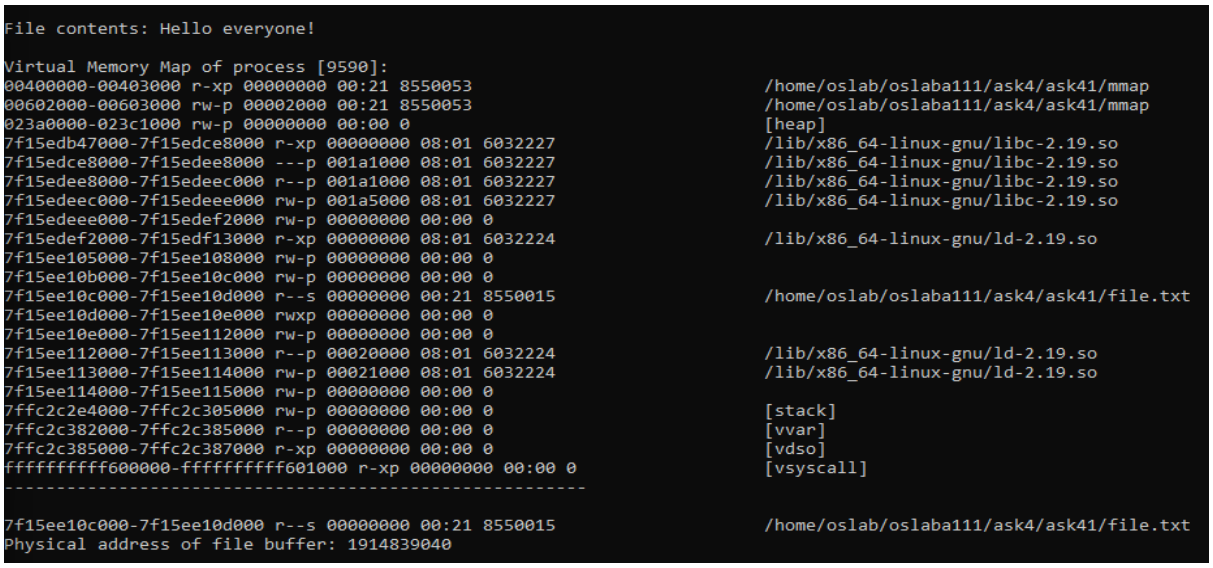
Από τη στιγμή που αρχικοποιήσαμε τον buffer παρατηρούμε ότι υπάρχει αντιστοίχιση της απεικόνισής μας σε πραγματική διεύθυνση φυσικής μνήμης.



```
Initializing buffer...
7f15ee10d000-7f15ee10e000 rwxp 00000000 00:00 0
Physical address of private buffer: 3313684480
```

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

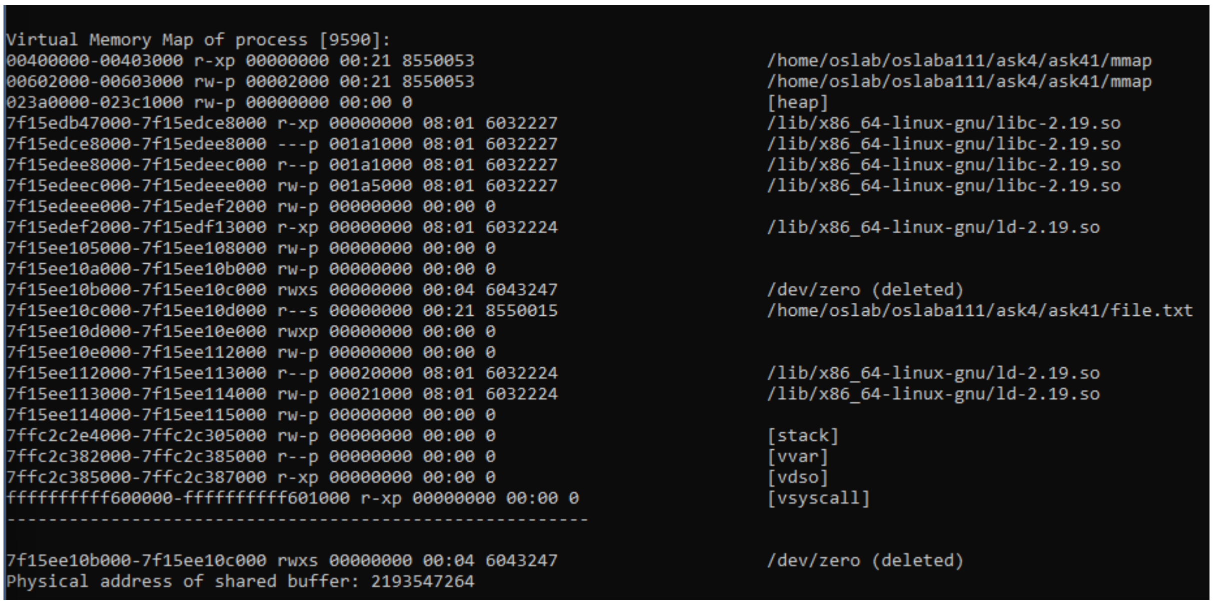
Αφού χρησιμοποιήσαμε την open() για να ανοίξουμε το αρχείο δημιουργείται ένα shared mapping μεγέθους όσο και το μέγεθος του αρχείου με χρήση των κατάλληλων flag για ανάγνωση αρχείου. Τα περιεχόμενα του buffer είναι και οι χαρακτήρες του αρχείου. Στον χάρτη απεικονίσεων παρατηρούμε ότι έχει δημιουργηθεί ένα νέο record για αυτή την απεικόνιση.



```
File contents: Hello everyone!

Virtual Memory Map of process [9590]:
00400000-00403000 r-xp 00000000 00:21 8550053                    /home/oslab/oslaba111/ask4/ask41/mmap
00602000-00603000 rw-p 00002000 00:21 8550053                    /home/oslab/oslaba111/ask4/ask41/mmap
023a0000-023c1000 rw-p 00000000 00:00 0                          [heap]
7f15edb47000-7f15edce8000 r-xp 00000000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f15edce8000-7f15edee8000 ---p 001a1000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f15edee8000-7f15edeec000 r--p 001a1000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f15edeec000-7f15edeee000 rw-p 001a5000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f15edeee000-7f15edef2000 rw-p 00000000 00:00 0
7f15edef2000-7f15edf13000 r-xp 00000000 08:01 6032224            /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee105000-7f15ee108000 rw-p 00000000 00:00 0
7f15ee10b000-7f15ee10c000 rw-p 00000000 00:00 0
7f15ee10c000-7f15ee10d000 r--s 00000000 00:21 8550015            /home/oslab/oslaba111/ask4/ask41/file.txt
7f15ee10d000-7f15ee10e000 rwxp 00000000 00:00 0
7f15ee10e000-7f15ee112000 rw-p 00000000 00:00 0
7f15ee112000-7f15ee113000 r--p 00020000 08:01 6032224            /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee113000-7f15ee114000 rw-p 00021000 08:01 6032224            /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee114000-7f15ee115000 rw-p 00000000 00:00 0
7ffc2c2e4000-7ffc2c305000 rw-p 00000000 00:00 0                  [stack]
7ffc2c382000-7ffc2c385000 r--p 00000000 00:00 0                  [vvar]
7ffc2c385000-7ffc2c387000 r-xp 00000000 00:00 0                  [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0          [vsyscall]
-------------------------------------------------------

7f15ee10c000-7f15ee10d000 r--s 00000000 00:21 8550015            /home/oslab/oslaba111/ask4/ask41/file.txt
Physical address of file buffer: 1914839040
```

4.1.6 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.

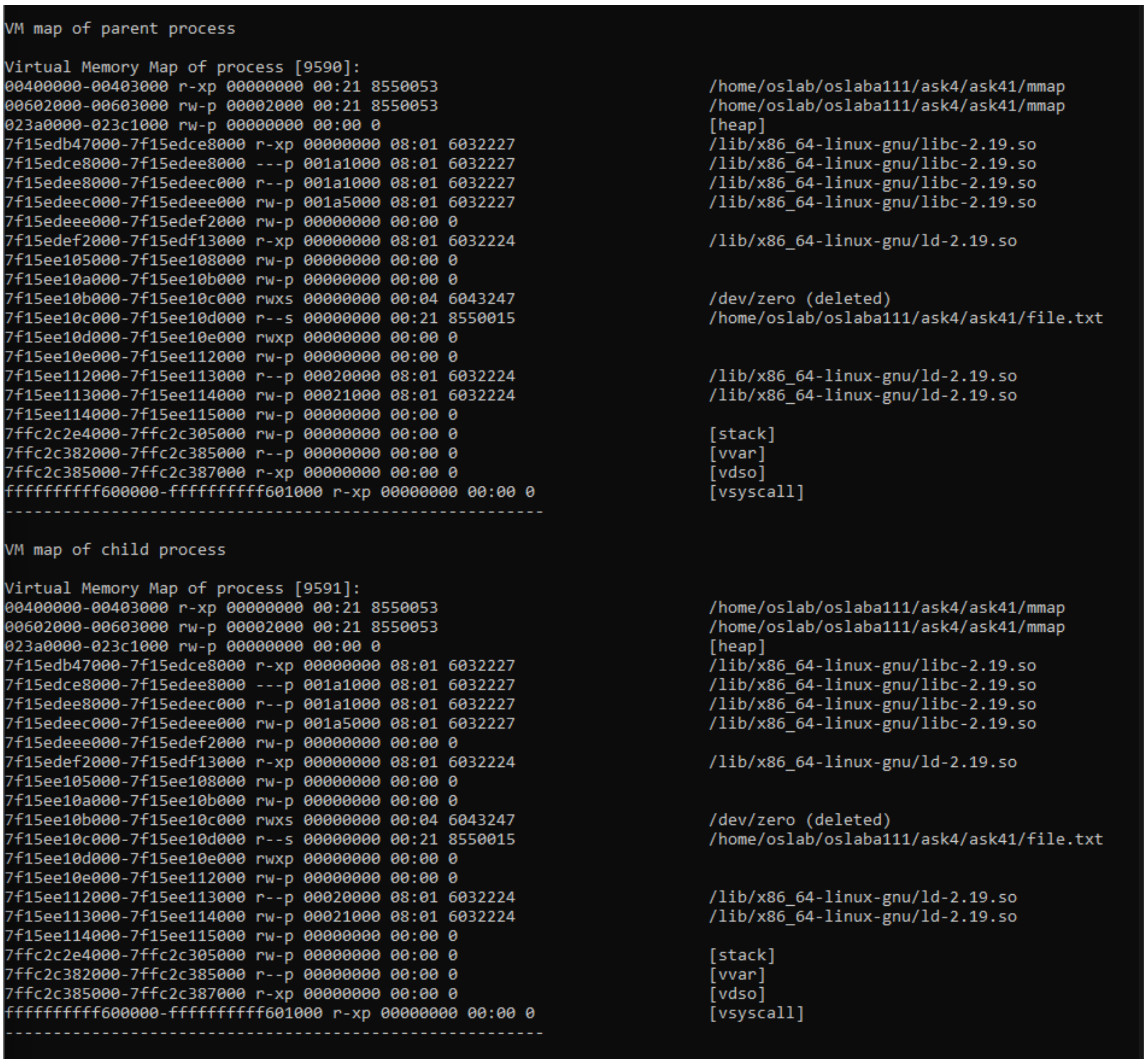
Το entry του νέου shared buffer που κατασκευάσαμε εμφανίζεται στον χάρτη απεικονίσεων αφού μετά την δημιουργία του ο buffer αρχικοποιείται.



```
Virtual Memory Map of process [9590]:
00400000-00403000 r-xp 00000000 00:21 8550053                    /home/oslab/oslaba111/ask4/ask41/mmap
00602000-00603000 rw-p 00002000 00:21 8550053                    /home/oslab/oslaba111/ask4/ask41/mmap
023a0000-023c1000 rw-p 00000000 00:00 0                          [heap]
7f15edb47000-7f15edce8000 r-xp 00000000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f15edce8000-7f15edee8000 ---p 001a1000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f15edee8000-7f15edeec000 r--p 001a1000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f15edeec000-7f15edeee000 rw-p 001a5000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f15edeee000-7f15edef2000 rw-p 00000000 00:00 0
7f15edef2000-7f15edf13000 r-xp 00000000 08:01 6032224            /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee105000-7f15ee108000 rw-p 00000000 00:00 0
7f15ee10a000-7f15ee10b000 rw-p 00000000 00:00 0
7f15ee10b000-7f15ee10c000 rwxs 00000000 00:04 6043247            /dev/zero (deleted)
7f15ee10c000-7f15ee10d000 r--s 00000000 00:21 8550015            /home/oslab/oslaba111/ask4/ask41/file.txt
7f15ee10d000-7f15ee10e000 rwxp 00000000 00:00 0
7f15ee10e000-7f15ee112000 rw-p 00000000 00:00 0
7f15ee112000-7f15ee113000 r--p 00020000 08:01 6032224            /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee113000-7f15ee114000 rw-p 00021000 08:01 6032224            /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee114000-7f15ee115000 rw-p 00000000 00:00 0
7ffc2c2e4000-7ffc2c305000 rw-p 00000000 00:00 0                  [stack]
7ffc2c382000-7ffc2c385000 r--p 00000000 00:00 0                  [vvar]
7ffc2c385000-7ffc2c387000 r-xp 00000000 00:00 0                  [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0          [vsyscall]
-------------------------------------------------------

7f15ee10b000-7f15ee10c000 rwxs 00000000 00:04 6043247            /dev/zero (deleted)
Physical address of shared buffer: 2193547264
```

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

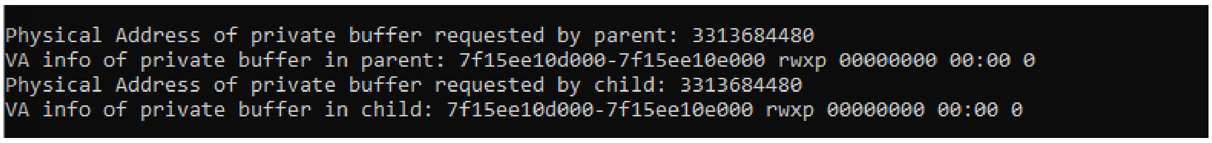
Δημιουργείται μια νέα διεργασία ,η διεργασία γονέας που εκτελεί την parent() και η διεργασία παιδί που εκτελεί την child(). Αφού η μια είναι αντίγραφο της άλλης, οι χάρτες απεικονίσεών είναι πανομοιότυποι.



```
VM map of parent process

Virtual Memory Map of process [9590]:
00400000-00403000 r-xp 00000000 00:21 8550053                            /home/oslab/oslaba111/ask4/ask41/mmap
00602000-00603000 rw-p 00002000 00:21 8550053                            /home/oslab/oslaba111/ask4/ask41/mmap
023a0000-023c1000 rw-p 00000000 00:00 0                                  [heap]
7f15edb47000-7f15edce8000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edce8000-7f15edee8000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edee8000-7f15edeec000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edeec000-7f15edeee000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edeee000-7f15edef2000 rw-p 00000000 00:00 0
7f15edef2000-7f15edf13000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee105000-7f15ee108000 rw-p 00000000 00:00 0
7f15ee10a000-7f15ee10b000 rw-p 00000000 00:00 0
7f15ee10b000-7f15ee10c000 rwxs 00000000 00:04 6043247                    /dev/zero (deleted)
7f15ee10c000-7f15ee10d000 r--s 00000000 00:21 8550015                    /home/oslab/oslaba111/ask4/ask41/file.txt
7f15ee10d000-7f15ee10e000 rwxp 00000000 00:00 0
7f15ee10e000-7f15ee112000 rw-p 00000000 00:00 0
7f15ee112000-7f15ee113000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee113000-7f15ee114000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee114000-7f15ee115000 rw-p 00000000 00:00 0
7ffc2c2e4000-7ffc2c305000 rw-p 00000000 00:00 0                          [stack]
7ffc2c382000-7ffc2c385000 r--p 00000000 00:00 0                          [vvar]
7ffc2c385000-7ffc2c387000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
--------------------------------------------------------

VM map of child process

Virtual Memory Map of process [9591]:
00400000-00403000 r-xp 00000000 00:21 8550053                            /home/oslab/oslaba111/ask4/ask41/mmap
00602000-00603000 rw-p 00002000 00:21 8550053                            /home/oslab/oslaba111/ask4/ask41/mmap
023a0000-023c1000 rw-p 00000000 00:00 0                                  [heap]
7f15edb47000-7f15edce8000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edce8000-7f15edee8000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edee8000-7f15edeec000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edeec000-7f15edeee000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edeee000-7f15edef2000 rw-p 00000000 00:00 0
7f15edef2000-7f15edf13000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee105000-7f15ee108000 rw-p 00000000 00:00 0
7f15ee10a000-7f15ee10b000 rw-p 00000000 00:00 0
7f15ee10b000-7f15ee10c000 rwxs 00000000 00:04 6043247                    /dev/zero (deleted)
7f15ee10c000-7f15ee10d000 r--s 00000000 00:21 8550015                    /home/oslab/oslaba111/ask4/ask41/file.txt
7f15ee10d000-7f15ee10e000 rwxp 00000000 00:00 0
7f15ee10e000-7f15ee112000 rw-p 00000000 00:00 0
7f15ee112000-7f15ee113000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee113000-7f15ee114000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee114000-7f15ee115000 rw-p 00000000 00:00 0
7ffc2c2e4000-7ffc2c305000 rw-p 00000000 00:00 0                          [stack]
7ffc2c382000-7ffc2c385000 r--p 00000000 00:00 0                          [vvar]
7ffc2c385000-7ffc2c387000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
--------------------------------------------------------
```

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

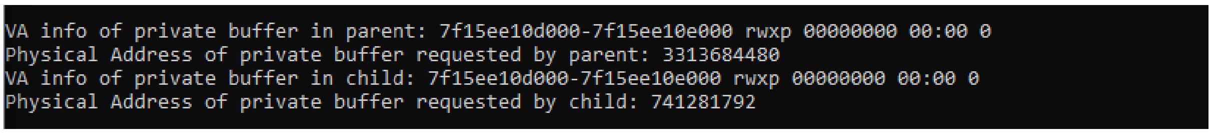
Αμέσως μετά το fork, η διεύθυνση στην οποία απεικονίζεται ο private buffer είναι η ίδια και στις δύο διεργασίες. Ακολουθείται Copy on Write, δηλαδή, ενώ αρχικά οι δείκτες δείχνουν στην ίδια διεύθυνση, όταν μία από τις δύο διεργασίες προσπαθήσει να γράψει στον private buffer,τότε το λειτουργικό αντιγράφει τα περιεχόμενα της μνήμης σε διαφορετικό πεδίο διευθύνσεων ώστε να αποφεύγονται οι άσκοπες αντιγραφές.



```
Physical Address of private buffer requested by parent: 3313684480
VA info of private buffer in parent: 7f15ee10d000-7f15ee10e000 rwxp 00000000 00:00 0
Physical Address of private buffer requested by child: 3313684480
VA info of private buffer in child: 7f15ee10d000-7f15ee10e000 rwxp 00000000 00:00 0
```

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

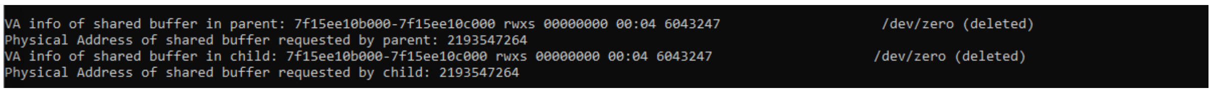
Αφού γράψουμε στον private buffer από το παιδί , η διεύθυνσή του μεταφέρεται σε άλλη διεύθυνση φυσικής μνήμης λόγω COW.



```
VA info of private buffer in parent: 7f15ee10d000-7f15ee10e000 rwxp 00000000 00:00 0
Physical Address of private buffer requested by parent: 3313684480
VA info of private buffer in child: 7f15ee10d000-7f15ee10e000 rwxp 00000000 00:00 0
Physical Address of private buffer requested by child: 741281792
```

4.1.10 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?

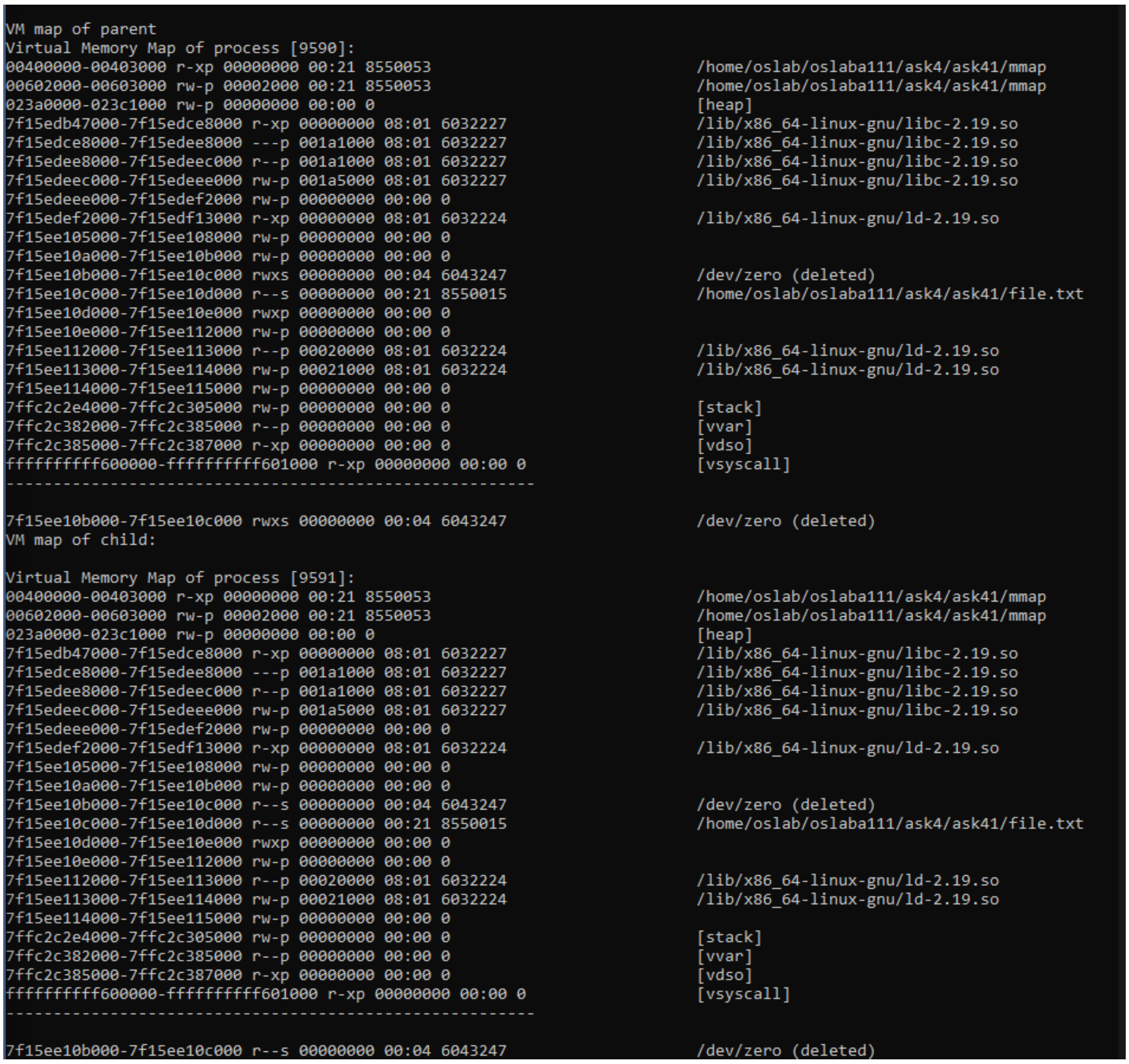
O shared buffer είναι στις ίδιες θέσεις μνήμης και για τις δύο διεργασίες.



```
VA info of shared buffer in parent: 7f15ee10b000-7f15ee10c000 rwxs 00000000 00:04 6043247                    /dev/zero (deleted)
Physical Address of shared buffer requested by parent: 2193547264
VA info of shared buffer in child: 7f15ee10b000-7f15ee10c000 rwxs 00000000 00:04 6043247                    /dev/zero (deleted)
Physical Address of shared buffer requested by child: 2193547264
```

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

Αλλάζοντας τα δικαιώματα του buffer στη διεργασία παιδί, παρατηρούμε ότι οι χάρτες των δύο διεργασιών διαφέρουν ως προς τα δικαιώματα αυτά. Δηλαδή, το παιδί έχει μόνο δικαίωμα ανάγνωσης (r) ενώ ο πατέρας συνεχίζει να έχει δικαιώματα ανάγνωσης και εγγραφής (rw).



```
VM map of parent
Virtual Memory Map of process [9590]:
00400000-00403000 r-xp 00000000 00:21 8550053                            /home/oslab/oslaba111/ask4/ask41/mmap
00602000-00603000 rw-p 00002000 00:21 8550053                            /home/oslab/oslaba111/ask4/ask41/mmap
023a0000-023c1000 rw-p 00000000 00:00 0                                  [heap]
7f15edb47000-7f15edce8000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edce8000-7f15edee8000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edee8000-7f15edeec000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edeec000-7f15edeee000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edeee000-7f15edef2000 rw-p 00000000 00:00 0
7f15edef2000-7f15edf13000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee105000-7f15ee108000 rw-p 00000000 00:00 0
7f15ee10a000-7f15ee10b000 rw-p 00000000 00:00 0
7f15ee10b000-7f15ee10c000 rwxs 00000000 00:04 6043247                    /dev/zero (deleted)
7f15ee10c000-7f15ee10d000 r--s 00000000 00:21 8550015                    /home/oslab/oslaba111/ask4/ask41/file.txt
7f15ee10d000-7f15ee10e000 rwxp 00000000 00:00 0
7f15ee10e000-7f15ee112000 rw-p 00000000 00:00 0
7f15ee112000-7f15ee113000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee113000-7f15ee114000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee114000-7f15ee115000 rw-p 00000000 00:00 0
7ffc2c2e4000-7ffc2c305000 rw-p 00000000 00:00 0                          [stack]
7ffc2c382000-7ffc2c385000 r--p 00000000 00:00 0                          [vvar]
7ffc2c385000-7ffc2c387000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
--------------------------------------------------------

7f15ee10b000-7f15ee10c000 rwxs 00000000 00:04 6043247                    /dev/zero (deleted)
VM map of child:

Virtual Memory Map of process [9591]:
00400000-00403000 r-xp 00000000 00:21 8550053                            /home/oslab/oslaba111/ask4/ask41/mmap
00602000-00603000 rw-p 00002000 00:21 8550053                            /home/oslab/oslaba111/ask4/ask41/mmap
023a0000-023c1000 rw-p 00000000 00:00 0                                  [heap]
7f15edb47000-7f15edce8000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edce8000-7f15edee8000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edee8000-7f15edeec000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edeec000-7f15edeee000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f15edeee000-7f15edef2000 rw-p 00000000 00:00 0
7f15edef2000-7f15edf13000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee105000-7f15ee108000 rw-p 00000000 00:00 0
7f15ee10a000-7f15ee10b000 rw-p 00000000 00:00 0
7f15ee10b000-7f15ee10c000 r--s 00000000 00:04 6043247                    /dev/zero (deleted)
7f15ee10c000-7f15ee10d000 r--s 00000000 00:21 8550015                    /home/oslab/oslaba111/ask4/ask41/file.txt
7f15ee10d000-7f15ee10e000 rwxp 00000000 00:00 0
7f15ee10e000-7f15ee112000 rw-p 00000000 00:00 0
7f15ee112000-7f15ee113000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee113000-7f15ee114000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f15ee114000-7f15ee115000 rw-p 00000000 00:00 0
7ffc2c2e4000-7ffc2c305000 rw-p 00000000 00:00 0                          [stack]
7ffc2c382000-7ffc2c385000 r--p 00000000 00:00 0                          [vvar]
7ffc2c385000-7ffc2c387000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
--------------------------------------------------------

7f15ee10b000-7f15ee10c000 r--s 00000000 00:04 6043247                    /dev/zero (deleted)
```

4.1.12 Τέλος, αποδεσμεύουμε τις απεικονίσεις και κλείνουμε το αρχείο “file.txt”.

Ο κώδικας για την πρώτη άσκηση :

```c
/*
 * mmap.c
 *
 * Examining the virtual memory of processes.
 *
 * Operating Systems course, CSLab, ECE, NTUA
 *
 */

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
```

```c
#include <sys/mman.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <stdint.h>
#include <signal.h>
#include <sys/wait.h>

#include "help.h"

#define RED     "\033[31m"
#define RESET   "\033[0m"

char *heap_private_buf;
char *heap_shared_buf;
char *file_shared_buf;

uint64_t buffer_size;

/*
 * Child process' entry point.
 */
void child(void)
{
//      uint64_t pa;

        /*
         * Step 7 - Child
         */
        if (0 != raise(SIGSTOP))
                die("raise(SIGSTOP)");
        /*
         * TODO: Write your code here to complete child's part of Step 7.
         */
        printf("VM map of child process\n");
        show_maps();

        /*
         * Step 8 - Child
         */
        if (0 != raise(SIGSTOP))
                die("raise(SIGSTOP)");
        /*
         * TODO: Write your code here to complete child's part of Step 8.
         */

        printf("Physical Address of private buffer requested by child: %ld\n", get_physical_address((uint64_t)heap_private_buf));
        printf("VA info of private buffer in child: ");
        show_va_info((uint64_t) heap_private_buf);

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

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

```

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

      for (j=0; j<(int) buffer_size; j++) {
              heap_shared_buf[j] = j;
      }
      printf("VA info of shared buffer in child: ");
      show_va_info((uint64_t) heap_shared_buf);

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

      /*
         * Step 11 - Child
         */

        //printf("Physical Address requested by child: %ld", get_physical_address(heap_shared_buf));

        if (0 != raise(SIGSTOP))
                die("raise(SIGSTOP)");

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

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

}

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

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

        /*
         * Step 7: Print parent's and child's maps. What do you see?
         * Step 7 - Parent
         */

        printf(RED "\nStep 7: Print parent's and child's map.\n" RESET);
        press_enter();

        printf("VM map of parent process\n");
```

```c
        show_maps();
        /*
         * 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.
         */

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

        if (-1 == kill(child_pid, SIGCONT))
                die("kill");
        if (-1 == waitpid(child_pid, &status, WUNTRACED))
                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("VA info of private buffer in parent: ");
        show_va_info((uint64_t) heap_private_buf);

        printf("Physical Address of private buffer requested by parent: %ld\n", get_physical_address((uint64_t)heap_private_buf));
        /*
         * 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("VA info of shared buffer in parent: ");
        show_va_info((uint64_t) heap_shared_buf);

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

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

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

        /*
         * TODO: Write your code here to complete parent's part of Step 11.
         */
        printf("VM map of parent");
        show_maps();
        show_va_info((uint64_t)heap_shared_buf);
        if (-1 == kill(child_pid, SIGCONT))
                die("kill");
        if (-1 == waitpid(child_pid, &status, 0))
                die("waitpid");

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

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

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.
         */
        //mmap ( NULL, sizeof(int),PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS, 1, 0 );
        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(heap_private_buf, buffer_size, PROT_READ | PROT_WRITE|
PROT_EXEC, MAP_PRIVATE | MAP_ANONYMOUS, -1,0);

          if (heap_private_buf == MAP_FAILED)
          {
                  perror("error 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);
          //show_va_info(heap_private_buf);
          //munmap(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.
           */
          //heap_zeros_buf = mmap(heap_zeros_buf, buffer_size,PROT_READ | PROT_WRITE,
MAP_PRIVATE| MAP_ANONYMOUS,-1,0);
          //show_va_info(heap_zeros_buf);
          //get_physical_address(heap_private_buf);
          int i;
          //heap_zeros_buf[0] = 0;
          printf("Initializing buffer...\n");
          for (i=0; i < (int) buffer_size; i++) {
                  *(heap_private_buf+i)= 0;
          }
          /*for (i=0; i <(int) buffer_size; i++) {
                  printf("%d", *(heap_private_buf+i));
          }*/

          show_va_info((uint64_t)heap_private_buf);
          printf("Physical address of private 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) {
                  perror("error opening file");
          }
          //show_maps();
          char c;
          file_shared_buf = mmap(NULL, buffer_size, PROT_READ, MAP_SHARED,fd,0);
```

```c

          if (file_shared_buf == MAP_FAILED)
          {
                  perror("error on step 5\n");
                  exit(-1);
          }

          printf("File contents: ");
          for (i=0; i<(int) buffer_size; i++){
                  c = *(file_shared_buf+i);
                  if (c !=  EOF)
                          putchar(c);
                  else
                          break;
          }
          //printf("\n");
          show_maps();
          show_va_info((uint64_t)file_shared_buf);

          printf("Physical address of file buffer: %ld\n",get_physical_address((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(heap_shared_buf, buffer_size, PROT_READ| PROT_WRITE| PROT_EXEC, MAP_SHARED| MAP_ANONYMOUS,-1,0);

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

          for(i=0; i<(int) buffer_size; i++){
                  *(heap_shared_buf+i)=0;
          }
          show_maps();
          show_va_info((uint64_t)heap_shared_buf);
          printf("Physical address of shared buffer: %ld\n",get_physical_address((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;
  }

```

ΑΣΚΗΣΗ 2^{Η}

4.2.1 Η υλοποίηση με νήματα περιμένουμε να έχει καλύτερη επίδοση από την υλοποίηση με διεργασίες.Με την δημιουργία μίας διεργασίας γίνεται αντιγραφή όλων των δεδομένων (program counter, registers, virtual memory κ.λπ.) και αυτό σε συνδυασμό με την mmap() κοστίζουν πολύ σε χρόνο. Αντίθετα, με νήματα δεν έχουμε αντιγραφή, το οποίο σε πιο σύνθετα προβλήματα θα είχε αισθητή διαφορά στην επίδοση παρόλα επειδή το πρόβλημα είναι απλό δεν παρατηρούμε μεγάλη αλλαγή στην επίδοση .

4.2.2 Αυτή τη φορά κατασκευάσαμε ένα shared buffer μεγέθους $x\_chars * y\_chars$ ο οποίο αποθηκεύει τη χρωματική τιμή κάθε χαρακτήρα. Αφού δημιουργηθούν οι διεργασίες παιδία, κάνουν τους υπολογισμούς των γραμμών και ανανεώνουν τον buffer και η διεργασία γονέας τυπώνει τον buffer χρησιμοποιώντας τη συνάρτηση output_mandel_line(). Ο συγχρονισμός εδώ επιτυγχάνεται απλά με τη χρήση πολλών wait() που είναι σε αριθμό όσες και οι διεργασίες παιδία που αναλαμβάνουν τον υπολογισμό.Εάν ο buffer μας είχε μέγεθος NPROCS * x_chars κάθε διεργασία μπορεί να αναλάβει NPROCS γραμμές. Με αυτό τον τρόπο, αφού μια διεργασία υπολογίσει και τυπώσει μία από τις γραμμές που της αντιστοιχούν θα κάνει raise(SIGSTOP) για να αναστείλει τον εαυτό της. Όταν όλες οι διεργασίες έχουν κάνει raise(SIGSTOP) τότε ο buffer θα γεμίσει με συνεχόμενες N γραμμές οι οποίες είναι έτοιμες για εκτύπωση. Η διεργασία γονιός περιμένει τα παιδιά της να σταματήσουν και μετά τυπώνει τα περιεχόμενα του buffer. Στη συνέχεια στέλνει σήμα SIGCONT στα παιδία ώστε να υπολογιστούν οι επόμενες N γραμμές. Αυτή η διαδικασία θα επαναληφθεί NPROCS / y_chars φορές.

Ο κώδικας για την άσκηση 2.1 :

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

#include <assert.h>
#include <math.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
```

```c
#include <sys/wait.h>
#include <unistd.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;

/*
 * 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;

sem_t *sem;

/*
 * 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;

```

```c
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, int N) {

  /*
   * A temporary array, used to hold color values for the line being drawn
   */
  int color_val[x_chars];
  compute_mandel_line(line, color_val);
  sem_wait(&sem[line % N]);
  output_mandel_line(fd, color_val);
  sem_post(&sem[(line + 1) % N]);
}

/*
 * 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, sysconf(_SC_PAGE_SIZE) * pages, PROT_READ | PROT_WRITE,
              MAP_ANONYMOUS | MAP_SHARED, -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);
  }
}

int main(int argc, char **argv) {
  int line, i, p, status;

  if (argc != 2) {
    perror("usage");
    exit(1);
  }

  int NPROCS = atoi(argv[1]);
  sem = (sem_t *)create_shared_memory_area(sizeof(sem_t) * NPROCS);

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

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

  for (i = 0; i < NPROCS; ++i)
    sem_init(&sem[i], 1, 0);
  sem_post(&sem[0]);

  for (i = 0; i < NPROCS; ++i) {
    p = fork();
    if (p == 0) {
      for (line = i; line < y_chars; line += NPROCS)
        compute_and_output_mandel_line(1, line, NPROCS);
      exit(0);
    }
  }

  for (i = 0; i < NPROCS; ++i)
    wait(&status);

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

```

Ο κώδικα για την άσκηση 2.2 :

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

#include <assert.h>
#include <math.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
```

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

#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;

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

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

/*
 * 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, int N) {
    /*
     * A temporary array, used to hold color values for the line being drawn
     */
    int j;
    int color_val[x_chars];
    compute_mandel_line(line, color_val);
    for (j = 0; j < x_chars; j++)
      buffer[x_chars * line + j] = color_val[j];
    // output_mandel_line(fd, color_val);
  }

  /*
   * 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, sysconf(_SC_PAGE_SIZE) * pages, PROT_READ | PROT_WRITE,
                MAP_ANONYMOUS | MAP_SHARED, -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
     */
```

```c
  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 line, i, p, status;

  if (argc != 2) {
    perror("usage");
    exit(1);
  }

  int NPROCS = atoi(argv[1]);
  buffer = create_shared_memory_area(sizeof(int) * x_chars * y_chars);

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

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

  for (i = 0; i < NPROCS; ++i) {
    p = fork();
    if (p < 0) {
      perror("Error:Fork");
      exit(-1);
    }
    if (p == 0) {
      for (line = i; line < y_chars; line += NPROCS)
        compute_and_output_mandel_line(1, line, NPROCS);
      exit(0);
    }
  }
  for (i = 0; i < NPROCS; ++i)
    wait(&status);

  for (i = 0; i < y_chars; ++i)
    output_mandel_line(1, buffer + x_chars * i);

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