



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Λειτουργικά Συστήματα Υπολογιστών

**4η εργαστηριακή αναφορά:
Μηχανισμοί Εικονικής Μνήμης**

Διδάσκοντες:

N. Κοζύρης
Γ. Γκούμας

oslaba84:

Ειρήνη Δόντη
AM 03119839

6ο εξάμηνο

Αθήνα 2022

Περιεχόμενα

1.1 Κλήσεις συστήματος και βασικοί μηχανισμοί του ΛΣ για τη διαχείριση της εικονικής μνήμης (Virtual Memory - VM).....σελ 2	
1.2 Παράλληλος υπολογισμός Mandelbrot με διεργασίες αντί για νήματα.....σελ 8	
1.2.1 Semaphores πάνω από διαμοιραζόμενη μνήμη.....σελ 8	
1.2.2 Υλοποίηση χωρίς semaphores.....σελ 10	

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

1. Τυπώνουμε το χάρτη της εικονικής μνήμης της τρέχουσας διεργασίας.

```
oslab@os-node1:~/myvm/mmap$ ./mmap

Step 1: Print the virtual address space map of this process [3362].

Virtual Memory Map of process [3362]:
00400000-00402000 r-xp 00000000 00:21 9577090 /home/oslab/oslab84/myvm/mmap/mmap
00601000-00602000 rw-p 00001000 00:21 9577090 /home/oslab/oslab84/myvm/mmap/mmap
00739000-0075a000 rw-p 00000000 00:00 0 [heap]
7f1edae41000-7f1edafe2000 r-xp 00000000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f1edafe2000-7f1edb1e2000 ---p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f1edb1e2000-7f1edb1e6000 r--p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f1edb1e6000-7f1edb1e8000 rw-p 001a5000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f1edb1e8000-7f1edb1ec000 rw-p 00000000 00:00 0
7f1edb1ec000-7f1edb20d000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f1edb20d000-7f1edb402000 rw-p 00000000 00:00 0
7f1edb402000-7f1edb40c000 rw-p 00000000 00:00 0
7f1edb40c000-7f1edb40d000 r--p 00020000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f1edb40d000-7f1edb40e000 rw-p 00021000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f1edb40e000-7f1edb40f000 rw-p 00000000 00:00 0
7ffe1fc3c000-7ffe1fc5d000 rw-p 00000000 00:00 0 [stack]
7ffe1fd8e000-7ffe1fd91000 r--p 00000000 00:00 0 [vvar]
7ffe1fd91000-7ffe1fd93000 r-xp 00000000 00:00 0 [vdso]
fffffffff600000-fffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
-----
```

2. Με κλήση συστήματος mmap() δεσμεύουμε buffer μεγέθους μιας σελίδας και τυπώνουμε ξανά το χάρτη.

```
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 [3362]:
00400000-00402000 r-xp 00000000 00:21 9577090 /home/oslab/oslab84/myvm/mmap/mmap
00601000-00602000 rw-p 00001000 00:21 9577090 /home/oslab/oslab84/myvm/mmap/mmap
00739000-0075a000 rw-p 00000000 00:00 0 [heap]
7f1edae41000-7f1edafe2000 r-xp 00000000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f1edafe2000-7f1edb1e2000 ---p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f1edb1e2000-7f1edb1e6000 r--p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f1edb1e6000-7f1edb1e8000 rw-p 001a5000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f1edb1e8000-7f1edb1ec000 rw-p 00000000 00:00 0
7f1edb1ec000-7f1edb20d000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f1edb20d000-7f1edb402000 rw-p 00000000 00:00 0
7f1edb402000-7f1edb40c000 rw-p 00000000 00:00 0
7f1edb40c000-7f1edb40d000 r--p 00020000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f1edb40d000-7f1edb40e000 rw-p 00021000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f1edb40e000-7f1edb40f000 rw-p 00000000 00:00 0
7ffe1fc3c000-7ffe1fc5d000 rw-p 00000000 00:00 0
7ffe1fd8e000-7ffe1fd91000 r--p 00000000 00:00 0 [stack]
7ffe1fd91000-7ffe1fd93000 r-xp 00000000 00:00 0 [vvar]
7ffe1fd93000-7ffe1fd93000 r-xp 00000000 00:00 0 [vdso]
fffffffff600000-fffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
-----
```

Ο χώρος εικονικών διευθύνσεων που δεσμεύσαμε, εντοπίζεται στο σημείο που υπάρχει το κόκκινο βέλος.

3. Βρίσκουμε και τυπώνουμε τη φυσική διεύθυνση μνήμης στην οποία απεικονίζεται η εικονική διεύθυνση του buffer.

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

The physical address is: VA[0x7f909267f000] is not mapped; no physical memory allocated.
```

Παρατηρούμε ότι δεν κατανεμήθηκε φυσική μνήμη από τον buffer. Αυτό συμβαίνει, καθώς ο buffer δεν αρχικοποιείται και συνεπώς λόγω του demand page.

4. Γεμίζουμε με μηδενικά τον buffer και επαναλαμβάνουμε το βήμα 3.

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

Now the buffer is full of zeros
The new physical address is: 4836962304

Παρατηρούμε ότι, σε αυτή την περίπτωση, κατανεμήθηκε φυσική μνήμη από τον buffer, καθώς ο buffer αρχικοποιήθηκε με μηδενικά.

5. Χρησιμοποιούμε την mmap() για να απεικονίσουμε το αρχείο file.txt στο χώρο διευθύνσεων της διεργασίας σας και τυπώνουμε το περιεχόμενό του.

Εντοπίζουμε τη νέα απεικόνιση στο χάρτη μνήμης.

```
Open file.txt . . .
The file contains:
Hello everyone!

Virtual Memory Map of process [21397]:
00400000-00403000 r-xp 00000000 00:21 9577090 /home/oslab/oslab84/myvm/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 9577090 /home/oslab/oslab84/myvm/mmap/mmap
020e3000-02104000 rw-p 00000000 00:00 0 [heap]
7f8c6d583000-7f8c6d724000 r-xp 00000000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d724000-7f8c6d924000 ---p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d924000-7f8c6d928000 r--p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d928000-7f8c6d92a000 rw-p 001a5000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d92a000-7f8c6d92e000 rw-p 00000000 00:00 0
7f8c6d92e000-7f8c6d94f000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f8c6db41000-7f8c6db44000 rw-p 00000000 00:00 0
7f8c6db47000-7f8c6db48000 rw-p 00000000 00:00 0
7f8c6db48000-7f8c6db49000 r--s 00000000 00:21 9577097 /home/oslab/oslab84/myvm/mmap/file.txt
7f8c6db49000-7f8c6db4a000 rwxp 00000000 00:00 0
7f8c6db4a000-7f8c6db4e000 rw-p 00000000 00:00 0
7f8c6db4e000-7f8c6db4f000 r--p 00020000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f8c6db4f000-7f8c6db50000 rw-p 00021000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f8c6db50000-7f8c6db51000 rw-p 00000000 00:00 0
7ffdaaf6e000-7ffdaaf8f000 rw-p 00000000 00:00 0 [stack]
7ffdaafa0000-7ffdaafa3000 r--p 00000000 00:00 0 [vvar]
7ffdaafa3000-7ffdaafa5000 r-xp 00000000 00:00 0 [vdso]
7ffdaafa5000-7ffdaafa6000 r-xp 00000000 00:00 0 [vsyscall]
-----
The physical address of file buffer is: 4833214464
```

6. Χρησιμοποιούμε την mmap() για να δεσμεύσουμε ένα νέο buffer, διαμοιραζόμενο αυτή τη φορά μεταξύ διεργασιών με μέγεθος μίας σελίδας.

Εντοπίζουμε τη νέα απεικόνιση στο χάρτη μνήμης.

```
Virtual Memory Map of process [21397]:
00400000-00403000 r-xp 00000000 00:21 9577090 /home/oslab/oslab84/myvm/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 9577090 /home/oslab/oslab84/myvm/mmap/mmap
020e3000-02104000 rw-p 00000000 00:00 0 [heap]
7f8c6d583000-7f8c6d724000 r-xp 00000000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d724000-7f8c6d924000 ---p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d924000-7f8c6d928000 r--p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d928000-7f8c6d92a000 rw-p 001a5000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d92a000-7f8c6d92e000 rw-p 00000000 00:00 0
7f8c6d92e000-7f8c6d94f000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f8c6db41000-7f8c6db44000 rw-p 00000000 00:00 0
7f8c6db47000-7f8c6db48000 rw-p 00000000 00:00 0
7f8c6db48000-7f8c6db49000 rwx 00000000 00:04 3440985 /dev/zero (deleted)
7f8c6db49000-7f8c6db4a000 r--s 00000000 00:21 9577097 /home/oslab/oslab84/myvm/mmap/file.txt
7f8c6db4a000-7f8c6db4e000 rwxp 00000000 00:00 0
7f8c6db4e000-7f8c6db4f000 rw-p 00000000 00:00 0
7f8c6db4f000-7f8c6db50000 r--p 00020000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f8c6db50000-7f8c6db51000 rw-p 00021000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f8c6db51000-7f8c6db52000 rw-p 00000000 00:00 0
7ffdaaf6e000-7ffdaaf8f000 rw-p 00000000 00:00 0 [stack]
7ffdaafa0000-7ffdaafa3000 r--p 00000000 00:00 0 [vvar]
7ffdaafa3000-7ffdaafa5000 r-xp 00000000 00:00 0 [vdso]
7ffdaafa5000-7ffdaafa6000 r-xp 00000000 00:00 0 [vsyscall]
-----
The physical address of heap buffer is: 4865964033
```

7. Τυπώνουμε το χάρτη της εικονικής μνήμης της διεργασίας πατέρα και της διεργασίας παιδιού.

```

Parent - VM map is:

Virtual Memory Map of process [21397]:
00400000-00403000 r-xp 00000000 00:21 9577090 /home/oslab/oslab84/myvm/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 9577090 /home/oslab/oslab84/myvm/mmap/mmap
020e3000-02104000 rw-p 00000000 00:00 0 [heap]
7f8c6d583000-7f8c6d724000 r-xp 00000000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d724000-7f8c6d924000 ---p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d924000-7f8c6d928000 r--p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d928000-7f8c6d92a000 rw-p 001a5000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d92a000-7f8c6d92e000 rw-p 00000000 00:00 0
7f8c6d92e000-7f8c6d94f000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f8c6db41000-7f8c6db44000 rw-p 00000000 00:00 0
7f8c6db46000-7f8c6db47000 rw-p 00000000 00:00 0
7f8c6db47000-7f8c6db48000 rwxS 00000000 00:04 3440985 /dev/zero (deleted)
7f8c6db48000-7f8c6db49000 r--s 00000000 00:21 9577097 /home/oslab/oslab84/myvm/mmap/file.txt
7f8c6db49000-7f8c6db4a000 rwxp 00000000 00:00 0
7f8c6db4a000-7f8c6db4e000 rw-p 00000000 00:00 0
7f8c6db4e000-7f8c6db4f000 r--p 00020000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f8c6db4f000-7f8c6db50000 rw-p 00021000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f8c6db50000-7f8c6db51000 rw-p 00000000 00:00 0
7ffdaaf6e000-7ffdaaf8f000 rw-p 00000000 00:00 0 [stack]
7ffdaafa0000-7ffdaafa3000 r--p 00000000 00:00 0 [vvar]
7ffdaafa3000-7ffdaafa5000 r-xp 00000000 00:00 0 [vdso]
fffffffff600000-fffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
-----

Child - VM map is:

Virtual Memory Map of process [21398]:
00400000-00403000 r-xp 00000000 00:21 9577090 /home/oslab/oslab84/myvm/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 9577090 /home/oslab/oslab84/myvm/mmap/mmap
020e3000-02104000 rw-p 00000000 00:00 0 [heap]
7f8c6d583000-7f8c6d724000 r-xp 00000000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d724000-7f8c6d924000 ---p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d924000-7f8c6d928000 r--p 001a1000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d928000-7f8c6d92a000 rw-p 001a5000 08:01 6032227 /lib/x86_64-linux-gnu/libc-2.19.so
7f8c6d92a000-7f8c6d92e000 rw-p 00000000 00:00 0
7f8c6d92e000-7f8c6d94f000 r-xp 00000000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f8c6db41000-7f8c6db44000 rw-p 00000000 00:00 0
7f8c6db46000-7f8c6db47000 rw-p 00000000 00:00 0
7f8c6db47000-7f8c6db48000 rwxS 00000000 00:04 3440985 /dev/zero (deleted)
7f8c6db48000-7f8c6db49000 r--s 00000000 00:21 9577097 /home/oslab/oslab84/myvm/mmap/file.txt
7f8c6db49000-7f8c6db4a000 rwxp 00000000 00:00 0
7f8c6db4a000-7f8c6db4e000 rw-p 00000000 00:00 0
7f8c6db4e000-7f8c6db4f000 r--p 00020000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f8c6db4f000-7f8c6db50000 rw-p 00021000 08:01 6032224 /lib/x86_64-linux-gnu/ld-2.19.so
7f8c6db50000-7f8c6db51000 rw-p 00000000 00:00 0
7ffdaaf6e000-7ffdaaf8f000 rw-p 00000000 00:00 0 [stack]
7ffdaafa0000-7ffdaafa3000 r--p 00000000 00:00 0 [vvar]
7ffdaafa3000-7ffdaafa5000 r-xp 00000000 00:00 0 [vdso]
fffffffff600000-fffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
-----

```

Παρατηρούμε ότι ο χάρτης εικονικής μνήμης της διεργασίας παιδί και της διεργασίας πατέρα ταυτίζεται, καθώς η διεργασία παιδί κληρονομεί τις θέσεις μνήμης της διεργασίας πατέρα.

8. Βρίσκουμε και τυπώνουμε τη φυσική διεύθυνση στην κύρια μνήμη του private buffer για τις διεργασίες πατέρα και παιδί. Παρατηρούμε ότι, μετά το fork(), η φυσική διεύθυνση μνήμης της διεργασίας παιδί ταυτίζεται με τη φυσική διεύθυνση μνήμης της διεργασίας πατέρα.

```

Parent - The physical address of private buffer is: 4412862464
7ff14ddff000-7ff14de00000 rwxp 00000000 00:00 0
Child - The physical address of private buffer is: 4412862464
7ff14ddff000-7ff14de00000 rwxp 00000000 00:00 0

```


9. Γράφουμε στον private buffer από τη διεργασία παιδί και επαναλαμβάνουμε το Βήμα 8. Παρατηρούμε ότι οι φυσικές διευθύνσεις στις διεργασίες πατέρα και παιδί διαφέρουν, καθώς γράφουμε στον private buffer από τη διεργασία παιδί και συνεπώς η διεργασία πατέρα δεν μπορεί να “επιβλέψει” την οποιαδήποτε αλλαγή στη διεργασία παιδί.

```
Parent - VA info of private buffer is: 7ff14ddff000-7ff14de00000 rwxp 00000000
00:00 0
4412862464
7ff14ddff000-7ff14de00000 rwxp 00000000 00:00 0
Child - The physical address of private buffer is: 176394240
```

10. Γράφουμε στον shared buffer από τη διεργασία παιδί και τυπώνουμε τη φυσική διεύθυνση για τις διεργασίες πατέρα παιδί. Σε σχέση με την private buffer, η φυσική μνήμη των διεργασιών πατέρα και παιδί ταυτίζεται. Αυτό συμβαίνει, καθώς η shared buffer επιτρέπει τον διαμοιρασμό πληροφοριών μεταξύ των διεργασιών πατέρα και παιδί και συνεπώς η φυσική μνήμη για τις διεργασίες πατέρα παιδί είναι ταυτόσημες.

```
Parent - VA info of shared buffer is: 00400000-00403000 r-xp 00000000 00:21 957
7090 /home/oslab/oslab84/myvm/mmap/mmap
1157851137
```

11. Απαγορεύουμε τις εγγραφές στον shared buffer για τη διεργασία παιδί. Εντοπίζουμε και τυπώνουμε την απεικόνιση του shared buffer στο χάρτη μνήμης των δύο διεργασιών για να επιβεβαιώσουμε την απαγόρευση.

```
Parent - VM map is:

Virtual Memory Map of process [14977]:
00400000-00403000 r-xp 00000000 00:21 9577090
oslab/oslab84/myvm/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 9577090
oslab/oslab84/myvm/mmap/mmap
01790000-017b1000 rw-p 00000000 00:00 0
7fe656c5e000-7fe656dff000 r-xp 00000000 08:01 6032227
86_64-linux-gnu/libc-2.19.so
7fe656dff000-7fe656fff000 ---p 001a1000 08:01 6032227
86_64-linux-gnu/libc-2.19.so
7fe656fff000-7fe657003000 r--p 001a1000 08:01 6032227
86_64-linux-gnu/libc-2.19.so
7fe657003000-7fe657005000 rw-p 001a5000 08:01 6032227
86_64-linux-gnu/libc-2.19.so
7fe657005000-7fe657009000 rw-p 00000000 00:00 0
7fe657009000-7fe65702a000 r-xp 00000000 08:01 6032224
86_64-linux-gnu/ld-2.19.so
7fe65721c000-7fe65721f000 rw-p 00000000 00:00 0
```



```

/* 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 "↳Step 7: Print parent's and child's map.\n RESET");
press_enter();
printf(
    "Parent's VM map is: \n");
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)) // WUNTRACED = return if a child has stopped
    die("waitpid");

/*
 * Step 8: Get the physical memory address for heap_private_buf.
 * Step 8 - Parent
 */
printf(RED "↳Step 8: Find the physical address of the private buffer (heap_private_buf) 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's physical address of private buffer is: ");
pa = get_physical_address((uint64_t)heap_private_buf); ///////////////
printf(
    "PRId64 %u, pa); ///////////////

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

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

/*
 * TODO Write your code here to complete parent's part of Step 9.
 */

printf(
    "Parent's VM map of private buffer is: ");
show_va_info((uint64_t)heap_private_buf);
pa = get_physical_address((uint64_t)heap_private_buf); ///////////////
printf(
    "PRId64 %u, pa); ///////////////

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

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

/*
 * TODO Write your code here to complete parent's part of Step 10.
 */

printf(
    "Parent's VM map of heap_shared_buf is: ");
show_va_info((uint64_t)heap_shared_buf);
pa = get_physical_address((uint64_t)heap_shared_buf); ///////////////
printf(
    "PRId64 %u, pa); ///////////////

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

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

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

printf(
    "Parent's VM map is: \n");
show_maps();
show_va_info((uint64_t)heap_shared_buf);
if (-1 == kill(child_pid, SIGCONT))
    die("kill");
if (-1 == waitpid(child_pid, &status, 0))
    die("waitpid");

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

/*
 * TODO Write your code here to complete parent's part of Step 12.
 */

munmap(heap_shared_buf, buffer_size); // deallocate buffer - destroy mapping
munmap(heap_private_buf, buffer_size);
munmap(file_shared_buf, buffer_size);
}

int main(void)
{
    pid_t mypid, pi;
    int fd = 0;
    uint64_t pa;
    mypid = getpid();

    buffer_size = 4 * get_page_size();

    /*
     * Step 1: Print the virtual address space layout of this process.
     */
    printf(RED "↳Step 1: Print the virtual address space map of this "
        "process.\n RESET, mypid);
    press_enter();

    /*
     * TODO Write your code here to complete Step 1.
     */

    show_maps(); //print memory mapping of current process

    /*
     * Step 2: Use mmap to allocate a buffer of 1 page and print the map
     * again. Store buffer in heap_private_buf.
     */
    printf(RED "↳Step 2: Use mmap() to allocate a private buffer "
        "of 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(0, buffer_size, PROT_READ | PROT_WRITE | PROT_EXEC, MAP_ANONYMOUS | MAP_PRIVATE, fd, 0); //call mmap to allocate buffer with one page size
    show_maps(); //print VM map of heap_private_buf
    show_va_info((uint64_t)heap_private_buf); //creates new mapping in virtual address space of calling process - print info for mapping

    /*
     * Step 3: Find the physical address of the first page of your buffer
     * in main memory. What do you see?
     */
    printf(RED "↳Step 3: Find the physical address of the "
        "first page of your buffer in main memory. What do you see?\n RESET");
    press_enter();

    /*
     * TODO Write your code here to complete Step 3.
     */
    printf(
        "Parent's physical address of heap_private_buf is: ");
    pa = get_physical_address((uint64_t)heap_private_buf); // Find and return physical memory in main memory of private buffer
    printf(
        "PRId64 %u, pa); ///////////////

    /*
     * Step 4: Write zeros to the buffer and repeat Step 3.
     */
    printf(RED "↳Step 4: Initialize the buffer with zeros and repeat "
        "Step 3. What happened?\n RESET");
    press_enter();

    /*
     * TODO Write your code here to complete Step 4.
     */
    memset(heap_private_buf, 0, buffer_size); //fill first buffer_size bytes of memory area pointed to by heap_private_buf with constant byte 0

    printf(
        "Parent's VM map of heap_private_buf is: ");
    show_maps();
    pa = get_physical_address((uint64_t)heap_private_buf); // Find and return physical memory in main memory of private buffer
    printf(
        "PRId64 %u, pa); ///////////////

    /*
     * Step 5: Use mmap(2) to map file.txt (memory-mapped files) and print
     * its content. Use file_shared_buf.
     */
    printf(RED "↳Step 5: Use mmap() to read and print file.txt. Repeat "
        "Step 3. What happened?\n RESET");
    press_enter();

    /*
     * TODO Write your code here to complete Step 5.
     */

    printf(
        "Parent's VM map of file_shared_buf is: ");
    show_maps();
    fd = open("file.txt", O_RDONLY); // open read-only file
    printf(
        "fd = %d\n", fd); //print fd
    printf(
        "Parent's VM map of file_shared_buf is: ");
    show_maps();
}

```



```

file_shared_buf = mmap(NULL, buffer_size, PROT_READ, MAP_SHARED, fd, 0); // allocate buffer size of one page. PROT_READ = the memory can be read, MAP_SHARED = if you write to map == changes writ
// go back to 2.1 (1ml)
if(file_shared_buf == MAP_FAILED) { //////////
    printf("mmap failed\n"); //////////
    exit(-1); //////////
}

printf("The file contains: \n"); //////////
printf("%s\n", file_shared_buf); //////////

show_map(); //////////
pa = get_physical_address((uint64_t)file_shared_buf); //////////
printf("4096 = 0x%x\n", pa); //////////
printf("0x PRId64\n", pa); //////////

/*
 * Step d: Use mmap(2) to allocate a shared buffer of 1 page. Use
 * heap_shared_buf.
 */
printf(RED "Use mmap(2) to allocate a shared buffer of size 1\n");
// (1ml) mmap: initialise the buffer and print the new
// information that has been created (0x RESET);
press_enter();

// (1ml) Write your code here to complete step e.
//

heap_shared_buf = mmap(heap_shared_buf, buffer_size, PROT_READ | PROT_WRITE | PROT_EXEC, MAP_SHARED | MAP_ANONYMOUS, -1, 0); // allocate of new shared buffer MAP_ANONYMOUS = every call creates a
// (1ml)
show_map();
pa = get_physical_address((uint64_t)heap_shared_buf); //////////
printf("0x PRId64\n", pa); //////////

p = fork();
if (p < 0)
    die("fork");
if (p == 0)
    child();
else
    parent(p);

if (!close(fd))
    perror("close");
return 0;
}

```

1.2 Παράλληλος υπολογισμός Mandelbrot με διεργασίες αντί για νήματα.

1.2.1 Semaphores πάνω από διαμοιραζόμενη μνήμη

Ερωτήσεις:

1. Η υλοποίηση με νήματα έχει καλύτερη επίδοση, καθώς οι διεργασίες δεν μοιράζονται τη μνήμη τους με άλλες διεργασίες.

Η υλοποίηση διεργασιών με τη βοήθεια semaphores, αυξάνει την επίδοση της υλοποίησης, καθώς βρίσκονται σε διαμοιραζόμενη μνήμη διεργασιών και συνεπώς η διεργασία μπορεί να επικοινωνήσει με τις υπόλοιπες διεργασίες. Με άλλα λόγια, οι τροποποιήσεις των διεργασιών γίνονται στον ίδιο χώρο μνήμης που εξασφαλίζει ο διαμοιρασμός μνήμης πάνω στους semaphores, με αποτέλεσμα να μη χρειάζεται σε κάθε τροποποίηση της διεργασίας να μεταφέρονται τα δεδομένα σε μία άλλη για να επιτευχθεί ο συγχρονισμός.

Ο κώδικας που χρησιμοποιήσαμε είναι ο ακόλουθος:

```

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

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <limits.h>
#include <math.h>

#define MANDEL_MAX_ITERATION 100000

char *heap_private_buf;
char *heap_shared_buf;
sem_t *sem_array;

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

/* Output at the terminal is k_chars wide by y_chars long
 */
int y_chars = 80;
int k_chars = 256;

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

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

```

int safe_atoi(char *s, int *val)
{
    long i;
    char *endp;
    long strtol(s, &endp, 0);
    if (s == endp && *endp == '\0')
    {
        *val = 0;
        return 0;
    }
    else
    {
        return -1;
    }
}

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

void usage(char *argv0)
{
    fprintf(stderr, "Usage: %s thread_count array_size\n\n", argv0);
    exit(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 ni;
    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 */
        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 = 0;
    char newline = '\n';
    for (i = 0; i < x_chars; i++) {
        /* Get the current color, then output the point */
        set_color(fd, color_val[i]);
        if (write(fd, &point, 1) < 1) {
            perror("compute_and_output_mandel_line: write point");
            exit(1);
        }
    }

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

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

    compute_mandel_line(line, color_val);
    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) // allocate memory which is accessed by all processes
{
    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; // number of allocated pages - sysconf(size of page in bytes) = get configuration info at run time - function that provides method in application to determine the system limit on memory

    /* Create a shared, anonymous mapping for this number of pages */
    /*
     * 1000
     */
    addr = mmap(NULL, pages, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0); // create shared mapping of pages

    return addr;
}

void destroy_shared_memory_area(void *addr, unsigned int numbytes) //delete shared mapping
{
    int pages;

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

    /*
     * 1000
     */
    pages = (numbytes + 1) / sysconf(_SC_PAGE_SIZE) + 1;

    if (munmap(addr, pages * sysconf(_SC_PAGE_SIZE)) == -1) { // check if deallocation of buffer is valid
        perror("destroy_shared_memory_area: munmap failed");
        exit(1);
    }
}

/* Start function for each thread */
void *thread_execute(int line, int procnt) // for each thread execute process and compute color for each line
{
    int line_num;
    int color_val[x_chars];
    for (line_num = line; line_num < x_chars; line_num += procnt) {
        compute_mandel_line(line_num, color_val);
        sem_wait(&sem_array[line]); // locks the semaphore pointed to by sem_array
        output_mandel_line(line_num, color_val);
        sem_post(&sem_array[line_num * procnt]); //unlock semaphore pointed to by sem_array
    }

    return NULL;
}

int main(int argc, char *argv[])
{
    int t, procnt, status;

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

    signal(SIGINT, handler); // signal handler function for a signal - returns previous value of signal - when press Ctrl+C terminate the process
    if (argc != 2) usage(argv[0]);
    if (safe_atoi(argv[1], &procnt) < 0 || procnt <= 0) {
        fprintf(stderr, "%s: Invalid thread count\n", argv[0]);
        exit(1);
    }

    // CREATE SEMAPHORES (ONE FOR EACH PROCESS)
    sem_array = create_shared_memory_area(procnt * sizeof(sem_t)); // create shared area in memory for each semaphore with sem size
    for (t = 0; t < procnt; t++) // initialize semaphores at the addresses pointed to by sem_array[]; shared between the threads of a process located at some visible to all threads access
        sem_init(&sem_array[t], 0, 1); // sem_init has pshared=1 so it can be shared in mmap

    // INITIALIZE FIRST SEMAPHORE
    sem_post(&sem_array[0]); // unlock first semaphore

    // CREATE FORKS (PROCESSES)
    pid_t child_pid[procnt];
    for (t = 0; t < procnt; t++) // create process of all children
        pid_t b = fork();
        if (b == 0) {
            child_pid[t] = t;
            if (b == 0) {
                perror("error with creation of child");
                exit(1);
            }
        }
        if (b == -1) {
            thread_execute(t, procnt); // execute thread for every process
        }
    }
}

```

```

    }
    exit(0);
}

for (i=0; i<procnt; i++){
    child_pid[i] = wait(&status); // pauses until execution of background process has ended
}

for (i=0; i<procnt; i++){
    sem_destroy(&sem_array[i]); // destroy all semaphores pointed to by sem_array
}

destroy_shared_memory_area(sem_array, procnt * sizeof(sem_t));
reset_xterm_color( );
return 0;
}

```

Μια ενδεικτική έξοδος του προγράμματος είναι η ακόλουθη:



1.2.2 Υλοποίηση χωρίς semaphores

Ερωτήσεις:

1. Το αποτέλεσμα δεν τυπώνεται απευθείας στην έξοδο, αλλά γράφεται στην αντίστοιχη θέση ενός διαμοιραζόμενου buffer και γι' αυτό χρειάζεται συγχρονισμός. Αυτός επιτυγχάνεται με τη χρήση του διαμοιραζόμενου buffer, διαστάσεων (θέσεων) μεγαλύτερων ή ίσων από το πλήθος των γραμμών που πρέπει να τυπωθούν στο τέλος.

Αν ο buffer έχει διαστάσεις $NPROCS \times x_chars$, δηλαδή αν είναι μικρότερος από αυτό που χρειαζόμαστε τότε θα πρέπει να συγχρονίσουμε τις διεργασίες. Αν δεν τις συγχρονίσουμε, τότε υπάρχει κίνδυνος να γίνει overwrite στον buffer όταν αυτός γεμίσει.

```

1 //
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/stat.h>
7 #include <fcntl.h>
8 #include <pthread.h>
9 #include <signal.h>
10 #include <csignal.h>
11 #include <ctype.h>
12 #include <sys/mman.h>
13 #include <sys/time.h>
14 #include <sys/stat.h>
15 #include <sys/poll.h>
16 #include <stdint.h>
17 #include "mandel.c.h"
18 #include "help.h"
19
20 #define RED "\033[31m"
21 #define RESET "\033[0m"
22
23 #define MANDEL_MAX_ITERATION 100000
24
25 #define parcar_pthread(ret, mag) \
26     do { ret = ret; parcar(mag); } while(0)
27
28 void handler(sig)
29 int sig;
30 {
31     reset_xterm_color();
32     printf(" ");
33     exit(1);
34 }
35
36 char *heap_private_buf;
37 char *heap_shared_buf;
38 //sem_t s;
39 int **buff;
40
41 //xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
42 // compile file parcar.c +
43 // xxxxxxxxxxxxxxxxxxxxxxxxxxxxx/
44
45 /*
46  * Output at the terminal is x_chars wide by y_chars long
47  */
48 int y_chars = 0;
49 int x_chars = 0;
50
51 /*
52  * The part of the complex plane to be drawn:
53  * upper left corner is (xmin, ymax), lower right corner is (xmax, ymin)
54  */
55 double xmin = -1.0, xmax = 1.0;
56 double ymin = -1.0, ymax = 1.0;
57
58 /*
59  * Every character in the final output is
60  * xstep x ystep units wide on the complex plane.
61  */
62 double xstep;
63 double ystep;

```

11

```

    char point = 0;
    char newline = '\n';

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

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

    compute_nandel_line(line, color_val);
    output_nandel_line(fd, color_val);
}

/*
 * Create a shared memory area, usable by all descendants of th 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 */
    /* 100MB */
    addr = mmap(0, pages, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0); // create shared mapping for pages

    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; // create page that we have to allocate

if (munmap(addr, pages * sysconf(_SC_PAGE_SIZE)) == -1) { // deallocate buffers - destroy mapping
    perror("destroy_shared_memory_area: munmap failed");
    exit(1);
}

/* Start function for each thread */
void *thread_execute(int line, int procnt)
{
    int line_num;
    for (line_num = line; line_num < x_chars; line_num += procnt) {
        compute_nandel_line(line_num, buff[line_num]);
    }

    return NULL;
}

int main(int argc, char *argv[])
{
    int t, procnt, status;

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

    signal(SIGINT, handler); // returns previous value of signal handler on failure
    if (argc != 3) { usage(argv[0]); }
    if (safe_atoi(argv[1], &procnt) <= 0 || procnt >= 100) {
        fprintf(stderr, "%s: Invalid value for x_num\n", argv[0]);
        exit(1);
    }

    // CREATE SHARED MEMORY SPACE FOR X * Y GRID
    buff = create_shared_memory_area(y_chars * sizeof(int));
    for (l = 0; l < y_chars; l++) { // Create shared memory for the line
        buff[l] = create_shared_memory_area(x_chars * sizeof(int));
    }

    // CREATE FORKS (PROCESSES)
    pid_t child_pid[procnt];
    for (l = 0; l < procnt; l++) {
        pid_t b = fork(0);
        child_pid[l] = b;
        if (b == 0) {
            perror("Error with creation of child");
            exit(1);
        }
    }

    if (b == 0) {
        thread_execute(l, procnt);
        exit(1);
    }

    // WAIT FOR CHILDREN PROCESSES TO END
    for (l = 0; l < procnt; l++) {
        child_pid[l] = wait(&status);
    }

    for (l = 0; l < y_chars; l++) {
        output_nandel_line(l, buff[l]);
    }

    sem_t *sem_array;
    destroy_shared_memory_area(sem_array, procnt * sizeof(sem_t));

    reset_xterm_color();
    return 0;
}

```


Μια ενδεικτική έξοδος του προγράμματος είναι η ακόλουθη:

