

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/oslab36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114 /home/oslab/oslab36/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]
fffffffff600000-fffffffff601000 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/oslab36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114 /home/oslab/oslab36/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]
fffffffff600000-fffffffff601000 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 αρχείο. 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/oslab36/exercise4/exer4_1/mmap  
00602000-00603000 rw-p 00002000 00:21 20455114 /home/oslab/oslab36/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/oslab36/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]  
fffffffff600000-fffffffff601000 r-xp 00000000 00:00 0 [vsyscall]  
-----  
Hello everyone!
```

6. Η νέα σελίδα φαίνεται παρακάτω. Παρατηρούμε ότι στα permissions έχει s αντί για r 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/oslab36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114 /home/oslab/oslab36/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
7f762dbac000-7f762dbc000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f762dbf000-7f762ddc2000 rw-p 00000000 00:00 0
7f762ddc5000-7f762ddc6000 rw-p 00000000 00:00 0
7f762ddc6000-7f762ddc7000 rwxp 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
7fffdab8000-7fffdab9000 rw-p 00000000 00:00 0 [stack]
7fffdab3000-7fffdab36000 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/oslab36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114 /home/oslab/oslab36/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
7f762dbac000-7f762dbc000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f762dbf000-7f762ddc2000 rw-p 00000000 00:00 0
7f762ddc5000-7f762ddc6000 rw-p 00000000 00:00 0
7f762ddc6000-7f762ddc7000 rwxp 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
7fffdab8000-7fffdab9000 rw-p 00000000 00:00 0 [stack]
7fffdab3000-7fffdab36000 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/oslab36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114 /home/oslab/oslab36/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
7f762dbac000-7f762dbc000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f762dbf000-7f762ddc2000 rw-p 00000000 00:00 0
7f762ddc5000-7f762ddc6000 rw-p 00000000 00:00 0
7f762ddc6000-7f762ddc7000 rwxp 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
7fffdab8000-7fffdab9000 rw-p 00000000 00:00 0 [stack]
7fffdab3000-7fffdab36000 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 III: 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/oslab36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114 /home/oslab/oslab36/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
7f762dbcd000-7f762ddc2000 rw-p 00000000 00:00 0
7f762ddc2000-7f762ddc6000 rw-p 00000000 00:00 0
7f762ddc6000-7f762ddc7000 rwxp 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
7ffffdaab8000-7ffffdaad9000 rw-p 00000000 00:00 0 [stack]
7ffffdab33000-7ffffdab36000 r--p 00000000 00:00 0 [vvar]
7ffffdab36000-7ffffdab38000 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/oslab36/exercise4/exer4_1/mmap
00602000-00603000 rw-p 00002000 00:21 20455114 /home/oslab/oslab36/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
7f762dbcd000-7f762ddc2000 rw-p 00000000 00:00 0
7f762ddc2000-7f762ddc6000 rw-p 00000000 00:00 0
7f762ddc6000-7f762ddc7000 r-xp 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
7ffffdaab8000-7ffffdaad9000 rw-p 00000000 00:00 0 [stack]
7ffffdab33000-7ffffdab36000 r--p 00000000 00:00 0 [vvar]
7ffffdab36000-7ffffdab38000 r-xp 00000000 00:00 0 [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
-----
```

## Κώδικας Άσκησης

### mmap.c

```
1  /*
2   * mmap.c
3   *
4   * Examining the virtual memory of processes.
5   *
6   * Operating Systems course, CSLab, ECE, NTUA
7   *
8   */
9
10
11 #include <stdlib.h>
12 #include <string.h>
13 #include <stdio.h>
14 #include <sys/mman.h>
15 #include <unistd.h>
```

```

16 #include <sys/types.h>
17 #include <sys/stat.h>
18 #include <fcntl.h>
19 #include <errno.h>
20 #include <stdint.h>
21 #include <signal.h>
22 #include <sys/wait.h>
23
24 #include "help.h"
25
26 #define RED      "\033[31m"
27 #define RESET    "\033[0m"
28
29
30 char *heap_private_buf;
31 char *heap_shared_buf;
32
33 char *file_shared_buf;
34
35 uint64_t buffer_size;
36
37
38
39
40 /*
41  * Child process' entry point.
42  */
43 void child(void)
44 {
45     uint64_t pa;
46     size_t pagesize = get_page_size();
47
48
49     /*
50      * Step 7 - Child
51      */
52     if (0 != raise(SIGSTOP))
53         die("raise(SIGSTOP)");
54     /*
55      * TODO: Write your code here to complete child's part of Step 7.
56      */
57     show_maps();
58
59
60     /*
61      * Step 8 - Child
62      */
63     if (0 != raise(SIGSTOP))
64         die("raise(SIGSTOP)");
65     /*
66      * TODO: Write your code here to complete child's part of Step 8.
67      */
68     printf("Child physical address of heap_private_buf: %lu\n",
69           get_physical_address(heap_private_buf));
70

```



```

71
72  /*
73   * Step 9 - Child
74   */
75  if (0 != raise(SIGSTOP))
76      die("raise(SIGSTOP)");
77  /*
78   * TODO: Write your code here to complete child's part of Step 9.
79   */
80  memset(heap_private_buf, 5, pagesize);
81  printf("Child physical address of heap_private_buf: %lu\n",
82         get_physical_address(heap_private_buf));
83
84
85  /*
86   * Step 10 - Child
87   */
88  if (0 != raise(SIGSTOP))
89      die("raise(SIGSTOP)");
90  /*
91   * TODO: Write your code here to complete child's part of Step
92   *      10.
93   */
94  memset(heap_shared_buf, 5, pagesize);
95  printf("Child physical address of heap_shared_buf: %lu\n",
96         get_physical_address(heap_shared_buf));
97
98  /*
99   * Step 11 - Child
100   */
101  if (0 != raise(SIGSTOP))
102      die("raise(SIGSTOP)");
103  /*
104   * TODO: Write your code here to complete child's part of Step
105   *      11.
106   */
107  mprotect(heap_shared_buf, pagesize, PROT_READ | PROT_EXEC);
108  show_maps();
109
110  /*
111   * Step 12 - Child
112   */
113  /*
114   * TODO: Write your code here to complete child's part of Step
115   *      12.
116   */
117  munmap(heap_shared_buf, pagesize);
118  munmap(heap_private_buf, pagesize);
119
120  }
121
122  /*
123   * Parent process' entry point.

```



```

122  */
123 void parent(pid_t child_pid)
124 {
125     uint64_t pa;
126     int status;
127     size_t pagesize = get_page_size();
128
129     /* Wait for the child to raise its first SIGSTOP. */
130     if (-1 == waitpid(child_pid, &status, WUNTRACED))
131         die("waitpid");
132
133     /*
134      * Step 7: Print parent's and child's maps. What do you see?
135      * Step 7 - Parent
136      */
137     printf(RED "\nStep 7: Print parent's and child's map.\n" RESET);
138     press_enter();
139
140     /*
141      * TODO: Write your code here to complete parent's part of Step
142      * 7.
143      */
144     show_maps();
145
146     if (-1 == kill(child_pid, SIGCONT))
147         die("kill");
148     if (-1 == waitpid(child_pid, &status, WUNTRACED))
149         die("waitpid");
150
151     /*
152      * Step 8: Get the physical memory address for heap_private_buf.
153      * Step 8 - Parent
154      */
155     printf(RED "\nStep 8: Find the physical address of the private
156             heap "
157             "buffer (main) for both the parent and the child.\n" RESET);
158     press_enter();
159
160     /*
161      * TODO: Write your code here to complete parent's part of Step
162      * 8.
163      */
164     printf("Parent physical address of heap_private_buff: %lu\n",
165            get_physical_address(heap_private_buf));
166
167     if (-1 == kill(child_pid, SIGCONT))
168         die("kill");
169     if (-1 == waitpid(child_pid, &status, WUNTRACED))
170         die("waitpid");
171
172     /*
173      * Step 9: Write to heap_private_buf. What happened?
174      * Step 9 - Parent

```

```

174     */
175     printf(RED "\nStep 9: Write to the private buffer from the child
176         and "
177         "repeat step 8. What happened?\n" RESET);
178     press_enter();
179     /*
180     * TODO: Write your code here to complete parent's part of Step
181     * 9.
182     */
183     printf("Parent physical address of heap_private_buff: %lu\n",
184         get_physical_address(heap_private_buf));
185
186     if (-1 == kill(child_pid, SIGCONT))
187         die("kill");
188     if (-1 == waitpid(child_pid, &status, WUNTRACED))
189         die("waitpid");
190
191     /*
192     * Step 10: Get the physical memory address for heap_shared_buf.
193     * Step 10 - Parent
194     */
195     printf(RED "\nStep 10: Write to the shared heap buffer (main)
196         from "
197         "child and get the physical address for both the parent and "
198         "the child. What happened?\n" RESET);
199     press_enter();
200     /*
201     * TODO: Write your code here to complete parent's part of Step
202     * 10.
203     */
204     printf("Parent physical address of heap_shared_buff: %lu\n",
205         get_physical_address(heap_shared_buf));
206
207     if (-1 == kill(child_pid, SIGCONT))
208         die("kill");
209     if (-1 == waitpid(child_pid, &status, WUNTRACED))
210         die("waitpid");
211
212     /*
213     * Step 11: Disable writing on the shared buffer for the child
214     * (hint: mprotect(2)).
215     * Step 11 - Parent
216     */
217     printf(RED "\nStep 11: Disable writing on the shared buffer for
218         the "
219         "child. Verify through the maps for the parent and the "
220         "child.\n" RESET);
221     press_enter();
222     /*

```

```

223     * TODO: Write your code here to complete parent's part of Step
224     11.
225     */
226     show_maps();
227
228     if (-1 == kill(child_pid, SIGCONT))
229         die("kill");
230     if (-1 == waitpid(child_pid, &status, 0))
231         die("waitpid");
232
233     /*
234     * Step 12: Free all buffers for parent and child.
235     * Step 12 - Parent
236     */
237
238     /*
239     * TODO: Write your code here to complete parent's part of Step
240     12.
241     */
242     munmap(heap_shared_buf, pagesize);
243     munmap(heap_private_buf, pagesize);
244 }
245
246 int main(void)
247 {
248     pid_t mypid, p;
249     int fd = -1;
250     uint64_t pa;
251
252     mypid = getpid();
253     buffer_size = 1 * get_page_size();
254
255     /*
256     * Step 1: Print the virtual address space layout of this process
257     */
258     printf(RED "\nStep 1: Print the virtual address space map of this
259     "
260     "process [%d].\n" RESET, mypid);
261     press_enter();
262     /*
263     * TODO: Write your code here to complete Step 1.
264     */
265     show_maps();
266
267     /*
268     * Step 2: Use mmap to allocate a buffer of 1 page and print the
269     map
270     * again. Store buffer in heap_private_buf.
271     */
272     printf(RED "\nStep 2: Use mmap(2) to allocate a private buffer of
273     "
274     "size equal to 1 page and print the VM map again.\n" RESET);
275     press_enter();

```

```

273  /*
274  * TODO: Write your code here to complete Step 2.
275  */
276  size_t pagesize = get_page_size();
277  heap_private_buf = mmap(NULL, pagesize, PROT_EXEC | PROT_READ |
    PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS
278  , -1, 0);
279  show_maps();
280
281
282  /*
283  * Step 3: Find the physical address of the first page of your
    buffer
284  * in main memory. What do you see?
285  */
286  printf(RED "\nStep 3: Find and print the physical address of the
    "
287  "buffer in main memory. What do you see?\n" RESET);
288  press_enter();
289  /*
290  * TODO: Write your code here to complete Step 3.
291  */
292  get_physical_address(heap_private_buf);
293
294
295  /*
296  * Step 4: Write zeros to the buffer and repeat Step 3.
297  */
298  printf(RED "\nStep 4: Initialize your buffer with zeros and
    repeat "
299  "Step 3. What happened?\n" RESET);
300  press_enter();
301  /*
302  * TODO: Write your code here to complete Step 4.
303  */
304  memset(heap_private_buf, 0, pagesize);
305  pa = heap_private_buf;
306  printf("0x%lu\n", get_physical_address(pa));
307
308  /*
309  * Step 5: Use mmap(2) to map file.txt (memory-mapped files) and
    print
310  * its content. Use file_shared_buf.
311  */
312  printf(RED "\nStep 5: Use mmap(2) to read and print file.txt.
    Print "
313  "the new mapping information that has been created.\n" RESET);
314  press_enter();
315  /*
316  * TODO: Write your code here to complete Step 5.
317  */
318  fd = open("./file.txt", O_RDONLY);
319  off_t file_size;
320  file_size = lseek(fd, 0, SEEK_END);
321  file_shared_buf = mmap(NULL, file_size, PROT_READ, MAP_PRIVATE,
    fd, 0);

```

```

322     show_maps();
323     //close(fd);
324     ssize_t n = write(1, file_shared_buf, file_size);
325     if (n == -1) {
326         printf("Write error!\n");
327     }
328     munmap(file_shared_buf, file_size);
329     printf("\n");
330
331
332     /*
333     * Step 6: Use mmap(2) to allocate a shared buffer of 1 page. Use
334     * heap_shared_buf.
335     */
336     printf(RED "\nStep 6: Use mmap(2) to allocate a shared buffer of
337             size "
338             "equal to 1 page. Initialize the buffer and print the new "
339             "mapping information that has been created.\n" RESET);
340     press_enter();
341     /*
342     * TODO: Write your code here to complete Step 6.
343     */
344     heap_shared_buf = mmap(NULL, pagesize, PROT_EXEC | PROT_READ |
345                             PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
346     memset(heap_shared_buf, 0, pagesize);
347     pa = heap_shared_buf;
348
349     show_maps();
350     printf("0x%lu\n", get_physical_address(heap_shared_buf));
351
352     p = fork();
353     if (p < 0)
354         die("fork");
355     if (p == 0) {
356         child();
357         return 0;
358     }
359     parent(p);
360
361     if (-1 == close(fd))
362         perror("close");
363     return 0;
364 }

```

## Makefile

```

1  .PHONY: all clean
2
3  all: mmap
4
5  CC = gcc
6  CFLAGS = -g -Wall -Wextra -O2

```

```

7 SHELL= /bin/bash
8
9 mmap: mmap.o help.o
10 $(CC) $(CFLAGS) $^ -o $@
11
12 %.s: %.c
13 $(CC) $(CFLAGS) -S -fverbose-asm $<
14
15 %.o: %.c
16 $(CC) $(CFLAGS) -c $<
17
18 %.i: %.c
19 gcc -Wall -E $< | indent -kr > $@
20
21 clean:
22 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

```

1 /*
2  * mandel.c
3  *
4  * A program to draw the Mandelbrot Set on a 256-color xterm.
5  *
6  */
7
8 #include <stdio.h>
9 #include <unistd.h>

```

```

10 #include <assert.h>
11 #include <string.h>
12 #include <math.h>
13 #include <stdlib.h>
14 #include <errno.h>
15 #include <sys/types.h>
16 #include <sys/wait.h>
17 #include <sys/mman.h>
18 #include <semaphore.h>
19
20 #include "mandel-lib.h"
21
22 #define MANDEL_MAX_ITERATION 100000
23
24 /*****
25  * Compile-time parameters *
26  *****/
27
28 /*
29  * Output at the terminal is is x_chars wide by y_chars long
30  */
31 int y_chars = 50;
32 int x_chars = 90;
33
34 /*
35  * The part of the complex plane to be drawn:
36  * upper left corner is (xmin, ymax), lower right corner is (xmax,
37   * ymin)
38  */
39 double xmin = -1.8, xmax = 1.0;
40 double ymin = -1.0, ymax = 1.0;
41
42 /*
43  * Every character in the final output is
44  * xstep x ystep units wide on the complex plane.
45  */
46 double xstep;
47 double ystep;
48
49 void usage(char *argv0)
50 {
51     fprintf(stderr, "Usage: %s process_count \n\n"
52         "Exactly one argument required:\n"
53         "    process_count: The number of processes to create.\n",
54         argv0);
55     exit(1);
56 }
57
58 int safe_atoi(char *s, int *val)
59 {
60     long l;
61     char *endp;
62
63     l = strtol(s, &endp, 10);
64     if (s != endp && *endp == '\0') {
65         *val = l;
66     }
67 }

```



```

65     return 0;
66 } else
67     return -1;
68 }
69
70 /*
71  * This function computes a line of output
72  * as an array of x_char color values.
73  */
74 void compute_mandel_line(int line, int color_val[])
75 {
76     /*
77      * x and y traverse the complex plane.
78      */
79     double x, y;
80
81     int n;
82     int val;
83
84     /* Find out the y value corresponding to this line */
85     y = ymax - ystep * line;
86
87     /* and iterate for all points on this line */
88     for (x = xmin, n = 0; n < x_chars; x+= xstep, n++) {
89
90         /* Compute the point's color value */
91         val = mandel_iterations_at_point(x, y, MANDEL_MAX_ITERATION);
92         if (val > 255)
93             val = 255;
94
95         /* And store it in the color_val[] array */
96         val = xterm_color(val);
97         color_val[n] = val;
98     }
99 }
100
101 /*
102  * This function outputs an array of x_char color values
103  * to a 256-color xterm.
104  */
105 void output_mandel_line(int fd, int color_val[])
106 {
107     int i;
108
109     char point = '@';
110     char newline = '\n';
111
112     for (i = 0; i < x_chars; i++) {
113         /* Set the current color, then output the point */
114         set_xterm_color(fd, color_val[i]);
115         if (write(fd, &point, 1) != 1) {
116             perror("compute_and_output_mandel_line: write point");
117             exit(1);
118         }
119     }
120 }

```

```

121  /* Now that the line is done, output a newline character */
122  if (write(fd, &newline, 1) != 1) {
123      perror("compute_and_output_mandel_line: write newline");
124      exit(1);
125  }
126  }
127
128  void compute_and_output_mandel_line(int fd, int line, int
    processNumber, int processCount, sem_t* sems)
129  {
130      /*
131       * A temporary array, used to hold color values for the line
    being drawn
132       */
133      int color_val[x_chars];
134
135
136      //No synchronization needed for the calculation
137      compute_mandel_line(line, color_val);
138
139      //Synchronization is added when writing to the output
140      //Get the previous semaphore's position (if we are at position 0
    get the last semaphore)
141      int previousSemPosition = (processNumber + processCount - 1) %
    processCount;
142
143      //Wait if previous process has not finished
144      sem_wait(&sems[previousSemPosition]);
145      output_mandel_line(fd, color_val);
146      //When finished signal to the next process
147      sem_post(&sems[processNumber]);
148  }
149
150
151  /*
152   * Function that is run by the child processes
153   */
154  void process_fn(int processNumber, int processCount, sem_t* sems) {
155      int line;
156      for (line = processNumber; line < y_chars; line += processCount)
    {
157          compute_and_output_mandel_line(1, line, processNumber,
    processCount, sems);
158      }
159
160      return;
161  }
162
163  /*
164   * Create a shared memory area, usable by all descendants of the
    calling
165   * process.
166   */
167  void *create_shared_memory_area(unsigned int numbytes)
168  {
169      int pages, totalPageBytes;

```

```

170     void *addr;
171
172     if (numbytes == 0) {
173         fprintf(stderr, "%s: internal error: called for
174             numbytes == 0\n", __func__);
175         exit(1);
176     }
177
178     /*
179      * Determine the number of pages needed, round up the
180      * requested number of
181      * pages
182      */
183     pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;
184     totalPageBytes = pages * sysconf(_SC_PAGE_SIZE);
185
186     /* Create a shared, anonymous mapping for this number of
187      * pages */
188     addr = mmap(NULL, totalPageBytes, PROT_READ | PROT_WRITE ,
189         MAP_SHARED | MAP_ANONYMOUS, -1, 0 );
190
191     return addr;
192 }
193
194 void destroy_shared_memory_area(void *addr, unsigned int numbytes)
195 {
196     int pages;
197
198     if (numbytes == 0) {
199         fprintf(stderr, "%s: internal error: called for
200             numbytes == 0\n", __func__);
201         exit(1);
202     }
203
204     /*
205      * Determine the number of pages needed, round up the
206      * requested number of
207      * pages
208      */
209     pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;
210
211     if (munmap(addr, pages * sysconf(_SC_PAGE_SIZE)) == -1) {
212         perror("destroy_shared_memory_area: munmap failed")
213         ;
214         exit(1);
215     }
216 }
217
218 int main(int argc, char *argv[])
219 {
220     if (argc != 2)
221         usage(argv[0]);
222
223     int processCount;

```

```

218     if (safe_atoi(argv[1], &processCount) < 0 || processCount <= 0) {
219         fprintf(stderr, "'%s' is not valid for 'process_count'\n", argv
220             [1]);
221         exit(1);
222     }
223
224     xstep = (xmax - xmin) / x_chars;
225     ystep = (ymax - ymin) / y_chars;
226
227     //Create shared memory for semaphore
228     sem_t* sems = create_shared_memory_area(processCount*sizeof(sem_t
229         *));
230
231     //Initialize semaphores
232     int i;
233     for (i = 0; i < processCount; i++) {
234         sem_init(&sems[i], 1, ((i==processCount-1)?1:0));
235     }
236
237     pid_t pid;
238     int status;
239
240     //Create child processes
241     for (i = 0; i < processCount; i++) {
242         //fork from main to create child process
243         pid = fork();
244         if (pid < 0) {
245             perror("Fork error");
246             exit(1);
247         } else if (pid == 0) {
248             //Child process enters here
249             process_fn(i, processCount, sems);
250             exit(0);
251         }
252     }
253
254     //Wait for all the children to finish
255     while ((pid = wait(&status) > 0));
256
257     //Clear shared memory
258     destroy_shared_memory_area(sems, processCount*sizeof(sem_t*));
259
260     reset_xterm_color(1);
261
262     return 0;
263 }

```

## 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

```
1  /*
2   * mandel.c
3   *
4   * A program to draw the Mandelbrot Set on a 256-color xterm.
5   *
6   */
7
8  #include <stdio.h>
9  #include <unistd.h>
10 #include <assert.h>
11 #include <string.h>
12 #include <math.h>
13 #include <stdlib.h>
14 #include <errno.h>
15 #include <sys/types.h>
16 #include <sys/wait.h>
17 #include <sys/mman.h>
18 #include <semaphore.h>
19
20 #include "mandel-lib.h"
21
22 #define MANDEL_MAX_ITERATION 100000
23
24 /*****
25  * Compile-time parameters *
26  *****/
27
28 /*
```

```

29  * Output at the terminal is is x_chars wide by y_chars long
30  */
31  int y_chars = 50;
32  int x_chars = 90;
33
34  /*
35   * The part of the complex plane to be drawn:
36   * upper left corner is (xmin, ymax), lower right corner is (xmax,
37   *   ymin)
38   */
39  double xmin = -1.8, xmax = 1.0;
40  double ymin = -1.0, ymax = 1.0;
41  /*
42   * Every character in the final output is
43   * xstep x ystep units wide on the complex plane.
44   */
45  double xstep;
46  double ystep;
47
48  void usage(char *argv0)
49  {
50      fprintf(stderr, "Usage: %s process_count \n\n"
51          "Exactly one argument required:\n"
52          "    process_count: The number of processes to create.\n",
53          argv0);
54      exit(1);
55  }
56
57  int safe_atoi(char *s, int *val)
58  {
59      long l;
60      char *endp;
61
62      l = strtol(s, &endp, 10);
63      if (s != endp && *endp == '\0') {
64          *val = l;
65          return 0;
66      } else
67          return -1;
68  }
69
70  /*
71   * This function computes a line of output
72   * as an array of x_char color values.
73   */
74  void compute_mandel_line(int line, int color_val[])
75  {
76      /*
77       * x and y traverse the complex plane.
78       */
79      double x, y;
80
81      int n;
82      int val;
83

```

```

84  /* Find out the y value corresponding to this line */
85  y = ymax - ystep * line;
86
87  /* and iterate for all points on this line */
88  for (x = xmin, n = 0; n < x_chars; x+= xstep, n++) {
89
90      /* Compute the point's color value */
91      val = mandel_iterations_at_point(x, y, MANDEL_MAX_ITERATION);
92      if (val > 255)
93          val = 255;
94
95      /* And store it in the color_val[] array */
96      val = xterm_color(val);
97      color_val[n] = val;
98  }
99  }
100
101  /*
102   * This function outputs an array of x_char color values
103   * to a 256-color xterm.
104   */
105  void output_mandel_line(int fd, int color_val[])
106  {
107      int i;
108
109      char point = '@';
110      char newline = '\n';
111
112      for (i = 0; i < x_chars; i++) {
113          /* Set the current color, then output the point */
114          set_xterm_color(fd, color_val[i]);
115          if (write(fd, &point, 1) != 1) {
116              perror("compute_and_output_mandel_line: write point");
117              exit(1);
118          }
119      }
120
121      /* Now that the line is done, output a newline character */
122      if (write(fd, &newline, 1) != 1) {
123          perror("compute_and_output_mandel_line: write newline");
124          exit(1);
125      }
126  }
127
128  void compute_and_store_mandel_line(int fd, int line, int
129      processNumber, int processCount, int** buffer)
130  {
131      /*
132       * A temporary array, used to hold color values for the line
133       * being drawn
134       */
135      int color_val[x_chars];
136
137      compute_mandel_line(line, color_val);

```



```

138     int i;
139
140     for (i=0;i<x_chars;i++) {
141         buffer[line][i] = color_val[i];
142     }
143 }
144
145
146 void process_fn(int processNumber, int processCount, int** buffer)
147 {
148     int line;
149     for (line = processNumber; line < y_chars; line += processCount)
150     {
151         compute_and_store_mandel_line(1, line, processNumber,
152                                     processCount, buffer);
153     }
154
155     return;
156 }
157
158 /*
159  * Create a shared memory area, usable by all descendants of the
160  * calling
161  * process.
162  */
163 void *create_shared_memory_area(unsigned int numbytes)
164 {
165     int pages, totalPageBytes;
166     void *addr;
167
168     if (numbytes == 0) {
169         fprintf(stderr, "%s: internal error: called for
170                 numbytes == 0\n", __func__);
171         exit(1);
172     }
173
174     /*
175      * Determine the number of pages needed, round up the
176      * requested number of
177      * pages
178      */
179     pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;
180     totalPageBytes = pages * sysconf(_SC_PAGE_SIZE);
181
182     /* Create a shared, anonymous mapping for this number of
183        pages */
184     addr = mmap(NULL, totalPageBytes, PROT_READ | PROT_WRITE,
185                MAP_SHARED | MAP_ANONYMOUS, -1, 0);
186
187     return addr;
188 }
189
190 void destroy_shared_memory_area(void *addr, unsigned int numbytes)
191 {
192     int pages;
193
194

```

```

185     if (numbytes == 0) {
186         fprintf(stderr, "%s: internal error: called for
187             numbytes == 0\n", __func__);
188         exit(1);
189     }
190     /*
191     * Determine the number of pages needed, round up the
192     * requested number of
193     * pages
194     */
195     pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;
196     if (munmap(addr, pages * sysconf(_SC_PAGE_SIZE)) == -1) {
197         perror("destroy_shared_memory_area: munmap failed")
198         ;
199         exit(1);
200     }
201 }
202 int main(int argc, char *argv[])
203 {
204     if (argc != 2)
205         usage(argv[0]);
206
207     int processCount;
208
209     if (safe_atoi(argv[1], &processCount) < 0 || processCount <= 0) {
210         fprintf(stderr, "'%s' is not valid for 'process_count'\n", argv
211             [1]);
212         exit(1);
213     }
214
215     xstep = (xmax - xmin) / x_chars;
216     ystep = (ymax - ymin) / y_chars;
217
218     //Initialize buffer in shared memory
219     int** buffer = (int**) create_shared_memory_area(x_chars*sizeof(
220         int*));
221
222     int i;
223     for (i=0;i<x_chars;i++) {
224         buffer[i] = (int*) create_shared_memory_area(y_chars*sizeof(int
225             ));
226     }
227
228     pid_t pid;
229     int status;
230
231     for (i = 0; i < processCount; i++) {
232         pid = fork(); //fork from main to create child process
233         if (pid < 0) {
234             perror("Fork error");
235             exit(1);

```

```

235     } else if (pid == 0) {
236         process_fn(i, processCount, buffer);
237         exit(0);
238     }
239 }
240
241 while ((pid = wait(&status) > 0));
242
243 //Outputs mandel line
244 //We loop through all the lines of the buffer and print them one
    by one
245 int line;
246 for (line=0;line<y_chars;line++) {
247     int color_val[x_chars];
248     for (i=0;i<x_chars;i++) {
249         color_val[i] = buffer[line][i];
250     }
251
252     output_mandel_line(1, color_val);
253 }
254
255 //Clear shared memory
256 for (i=0;i<x_chars;i++) {
257     destroy_shared_memory_area(buffer[i], y_chars*sizeof(int));
258 }
259 destroy_shared_memory_area(buffer, x_chars*sizeof(int));
260
261 reset_xterm_color(1);
262
263 return 0;
264 }

```

## Makefile

```

1  #
2  # Makefile
3  #
4
5  CC = gcc
6
7  CFLAGS = -Wall -O2 -pthread
8  LIBS =
9
10 all: mandel-sem mandel-buffer
11
12
13 ## Mandel
14
15 mandel-sem: mandel-lib.o mandel-sem.o
16     $(CC) $(CFLAGS) -o mandel-sem mandel-lib.o mandel-sem.o $(LIBS)
17
18 mandel-buffer: mandel-lib.o mandel-buffer.o
19     $(CC) $(CFLAGS) -o mandel-buffer mandel-lib.o mandel-buffer.o $(LIBS)

```

```

20
21 mandel-lib.o: mandel-lib.h mandel-lib.c
22 $(CC) $(CFLAGS) -c -o mandel-lib.o mandel-lib.c $(LIBS)
23
24 mandel-sem.o: mandel-sem.c
25 $(CC) $(CFLAGS) -c -o mandel-sem.o mandel-sem.c $(LIBS)
26
27 mandel-buffer.o: mandel-buffer.c
28 $(CC) $(CFLAGS) -c -o mandel-buffer.o mandel-buffer.c $(LIBS)
29
30 clean:
31 rm -f *.o mandel-sem mandel-buffer

```

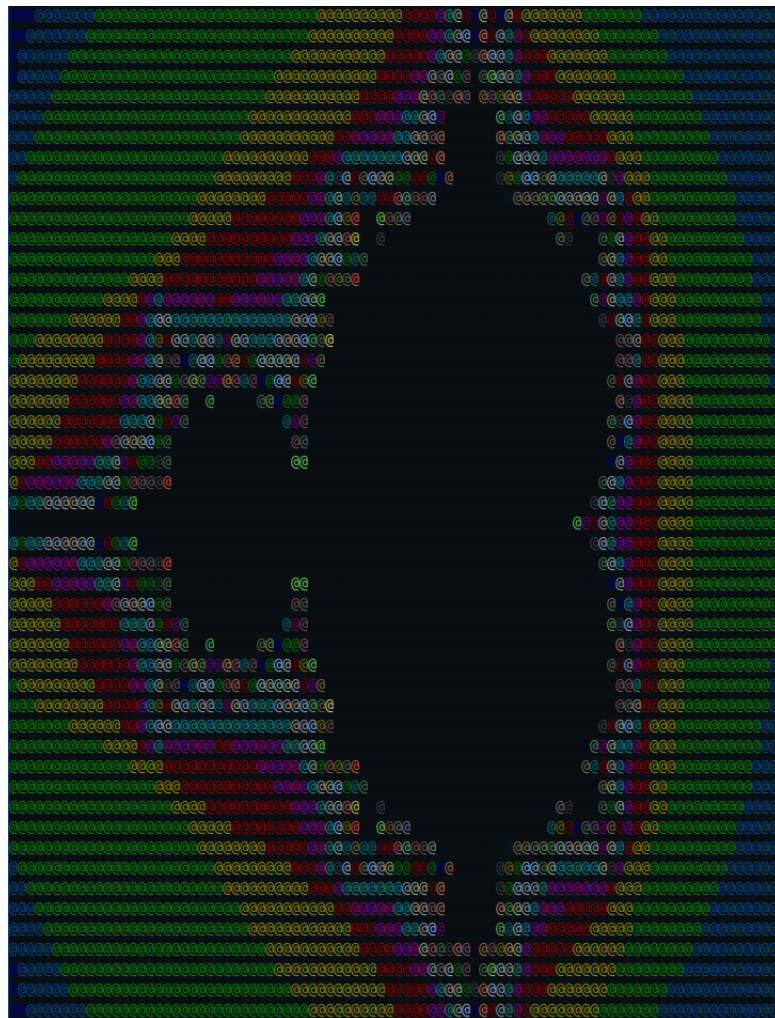


Figure 1: Output του mandel-buffer. Προφανώς το ίδιο και για mandel-sem