ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Λειτουργικά Συστήματα Υπολογιστών

4ο Εργαστήριο

Ομάδα 36
Αναστάσιος Λαγός - el13531
Κωνσταντίνος Βασιλάκης - el16504

Ασκηση 1.1

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

Εκτελούμε τον κώδικα και έχουμε τα παρακάτω βήματα

1. Εδώ έχουμε τον χάρτη της εικονικής μνήμης της συγκεκριμένης διεργασίας

```
Step 1: Print the virtual address space map of this process [13347].


Virtual Memory Map of process [13347]:
00400000-00403000 r-xp 00000000 00:21 20455114                    /home/oslab/oslaba36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114                    /home/oslab/oslaba36/exercise4/exer4_1/mmap
015b7000-015d8000 rw-p 00000000 00:00 0                           [heap]
7f93517d8000-7f9351979000 r-xp 00000000 08:01 6032227             /lib/x86_64-linux-gnu/libc-2.19.so
7f9351979000-7f9351b79000 ---p 001a1000 08:01 6032227             /lib/x86_64-linux-gnu/libc-2.19.so
7f9351b79000-7f9351b7d000 r--p 001a1000 08:01 6032227             /lib/x86_64-linux-gnu/libc-2.19.so
7f9351b7d000-7f9351b7f000 rw-p 001a5000 08:01 6032227             /lib/x86_64-linux-gnu/libc-2.19.so
7f9351b7f000-7f9351b83000 rw-p 00000000 00:00 0
7f9351b83000-7f9351ba4000 r-xp 00000000 08:01 6032224             /lib/x86_64-linux-gnu/ld-2.19.so
7f9351d96000-7f9351d99000 rw-p 00000000 00:00 0
7f9351d9e000-7f9351da3000 rw-p 00000000 00:00 0
7f9351da3000-7f9351da4000 r--p 00020000 08:01 6032224             /lib/x86_64-linux-gnu/ld-2.19.so
7f9351da4000-7f9351da5000 rw-p 00021000 08:01 6032224             /lib/x86_64-linux-gnu/ld-2.19.so
7f9351da5000-7f9351da6000 rw-p 00000000 00:00 0
7fff0eab1000-7fff0ead2000 rw-p 00000000 00:00 0                   [stack]
7fff0eae7000-7fff0eaea000 r--p 00000000 00:00 0                   [vvar]
7fff0eaea000-7fff0eaec000 r-xp 00000000 00:00 0                   [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0           [vsyscall]
--------------------------------------------------------
```

2. Βλέπουμε παρακάτω τον νέο χάρτη της εικονικής μνήμης που έχει δημιουργηθεί η νέα σελίδα. Την ξεχωρίζουμε εύκολα γιατί σε αυτήν δώσαμε δικαιώματα read-/write/execute.

```
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 [13347]:
00400000-00403000 r-xp 00000000 00:21 20455114                    /home/oslab/oslaba36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114                    /home/oslab/oslaba36/exercise4/exer4_1/mmap
015b7000-015d8000 rw-p 00000000 00:00 0                           [heap]
7f93517d8000-7f9351979000 r-xp 00000000 08:01 6032227             /lib/x86_64-linux-gnu/libc-2.19.so
7f9351979000-7f9351b79000 ---p 001a1000 08:01 6032227             /lib/x86_64-linux-gnu/libc-2.19.so
7f9351b79000-7f9351b7d000 r--p 001a1000 08:01 6032227             /lib/x86_64-linux-gnu/libc-2.19.so
7f9351b7d000-7f9351b7f000 rw-p 001a5000 08:01 6032227             /lib/x86_64-linux-gnu/libc-2.19.so
7f9351b7f000-7f9351b83000 rw-p 00000000 00:00 0
7f9351b83000-7f9351ba4000 r-xp 00000000 08:01 6032224             /lib/x86_64-linux-gnu/ld-2.19.so
7f9351d96000-7f9351d99000 rw-p 00000000 00:00 0
7f9351d9d000-7f9351d9e000 rw-p 00000000 00:00 0
7f9351d9e000-7f9351d9f000 rwxp 00000000 00:00 0
7f9351d9f000-7f9351da3000 rw-p 00000000 00:00 0
7f9351da3000-7f9351da4000 r--p 00020000 08:01 6032224             /lib/x86_64-linux-gnu/ld-2.19.so
7f9351da4000-7f9351da5000 rw-p 00021000 08:01 6032224             /lib/x86_64-linux-gnu/ld-2.19.so
7f9351da5000-7f9351da6000 rw-p 00000000 00:00 0
7fff0eab1000-7fff0ead2000 rw-p 00000000 00:00 0                   [stack]
7fff0eae7000-7fff0eaea000 r--p 00000000 00:00 0                   [vvar]
7fff0eaea000-7fff0eaec000 r-xp 00000000 00:00 0                   [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0           [vsyscall]
--------------------------------------------------------
```

3. Παρατηρούμε ότι δεν υπάρχει φυσική διεύθυνση για αυτήν την σελίδα. Αυτό γίνεται επειδή στα Linux έχουμε demand paging δηλαδή μία εικονική διεύθυνση δεσμεύεται στην μνήμη μόνο όταν αυτό χρειαστεί (πχ κάνουμε κάποια εγγραφή).

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

VA[0x7f9351d9e000] is not mapped; no physical memory allocated.
```

4. Εδώ παρατηρούμε ότι μόλις κάνουμε αρχικοποίηση η σελίδα αυτή δεσμεύεται στην φυσική μνήμη.

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

0x4643131392
```

5. Παρακάτω φορτώνουμε στην μνήμη και τυπώνουμε το αρχειο. Μπορούμε εύκολα να ξεχωρίσουμε την σελίδα που δημιουργήθηκε στην εικονική μνήμη από το path του.

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


Virtual Memory Map of process [13347]:
00400000-00403000 r-xp 00000000 00:21 20455114                   /home/oslab/oslaba36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114                   /home/oslab/oslaba36/exercise4/exer4_1/mmap
015b7000-015d8000 rw-p 00000000 00:00 0                          [heap]
7f93517d8000-7f9351979000 r-xp 00000000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f9351979000-7f9351b79000 ---p 001a1000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f9351b79000-7f9351b7d000 r--p 001a1000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f9351b7d000-7f9351b7f000 rw-p 001a5000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f9351b7f000-7f9351b83000 rw-p 00000000 00:00 0
7f9351b83000-7f9351ba4000 r-xp 00000000 08:01 6032224            /lib/x86_64-linux-gnu/ld-2.19.so
7f9351d96000-7f9351d99000 rw-p 00000000 00:00 0
7f9351d9c000-7f9351d9d000 rw-p 00000000 00:00 0
7f9351d9d000-7f9351d9e000 r--p 00000000 00:21 20455008           /home/oslab/oslaba36/exercise4/exer4 1/file.txt
7f9351d9e000-7f9351d9f000 rwxp 00000000 00:00 0
7f9351d9f000-7f9351da3000 rw-p 00000000 00:00 0
7f9351da3000-7f9351da4000 r--p 00020000 08:01 6032224            /lib/x86_64-linux-gnu/ld-2.19.so
7f9351da4000-7f9351da5000 rw-p 00021000 08:01 6032224            /lib/x86_64-linux-gnu/ld-2.19.so
7f9351da5000-7f9351da6000 rw-p 00000000 00:00 0
7fff0eab1000-7fff0ead2000 rw-p 00000000 00:00 0                  [stack]
7fff0eae7000-7fff0eaea000 r--p 00000000 00:00 0                  [vvar]
7fff0eaea000-7fff0eaec000 r-xp 00000000 00:00 0                  [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0          [vsyscall]
--------------------------------------------------------

Hello everyone!
```

6. Η νέα σελίδα φαίνεται παρακάτω. Παρατηρούμε ότι στα permissions έχει s αντί για p στον τελευταίο χαρακτήρα που δείχνει ότι είναι shared.

```
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 [13358]:
00400000-00403000 r-xp 00000000 00:21 20455114                           /home/oslab/oslaba36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114                           /home/oslab/oslaba36/exercise4/exer4_1/mmap
025a7000-025c8000 rw-p 00000000 00:00 0                                  [heap]
7f762d801000-7f762d9a2000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f762d9a2000-7f762dba2000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba2000-7f762dba6000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba6000-7f762dba8000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba8000-7f762dbac000 rw-p 00000000 00:00 0
7f762dbac000-7f762dbcd000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddbf000-7f762ddc2000 rw-p 00000000 00:00 0
7f762ddc5000-7f762ddc6000 rw-p 00000000 00:00 0
7f762ddc6000-7f762ddc7000 rwxs 00000000 00:04 2313263                    /dev/zero (deleted)
7f762ddc7000-7f762ddc8000 rwxp 00000000 00:00 0
7f762ddc8000-7f762ddcc000 rw-p 00000000 00:00 0
7f762ddcc000-7f762ddcd000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddcd000-7f762ddce000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddce000-7f762ddcf000 rw-p 00000000 00:00 0
7fffdaab8000-7fffdaad9000 rw-p 00000000 00:00 0                          [stack]
7fffdab33000-7fffdab36000 r--p 00000000 00:00 0                          [vvar]
7fffdab36000-7fffdab38000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
---------------------------------------------------------
```

7. Εδώ οι δύο χάρτες μνήμης είναι ίδιοι. Ο λόγος είναι ότι όταν μία διεργασία κάνει fork τότε αντιγράφεται η μνήμη που καταλαμβάνει για την διαδικασία παιδί.

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


Virtual Memory Map of process [13358]:
00400000-00403000 r-xp 00000000 00:21 20455114                           /home/oslab/oslaba36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114                           /home/oslab/oslaba36/exercise4/exer4_1/mmap
025a7000-025c8000 rw-p 00000000 00:00 0                                  [heap]
7f762d801000-7f762d9a2000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f762d9a2000-7f762dba2000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba2000-7f762dba6000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba6000-7f762dba8000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba8000-7f762dbac000 rw-p 00000000 00:00 0
7f762dbac000-7f762dbcd000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddbf000-7f762ddc2000 rw-p 00000000 00:00 0
7f762ddc5000-7f762ddc6000 rw-p 00000000 00:00 0
7f762ddc6000-7f762ddc7000 rwxs 00000000 00:04 2313263                    /dev/zero (deleted)
7f762ddc7000-7f762ddc8000 rwxp 00000000 00:00 0
7f762ddc8000-7f762ddcc000 rw-p 00000000 00:00 0
7f762ddcc000-7f762ddcd000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddcd000-7f762ddce000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddce000-7f762ddcf000 rw-p 00000000 00:00 0
7fffdaab8000-7fffdaad9000 rw-p 00000000 00:00 0                          [stack]
7fffdab33000-7fffdab36000 r--p 00000000 00:00 0                          [vvar]
7fffdab36000-7fffdab38000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
---------------------------------------------------------


Virtual Memory Map of process [13359]:
00400000-00403000 r-xp 00000000 00:21 20455114                           /home/oslab/oslaba36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114                           /home/oslab/oslaba36/exercise4/exer4_1/mmap
025a7000-025c8000 rw-p 00000000 00:00 0                                  [heap]
7f762d801000-7f762d9a2000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f762d9a2000-7f762dba2000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba2000-7f762dba6000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba6000-7f762dba8000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba8000-7f762dbac000 rw-p 00000000 00:00 0
7f762dbac000-7f762dbcd000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddbf000-7f762ddc2000 rw-p 00000000 00:00 0
7f762ddc5000-7f762ddc6000 rw-p 00000000 00:00 0
7f762ddc6000-7f762ddc7000 rwxs 00000000 00:04 2313263                    /dev/zero (deleted)
7f762ddc7000-7f762ddc8000 rwxp 00000000 00:00 0
7f762ddc8000-7f762ddcc000 rw-p 00000000 00:00 0
7f762ddcc000-7f762ddcd000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddcd000-7f762ddce000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddce000-7f762ddcf000 rw-p 00000000 00:00 0
7fffdaab8000-7fffdaad9000 rw-p 00000000 00:00 0                          [stack]
7fffdab33000-7fffdab36000 r--p 00000000 00:00 0                          [vvar]
7fffdab36000-7fffdab38000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
---------------------------------------------------------
```

8. Παρατηρούμε ότι οι δύο διεργασίες δείχνουν στην ίδια θέση φυσικής μνήμης αν και η σελίδα είναι private. Αυτό γίνεται γιατί στο linux έχουμε την λογική copy on write δηλαδή παρόλο που οι δύο σελίδες είναι ξεχωριστές θα δημιουργηθεί καινούρια μόνο όταν αυτό χρειαστεί (πχ σε κάποιο modification).

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

Parent physical address of heap_private_buff: 1414520832
Child physical address of heap_private_buff: 1414520832
```

9. Εδώ παρατηρούμε αυτό που περιγράφηκε στο προηγούμενο βήμα ότι δηλαδή μόλις έγινε εγγραφή στον buffer του child process δεσμεύτηκε νέος χώρος φυσική μνήμης για αυτήν την σελίδα.

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

Parent physical address of heap_private_buff: 1414520832
Child physical address of heap_private_buff: 265641984
```

10. Σε αυτήν την περίπτωση επειδή η σελίδα είναι διαμοιραζόμενη δεν δημιουργείται καινούρια θέση μνήμης αφού οι δύο διεργασίες μοιράζονται από κοινού αυτήν την θέση.

```
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 physical address of heap_shared_buff: 203497472
Child physical address of heap_shared_buf: 203497472
```

11. Παρακάτω βλέπουμε ότι παρόλο που η σελίδα είναι διαμοιραζόμενη τα δικαιώματα της διεργασίας σε αυτήν την σελίδα βρίσκονται στον χάρτη μνήμης της διεργασίας που δεν μοιράζεται με τις υπόλοιπες οπότε αλλάξανε μόνο στην μία. Έτσι μπορούμε να έχουμε διαφορετικά δικαιώματα για διαφορετικές διεργασίες με διαμοιραζόμενες σελίδες.

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

Virtual Memory Map of process [13358]:
00400000-00403000 r-xp 00000000 00:21 20455114                          /home/oslab/oslaba36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114                          /home/oslab/oslaba36/exercise4/exer4_1/mmap
025a7000-025c8000 rw-p 00000000 00:00 0                                 [heap]
7f762d801000-7f762d9a2000 r-xp 00000000 08:01 6032227                   /lib/x86_64-linux-gnu/libc-2.19.so
7f762d9a2000-7f762dba2000 ---p 001a1000 08:01 6032227                   /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba2000-7f762dba6000 r--p 001a1000 08:01 6032227                   /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba6000-7f762dba8000 rw-p 001a5000 08:01 6032227                   /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba8000-7f762dbac000 rw-p 00000000 00:00 0
7f762dbac000-7f762dbcd000 r-xp 00000000 08:01 6032224                   /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddbf000-7f762ddc2000 rw-p 00000000 00:00 0
7f762ddc5000-7f762ddc6000 rw-p 00000000 00:00 0
7f762ddc6000-7f762ddc7000 rwxs 00000000 00:04 2313263                   /dev/zero (deleted)
7f762ddc7000-7f762ddc8000 rwxp 00000000 00:00 0
7f762ddc8000-7f762ddcc000 rw-p 00000000 00:00 0
7f762ddcc000-7f762ddcd000 r--p 00020000 08:01 6032224                   /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddcd000-7f762ddce000 rw-p 00021000 08:01 6032224                   /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddce000-7f762ddcf000 rw-p 00000000 00:00 0
7fffdaab8000-7fffdaad9000 rw-p 00000000 00:00 0                         [stack]
7fffdab33000-7fffdab36000 r--p 00000000 00:00 0                         [vvar]
7fffdab36000-7fffdab38000 r-xp 00000000 00:00 0                         [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                 [vsyscall]
--------------------------------------------------------


Virtual Memory Map of process [13359]:
00400000-00403000 r-xp 00000000 00:21 20455114                          /home/oslab/oslaba36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114                          /home/oslab/oslaba36/exercise4/exer4_1/mmap
025a7000-025c8000 rw-p 00000000 00:00 0                                 [heap]
7f762d801000-7f762d9a2000 r-xp 00000000 08:01 6032227                   /lib/x86_64-linux-gnu/libc-2.19.so
7f762d9a2000-7f762dba2000 ---p 001a1000 08:01 6032227                   /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba2000-7f762dba6000 r--p 001a1000 08:01 6032227                   /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba6000-7f762dba8000 rw-p 001a5000 08:01 6032227                   /lib/x86_64-linux-gnu/libc-2.19.so
7f762dba8000-7f762dbac000 rw-p 00000000 00:00 0
7f762dbac000-7f762dbcd000 r-xp 00000000 08:01 6032224                   /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddbf000-7f762ddc2000 rw-p 00000000 00:00 0
7f762ddc5000-7f762ddc6000 rw-p 00000000 00:00 0
7f762ddc6000-7f762ddc7000 r-xs 00000000 00:04 2313263                   /dev/zero (deleted)
7f762ddc7000-7f762ddc8000 rwxp 00000000 00:00 0
7f762ddc8000-7f762ddcc000 rw-p 00000000 00:00 0
7f762ddcc000-7f762ddcd000 r--p 00020000 08:01 6032224                   /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddcd000-7f762ddce000 rw-p 00021000 08:01 6032224                   /lib/x86_64-linux-gnu/ld-2.19.so
7f762ddce000-7f762ddcf000 rw-p 00000000 00:00 0
7fffdaab8000-7fffdaad9000 rw-p 00000000 00:00 0                         [stack]
7fffdab33000-7fffdab36000 r--p 00000000 00:00 0                         [vvar]
7fffdab36000-7fffdab38000 r-xp 00000000 00:00 0                         [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                 [vsyscall]
--------------------------------------------------------
```

Κώδικας Άσκησης

mmap.c

```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>
```

```c
#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;
  size_t pagesize = get_page_size();


  /*
   * Step 7 - Child
   */
  if (0 != raise(SIGSTOP))
    die("raise(SIGSTOP)");
  /*
   * TODO: Write your code here to complete child's part of Step 7.
   */
  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 physical address of heap_private_buff: %lu\n",
      get_physical_address(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.
	 */
	memset(heap_private_buf, 5, pagesize);
	printf("Child physical address of heap_private_buff: %lu\n", get_physical_address(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.
	 */
	memset(heap_shared_buf, 5, pagesize);
	printf("Child physical address of heap_shared_buf: %lu\n", get_physical_address(heap_shared_buf));


	/*
	 * Step 11 - Child
	 */
	if (0 != raise(SIGSTOP))
		die("raise(SIGSTOP)");
	/*
	 * TODO: Write your code here to complete child's part of Step 11.
	 */
	mprotect(heap_shared_buf, pagesize, PROT_READ | PROT_EXEC);
	show_maps();


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

}

	/*
 * Parent process' entry point.
```

```c
 */
void parent(pid_t child_pid)
{
  uint64_t pa;
  int status;
  size_t pagesize = get_page_size();

  /* 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.
   */
  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 physical address of heap_private_buff: %lu\n", get_physical_address(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();

/*
 * TODO: Write your code here to complete parent's part of Step 9.
 */
printf("Parent physical address of heap_private_buff: %lu\n", get_physical_address(heap_private_buf));


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


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

/*
 * TODO: Write your code here to complete parent's part of Step 10.
 */
printf("Parent physical address of heap_shared_buff: %lu\n", get_physical_address(heap_shared_buf));


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


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

/*
```

```
	 * TODO: Write your code here to complete parent's part of Step 11.
	 */
	show_maps();

	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, pagesize);
	munmap(heap_private_buf, pagesize);
}

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.
 */
size_t pagesize = get_page_size();
heap_private_buf = mmap(NULL, pagesize, PROT_EXEC | PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS
, -1, 0);
show_maps();


/*
 * 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(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, pagesize);
pa = heap_private_buf;
printf("0x%lu\n",get_physical_address(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);
off_t file_size;
file_size = lseek(fd, 0, SEEK_END);
file_shared_buf = mmap(NULL, file_size ,PROT_READ , MAP_PRIVATE, fd, 0);
```

```
  show_maps();
  //close(fd);
  ssize_t n = write(1, file_shared_buf, file_size);
  if (n == -1) {
    printf("Write error!\n");
  }
  munmap(file_shared_buf, file_size);
  printf("\n");


  /*
   * 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, pagesize, PROT_EXEC | PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
  memset(heap_shared_buf, 0, pagesize);
  pa = heap_shared_buf;

  show_maps();
  printf("0x%lu\n", get_physical_address(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;
}
```

Makefile

```
.PHONY: all clean

all: mmap

CC = gcc
CFLAGS = -g -Wall -Wextra -O2
```

```
SHELL= /bin/bash

mmap: mmap.o help.o
	$(CC) $(CFLAGS) $^ -o $@

%.s: %.c
	$(CC) $(CFLAGS) -S -fverbose-asm $<

%.o: %.c
	$(CC) $(CFLAGS) -c $<

%.i: %.c
	gcc -Wall -E $< | indent -kr > $@

clean:
	rm -f *.o mmap
```

Ασκηση 1.2.1

Semaphores πάνω από διαμοιραζόμενη μνήμη

1. Η επίδοση των threads θα πρέπει να είναι ελαφρώς καλύτερη από άποψη χρόνου (ανάλογα με το πόσα φτιάχνουμε τόσο χειροτερεύει) επειδή για την αρχικοποίησης μίας διεργασίας χρειάζονται περισσότερα resources αφού πρέπει να αντιγράψουμε την μνήμη του process για κάθε διεργασία παιδί. Για τον ίδιο λόγο ο χώρος στην ram θα είναι πολύ περισσότερος για τα processes. Για τα threads δεν έχουμε τα παραπάνω προβλήματα αφού χρησιμοποιούν την μνήμη της διεργασίας που δημιουργούνται. Γενικότερα προτιμάμε processes σε πιο βαριές διαδικασίες που υπάρχει και ανάγκη για isolation.

Κώδικας Άσκησης

mandel-sem.c

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

#include <stdio.h>
#include <unistd.h>
```

```c
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <semaphore.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;


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

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

  l = strtol(s, &endp, 10);
  if (s != endp && *endp == '\0') {
    *val = l;
```

```
    return 0;
  } else
    return -1;
}

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

  int n;
  int val;

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

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

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

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

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

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

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

```

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

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


  //No synchronization needed for the calculation
  compute_mandel_line(line, color_val);

  //Synchronization is added when writing to the output
  //Get the previous semaphore's position (if we are at position 0 get the last semaphore)
  int previousSemPosition = (processNumber + processCount - 1) % processCount;

  //Wait if previous process has not finished
  sem_wait(&sems[previousSemPosition]);
  output_mandel_line(fd, color_val);
  //When finished signal to the next process
  sem_post(&sems[processNumber]);
}


/*
 * Function that is run by the child processes
 */
void process_fn(int processNumber, int processCount, sem_t* sems) {
  int line;
  for (line = processNumber; line < y_chars; line += processCount) {
    compute_and_output_mandel_line(1, line, processNumber, processCount, sems);
  }

  return;
}

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

```
        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;
        totalPageBytes = pages * sysconf(_SC_PAGE_SIZE);

        /* Create a shared, anonymous mapping for this number of pages */
        addr = mmap(NULL, totalPageBytes, 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);
        }
}

int main(int argc, char *argv[])
{
  if (argc != 2)
    usage(argv[0]);


    int processCount;

```

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

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

  //Create shared memory for semaphore
  sem_t* sems = create_shared_memory_area(processCount*sizeof(sem_t*));

  //Initialize semaphores
  int i;
  for (i = 0; i < processCount; i++) {
    sem_init(&sems[i],1,((i==processCount-1)?1:0));
  }

  pid_t pid;
  int status;

  //Create child processes
  for (i = 0; i < processCount; i++) {
    //fork from main to create child process
    pid = fork();
    if (pid < 0) {
      perror("Fork error");
      exit(1);
    } else if (pid == 0) {
      //Child process enters here
      process_fn(i, processCount, sems);
      exit(0);
    }
  }

  //Wait for all the children to finish
  while ((pid = wait(&status) > 0));

  //Clear shared memory
  destroy_shared_memory_area(sems, processCount*sizeof(sem_t*));

  reset_xterm_color(1);

  return 0;
}
```

Ασκηση 1.2.2

Υλοποίηση χωρίς semaphores

1. Εδώ δεν χρειαζόμαστε συγχρονισμό αφού υπολογίζουμε πρώτα όλες τις γραμμές στον buffer οι οποίες είναι ανεξάρτητες μεταξύ τους και μετά τις τυπώνουμε με την σειρά. Αν είχαμε έναν μικρότερο buffer θα μπορούσαμε να βάλουμε πολλούς τέτοιους buffers αν γίνεται αλλιώς για να χρησιμοποιήσουμε μόνο αυτόν χρειαζόμαστε κάποιο σχήμα συγχρονισμού. Θα μπορούσαμε πχ να χρησιμοποιήσουμε ένα multiple producer - one consumer σχήμα όπου έχουμε τις διεργασίες παιδιά να μπορούν να μπουν μόνο μία φορά για να υπολογίσουν από μία γραμμή για κάθε επανάληψη. Μετά να δίνεται η δυνατότητα στο parent process να τυπώσει τον buffer πριν επιτραπεί στις διεργασίες παιδιά να υπολογίσουν τις επόμενες γραμμές.

Κώδικας Άσκησης

mandel-buffer.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 <sys/types.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <semaphore.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;


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

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

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

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

  int n;
  int val;

```

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


  compute_mandel_line(line, color_val);

```

```
  int i;

  for (i=0;i<x_chars;i++) {
    buffer[line][i] = color_val[i];
  }
}


void process_fn(int processNumber, int processCount, int** buffer) {
  int line;
  for (line = processNumber; line < y_chars; line += processCount) {
    compute_and_store_mandel_line(1, line, processNumber, processCount, buffer);
  }

  return;
}

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

        /* Create a shared, anonymous mapping for this number of pages */
        addr = mmap(NULL, totalPageBytes, 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);
        }
}

int main(int argc, char *argv[])
{
  if (argc != 2)
    usage(argv[0]);


    int processCount;

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

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


  //Initialize buffer in shared memory
  int** buffer = (int**) create_shared_memory_area(x_chars*sizeof(int*));

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

  pid_t pid;
  int status;

  for (i = 0; i < processCount; i++) {
    pid = fork();   //fork from main to create child process
    if (pid < 0) {
      perror("Fork error");
      exit(1);
```

```
    } else if (pid == 0) {
      process_fn(i, processCount, buffer);
      exit(0);
    }
  }

  while ((pid = wait(&status) > 0));

  //Outputs mandel line
  //We loop through all the lines of the buffer and print them one by one
  int line;
  for (line=0;line<y_chars;line++) {
    int color_val[x_chars];
    for (i=0;i<x_chars;i++) {
      color_val[i] = buffer[line][i];
    }

    output_mandel_line(1, color_val);
  }

  //Clear shared memory
  for (i=0;i<x_chars;i++) {
    destroy_shared_memory_area(buffer[i], y_chars*sizeof(int));
  }
  destroy_shared_memory_area(buffer, x_chars*sizeof(int));

  reset_xterm_color(1);

  return 0;
}
```

Makefile

```
#
# Makefile
#

CC = gcc

CFLAGS = -Wall -O2 -pthread
LIBS =

all: mandel-sem mandel-buffer


## Mandel

mandel-sem: mandel-lib.o mandel-sem.o
	$(CC) $(CFLAGS) -o mandel-sem mandel-lib.o mandel-sem.o $(LIBS)

mandel-buffer: mandel-lib.o mandel-buffer.o
	$(CC) $(CFLAGS) -o mandel-buffer mandel-lib.o mandel-buffer.o $(LIBS)
```

```
mandel-lib.o: mandel-lib.h mandel-lib.c
	$(CC) $(CFLAGS) -c -o mandel-lib.o mandel-lib.c $(LIBS)

mandel-sem.o: mandel-sem.c
	$(CC) $(CFLAGS) -c -o mandel-sem.o mandel-sem.c $(LIBS)

mandel-buffer.o: mandel-buffer.c
	$(CC) $(CFLAGS) -c -o mandel-buffer.o mandel-buffer.c $(LIBS)

clean:
	rm -f *.o mandel-sem mandel-buffer
```

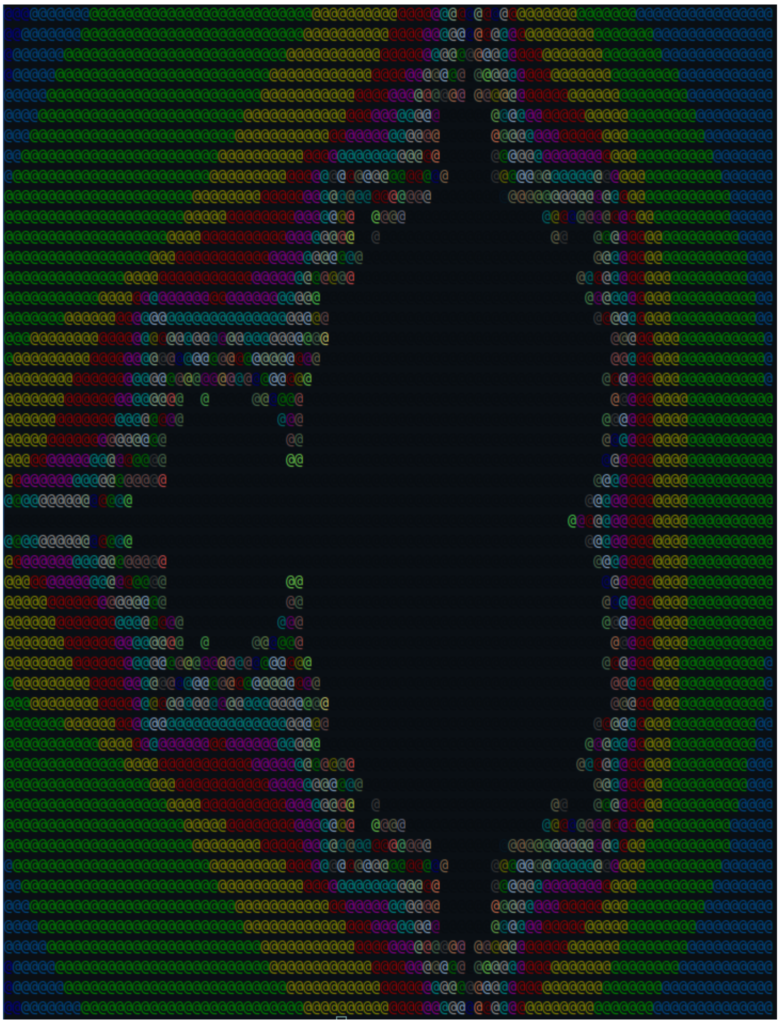

Figure 1: Output του mandel-buffer. Προφανώς το ίδιο και για mandel-sem