Λειτουργικά Συστήματα

4η Σειρά Ασκήσεων

Ακαδημαϊκό έτος 2020-2021

Μέρος Ι

1. Παρακάτω φαίνεται ο χάρτης μνήμης της παρούσας διεργασίας. Χρησιμοποιούμε την βοηθητική συνάρτηση show_maps(), η οποία τυπώνει τον χάρτη μνήμης της τρέχουσας διεργασίας. Κάθε σειρά αντιπροσωπεύει μια απεικόνιση αυτής της διεργασίας. Το πρώτο πεδίο είναι το εύρος αυτής της απεικόνισης στην εικονική μνήμη (VMA), το δεύτερο υποδεικνύει τα δικαιώματα του χρήστη σε αυτήν την απεικόνιση, ενώ στην τελευταία στήλη, δίνεται η αντιστοιχία σε πραγματική (φυσική) θέση μνήμης, αν αυτή υπάρχει.

```
Step 1: Print the virtual address space map of this process [19193].


Virtual Memory Map of process [19193]:
00400000-00402000 r-xp 00000000 00:21 20197755                           /home/oslab/oslaba46/asg4/mmap/mmap
00601000-00602000 rw-p 00001000 00:21 20197755                           /home/oslab/oslaba46/asg4/mmap/mmap
00acf000-00af0000 rw-p 00000000 00:00 0                                  [heap]
7fd545996000-7fd545b37000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fd545b37000-7fd545d37000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fd545d37000-7fd545d3b000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fd545d3b000-7fd545d3d000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fd545d3d000-7fd545d41000 rw-p 00000000 00:00 0
7fd545d41000-7fd545d62000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fd545f54000-7fd545f57000 rw-p 00000000 00:00 0
7fd545f5c000-7fd545f61000 rw-p 00000000 00:00 0
7fd545f61000-7fd545f62000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fd545f62000-7fd545f63000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fd545f63000-7fd545f64000 rw-p 00000000 00:00 0
7ffd33687000-7ffd336a8000 rw-p 00000000 00:00 0                          [stack]
7ffd337b7000-7ffd337ba000 r--p 00000000 00:00 0                          [vvar]
7ffd337ba000-7ffd337bc000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

2. Με την κλήση συστήματος mmap() δεσμεύτηκε buffer (προσωρινή μνήμη) μεγέθους μιας σελίδας (page) και έπειτα τυπώσαμε παρακάτω τον χάρτη, με τον τρόπο που αναλύθηκε παραπάνω. Εντοπίζουμε στο χάρτη μνήμης τον εξής χώρο διευθύνσεων που έχουμε δεσμεύσει για την παραπάνω διαδικασία: 7fcfb2c34000-7fcfb2c3a000 rw-p 00000000 00:00 0.

```
Virtual Memory Map of process [21138]:
00400000-00402000 r-xp 00000000 00:21 20197666                           /home/oslab/oslaba46/asg4/mmap/mmap
00601000-00602000 rw-p 00001000 00:21 20197666                           /home/oslab/oslaba46/asg4/mmap/mmap
0121e000-0123f000 rw-p 00000000 00:00 0                                  [heap]
7fcfb266f000-7fcfb2810000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fcfb2810000-7fcfb2a10000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fcfb2a10000-7fcfb2a14000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fcfb2a14000-7fcfb2a16000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fcfb2a16000-7fcfb2a1a000 rw-p 00000000 00:00 0
7fcfb2a1a000-7fcfb2a3b000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fcfb2c2d000-7fcfb2c30000 rw-p 00000000 00:00 0
7fcfb2c34000-7fcfb2c3a000 rw-p 00000000 00:00 0
7fcfb2c3a000-7fcfb2c3b000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fcfb2c3b000-7fcfb2c3c000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fcfb2c3c000-7fcfb2c3d000 rw-p 00000000 00:00 0
7ffffb696000-7ffffb6b7000 rw-p 00000000 00:00 0                          [stack]
7ffffb739000-7ffffb73c000 r--p 00000000 00:00 0                          [vvar]
7ffffb73c000-7ffffb73e000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

3. Χρησιμοποιώντας την συνάρτηση get_physical_address(), τυπώνουμε την φυσική διεύθυνση μνήμης την οποία απεικονίζει ο buffer. Παρατηρούμε ότι δεν έχει παραχθεί κάποιος χώρος στην φυσική μνήμη, για την αποθήκευση του buffer. Αυτό συμβαίνει, επειδή ο buffer δεν έχει αρχικοποιηθεί, επομένως δεν υπάρχει λόγος να δεσμευτεί χώρος από τη φυσική μνήμη σε αυτό το στάδιο.

```
VA[0x7f180b5f8000] is not mapped; no physical memory allocated.
Physical address 0
```

4. Από την στιγμή που έχουμε αρχικοποιήσει, τον buffer, υπάρχει αντιστοίχιση στην φυσική μνήμη. Αυτό είναι άλλωστε και λογικά ακόλουθο, καθώς δεν είναι δυνατόν να υπάρχουν δεδομένα αποθηκευμένα, αν δεν υπάρχει κάποια φυσική αποθήκευση.

```
7f180b5f7000-7f180b5fd000 rw-p 00000000 00:00
Physical address 377569280
```

5. Ανοίγουμε το αρχείο file.txt με την κλήση συστήματος open και με δικαιώματα διαβασματος και στην συνέχεια χρησιμοποιούμε την συνάρτηση mmap. Εκτυπώνουμε το περιεχόμενο του αρχείου και έπειτα δίνονται οι πληροφορίες της εικονικής μνήμης που αφορά στο αρχείο αυτό, καθώς και η φυσική του διεύθυνση (θέση στη φυσική μνήμη). Από τα παραπάνω, επιβεβαιώνεται και η σωστή ανάθεση δικαιώματος ανάγνωσης, όπως αυτά φαίνονται παρακάτω:

```
Hello everyone!
7f180b5f7000-7f180b5f8000 r--p 00000000 00:21 20197756                   /home/oslab/oslaba46/asg4/mmap/file.txt
Physical address is 212447232
```

6. Παρατηρώντας τον χάρτη μνήμης, βλέπουμε στο 2ο πεδίο πως πρόκειται για ένα διαμοιραζόμενο αρχείο λόγω του s με δικαιώματα ανάγνωσης και εγγραφής. Στο Linux, ένας κοινόχρηστος ανώνυμος χάρτης βασίζεται στην πραγματικότητα σε αρχεία. Ο πυρήνας δημιουργεί ένα αρχείο σε tmpfs (μια παρουσία / dev / zero (είναι ένα αρχείο, το οποίο παρέχει τόσα μηδενικά όσοι και οι χαρακτήρες που διαβάζονται)). Το αρχείο διαγράφεται αμέσως, οπότε δεν είναι δυνατή η πρόσβαση του από άλλες διαδικασίες, εκτός εάν κληρονομήσουν τον χάρτη μέσω fork().

```
Virtual Memory Map of process [10563]:
00400000-00403000 r-xp 00000000 00:21 20197655                   /home/oslab/oslaba46/asg4/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 20197655                   /home/oslab/oslaba46/asg4/mmap/mmap
00ce2000-00d03000 rw-p 00000000 00:00 0                          [heap]
7f180b032000-7f180b1d3000 r-xp 00000000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f180b1d3000-7f180b3d3000 ---p 001a1000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f180b3d3000-7f180b3d7000 r--p 001a1000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f180b3d7000-7f180b3d9000 rw-p 001a5000 08:01 6032227            /lib/x86_64-linux-gnu/libc-2.19.so
7f180b3d9000-7f180b3dd000 rw-p 00000000 00:00 0
7f180b3dd000-7f180b3fe000 r-xp 00000000 08:01 6032224            /lib/x86_64-linux-gnu/ld-2.19.so
7f180b5f0000-7f180b5f3000 rw-p 00000000 00:00 0
7f180b5f5000-7f180b5f6000 rw-p 00000000 00:00 0
7f180b5f6000-7f180b5f7000 rw-s 00000000 00:04 1398213            /dev/zero (deleted)
7f180b5f7000-7f180b5f8000 r--p 00000000 00:21 20197756           /home/oslab/oslaba46/asg4/mmap/file.txt
7f180b5f8000-7f180b5fd000 rw-p 00000000 00:00 0
7f180b5fd000-7f180b5fe000 r--p 00020000 08:01 6032224            /lib/x86_64-linux-gnu/ld-2.19.so
7f180b5fe000-7f180b5ff000 rw-p 00021000 08:01 6032224            /lib/x86_64-linux-gnu/ld-2.19.so
7f180b5ff000-7f180b600000 rw-p 00000000 00:00 0
7ffcc161f000-7ffcc1640000 rw-p 00000000 00:00 0                  [stack]
7ffcc172e000-7ffcc1731000 r--p 00000000 00:00 0                  [vvar]
7ffcc1731000-7ffcc1733000 r-xp 00000000 00:00 0                  [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0          [vsyscall]
-----------------------------------------------------

7f180b5f6000-7f180b5f7000 rw-s 00000000 00:04 1398213            /dev/zero (deleted)
Physical Address is 2033410048
```

7. Παρατηρούμε ότι οι χάρτες μνήμης των διεργασιών του πατέρα και του παιδιού είναι πανομοιότυπες. Αυτό σχετίζεται με τις ιδιότητες της συνάρτησης fork(), καθώς στο παιδί κληρονομούνται όλα τα ανοιχτά αρχεία και το αντίγραφο της μνήμης της αρχικής διεργασίας κλπ μέχρι το σημείο εκείνο πριν γίνει η διακλάδωση. Στις εικονικές μνήμες, δεν γίνεται εκ νέου αντιγραφή και αποθήκευση όλων αυτών των αρχείων και μεταβλητών. Αντιθέτως και οι δύο διεργασίες, δείχνουν στο ίδιο μέρος της εικονικής αλλά και φυσικής μνήμης, αποφεύγοντας έτσι, να

υπάρχουν δεδομένα πολλαπλές φορές στην φυσική μνήμη. Άρα μετά την δημιουργία της νέας διεργασίας, μοιράζονται σελίδες ο πατέρας με το παιδί, με διαφορετικά δικαιώματα πρόσβασης.

```
Virtual Memory Map of process [10563]:
00400000-00403000 r-xp 00000000 00:21 20197655                           /home/oslab/oslaba46/asg4/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 20197655                           /home/oslab/oslaba46/asg4/mmap/mmap
00ce2000-00d03000 rw-p 00000000 00:00 0                                  [heap]
7f180b032000-7f180b1d3000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f180b1d3000-7f180b3d3000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f180b3d3000-7f180b3d7000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f180b3d7000-7f180b3d9000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f180b3d9000-7f180b3dd000 rw-p 00000000 00:00 0
7f180b3dd000-7f180b3fe000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f180b5f0000-7f180b5f3000 rw-p 00000000 00:00 0
7f180b5f5000-7f180b5f6000 rw-p 00000000 00:00 0
7f180b5f6000-7f180b5f7000 rw-s 00000000 00:04 1398213                    /dev/zero (deleted)
7f180b5f7000-7f180b5f8000 r--p 00000000 00:21 20197756                   /home/oslab/oslaba46/asg4/mmap/file.txt
7f180b5f8000-7f180b5fd000 rw-p 00000000 00:00 0
7f180b5fd000-7f180b5fe000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f180b5fe000-7f180b5ff000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f180b5ff000-7f180b600000 rw-p 00000000 00:00 0
7ffcc161f000-7ffcc1640000 rw-p 00000000 00:00 0                          [stack]
7ffcc172e000-7ffcc1731000 r--p 00000000 00:00 0                          [vvar]
7ffcc1731000-7ffcc1733000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

```
Virtual Memory Map of process [25770]:
00400000-00403000 r-xp 00000000 00:21 20197655                           /home/oslab/oslaba46/asg4/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 20197655                           /home/oslab/oslaba46/asg4/mmap/mmap
00ce2000-00d03000 rw-p 00000000 00:00 0                                  [heap]
7f180b032000-7f180b1d3000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f180b1d3000-7f180b3d3000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f180b3d3000-7f180b3d7000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f180b3d7000-7f180b3d9000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f180b3d9000-7f180b3dd000 rw-p 00000000 00:00 0
7f180b3dd000-7f180b3fe000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f180b5f0000-7f180b5f3000 rw-p 00000000 00:00 0
7f180b5f5000-7f180b5f6000 rw-p 00000000 00:00 0
7f180b5f6000-7f180b5f7000 rw-s 00000000 00:04 1398213                    /dev/zero (deleted)
7f180b5f7000-7f180b5f8000 r--p 00000000 00:21 20197756                   /home/oslab/oslaba46/asg4/mmap/file.txt
7f180b5f8000-7f180b5fd000 rw-p 00000000 00:00 0
7f180b5fd000-7f180b5fe000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f180b5fe000-7f180b5ff000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f180b5ff000-7f180b600000 rw-p 00000000 00:00 0
7ffcc161f000-7ffcc1640000 rw-p 00000000 00:00 0                          [stack]
7ffcc172e000-7ffcc1731000 r--p 00000000 00:00 0                          [vvar]
7ffcc1731000-7ffcc1733000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

8. Δείχνουν και οι δύο διεργασίες στην ίδια θέση της φυσικής μνήμης, κάτι αναμενόμενο, από τα παραπάνω. Εδώ παρατηρούμε, ότι στο στάδιο αυτό, δεν εμφανίζονται διαφορές μεταξύ ιδιωτικό και διαμοιριζόμενου “buffer”, όμως αυτές θα γίνουν εμφανείς παρακάτω.

```
7f180b5f8000-7f180b5fd000 rw-p 00000000 00:00 0
7f180b5f8000-7f180b5fd000 rw-p 00000000 00:00 0
Physical Memory of Child Process pa 377569280
Physical Memory of Parent Process pa 377569280
```

9. Αν τροποποιήσουμε το περιεχόμενο του private buffer, θα παρατηρήσουμε ότι, η φυσική διεύθυνση του παιδιού αλλάζει. Αυτό συμβαίνει, καθώς όταν η διεργασία παιδί, πάει να γράψει σε κάποια σελίδα, γίνεται εύρεση νέου πλαισίου, αντιγραφή και αλλαγή του πίνακα σελίδων. Ενώ λοιπόν οι εικονικές διευθύνσεις είναι οι ίδιες, η κάθε διεργασία, γράφει στην "δική της" μνήμη.

```
7f180b5f8000-7f180b5fd000 rw-p 00000000 00:00 0
Physical Memory of Parent Process pa (after child writes)377569280
7f180b5f8000-7f180b5fd000 rw-p 00000000 00:00 0
Physical Memory of Child Process pa (after write) 3240271872
```

10. Αντιθέτως, όταν γίνεται εγγραφή στον κοινό buffer, δεν παρατηρείται κάτι παρόμοιο με το (9), καθώς και οι δύο διεργασίες, έχουν δικαίωμα για εγγραφή στο ίδιο σημείο. Έτσι δεν γίνεται καμία αλλαγή στις υπάρχουσες διευθύνσεις, απλά έχουν και οι δύο διεργασίες πρόσβαση σε αυτές για διάβασμα και γράψιμο.

```
7f180b5f6000-7f180b5f7000 rw-s 00000000 00:04 1398213                    /dev/zero (deleted)
Physical Memory of Parent Process shared_file (after child write) 2033410048
7f180b5f6000-7f180b5f7000 rw-s 00000000 00:04 1398213                    /dev/zero (deleted)
Physical Memory of Child Process shared_file (after write) 2033410048
```

11. Μέσω της συνάρτησης mprotect, αλλάζουμε τα δικαιώματα της διεργασίας παιδί, κάτι το οποίο φαίνεται και στις παρακάτω πληροφορίες. Το δικαίωμα του παιδιού δηλαδή, έχει περιοριστεί σε read.

```
7f180b5f6000-7f180b5f7000 rw-s 00000000 00:04 1398213                    /dev/zero (deleted)
Physical Memory of Parent Process shared_file (after child writing access is RESTRICTED) 2033410048
7f180b5f6000-7f180b5f7000 r--s 00000000 00:04 1398213                    /dev/zero (deleted)
Physical Memory of Child Process AFTER RESTRICTION (no writing) 2033410048
```

12. Στο τελευταίο ζήτημα έγινε η αποδέσμευση που ζητείται.

Κώδικάς:

```c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```c
#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;

uint64_t *pa;
char *file_shared_buf1;
/*
 * Child process' entry point.
 */
void child(void)
{
      //uint64_t pa;
      /*
      * Step 7 - Child
      */
      if (0 != raise(SIGSTOP))
                  die("raise(SIGSTOP)");
      /*
      * TODO: Write your code here to complete child's part of Step 7.
      */
      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.
        */

        uint64_t addr5;
        addr5 = get_physical_address((uint64_t)pa);
        show_va_info((uint64_t)pa);
        printf("Physical Address of Child Process pa %ld",addr5);

        printf("\n");
        /*
        * Step 9 - Child
        */
        if (0 != raise(SIGSTOP))
                    die("raise(SIGSTOP)");
        /*
        * TODO: Write your code here to complete child's part of Step 9.
        */
        memset(pa,1,buffer_size);
        addr5 = get_physical_address((uint64_t)pa);
        show_va_info((uint64_t)pa);
        printf("Physical Address of Child Process pa (after write)
%ld\n",addr5);

        /*
        * Step 10 - Child
        */
        if (0 != raise(SIGSTOP))
                    die("raise(SIGSTOP)");
        /*
        * TODO: Write your code here to complete child's part of Step 10.
        */
        uint64_t addr7;
        memset(file_shared_buf1,2,buffer_size);
        addr7 = get_physical_address((uint64_t)file_shared_buf1);
        show_va_info((uint64_t)file_shared_buf1);
        printf("Physical Address of Child Process shared_file (after write)
%ld\n",addr7);

        /*
        * Step 11 - Child

        */
        if (0 != raise(SIGSTOP))
```

```
                die("raise(SIGSTOP)");
        /*
        * TODO: Write your code here to complete child's part of Step 11.
        */
        mprotect((uint64_t*)file_shared_buf1,buffer_size,PROT_READ);

        uint64_t addr9,addr10;
        addr10 = get_physical_address((uint64_t)file_shared_buf1);
        show_va_info((uint64_t)file_shared_buf1);
        printf("Physical Address of Child Process AFTER RESTRICTION (no
writing) %ld\n",addr10);

        memset(file_shared_buf1,3,buffer_size);

        addr9 = get_physical_address((uint64_t)file_shared_buf1);
        show_va_info((uint64_t)file_shared_buf1);
        printf("Physical Address of Child Process AFTER RESTRICTION (after
writing)%ld\n",addr9);
        show_maps();
        /*
        * Step 12 - Child
        */
        if (0 != raise(SIGSTOP))
                die("raise(SIGSTOP)");

        /*
        * TODO: Write your code here to complete child's part of Step 12.
        */
        printf("deleting from child \n");
        munmap((uint64_t*)file_shared_buf1,buffer_size);
        munmap((uint64_t*)pa,buffer_size);


}

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

 /* Wait for the child to raise its first SIGSTOP. */
```

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

        /*
        * Step 7: Print parent's and child's maps. What do you see?
        * Step 7 - Parent
        */
        printf(RED "\nStep 7: Print parent's and child's map.\n" RESET);
        press_enter();

        show_maps();

        /*
        * TODO: Write your code here to complete parent's part of Step 7.
        */

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



        /*
        * Step 8: Get the physical memory address for heap_private_buf.
        * Step 8 - Parent
        */
        printf(RED "\nStep 8: Find the physical address of the private heap "
                    "buffer (main) for both the parent and the child.\n"
RESET);
        press_enter();

        /*
        * TODO: Write your code here to complete parent's part of Step 8.
        */

        //press_enter();
        uint64_t addr6;
        addr6 = get_physical_address((uint64_t)pa);
        show_va_info((uint64_t)pa);
        printf("Physical Address of Parent Process pa %ld",addr6);

        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.
        */
        addr6 = get_physical_address((uint64_t)pa);
        show_va_info((uint64_t)pa);
        printf("Physical Address of Parent Process pa (after child
writes)%ld\n",addr6);

        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.
        */

        uint64_t addr8;
        addr8 = get_physical_address((uint64_t)file_shared_buf1);
        show_va_info((uint64_t)file_shared_buf1);
        printf("Physical Address of Parent Process shared_file (after child
```

```
write) %ld\n",addr8);

      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

      * 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();

      uint64_t addr10;
      addr10 = get_physical_address((uint64_t)file_shared_buf1);
      show_va_info((uint64_t)file_shared_buf1);
      printf("Physical Address of Parent Process shared_file (after child
writing access is RESTRICTED) %ld\n",addr10);

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

      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
      */
      printf(RED "\nStep 12: Emptying buffers. \n" RESET);
      press_enter();

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

```
        munmap((uint64_t*)file_shared_buf1,buffer_size);
        munmap((uint64_t*)pa,buffer_size);

 /*         uint64_t addr13,addr14;
        addr13 = get_physical_address((uint64_t)pa);
        show_va_info((uint64_t)pa);
        printf("Physical Memory(1) of Parent, emptied %ld\n",addr13);
        addr14 = get_physical_address((uint64_t)file_shared_buf1);
        show_va_info((uint64_t)file_shared_buf1);
        printf("Physical Memory(2) of Parent, emptied %ld\n",addr14);


        printf(RED "\nPress 'Enter' to exit. \n" RESET);
        press_enter();

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

int main(void)
{
        pid_t mypid, p;
        int fd = -1;

        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.

	*/


	pa =
mmap(NULL,buffer_size,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,fd,0);
	if (pa == MAP_FAILED){
		   printf("Map failed");
	}

	show_maps();

	show_va_info((uint64_t)pa);

	/*
	* 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.
	*/
	uint64_t addr1 =  get_physical_address((uint64_t)pa);
	printf("Physical address is %ld",addr1);
	/* 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(pa,0,buffer_size);
	show_va_info((uint64_t)pa);
	uint64_t newaddr;
	newaddr = get_physical_address((uint64_t)pa);
	printf("Physical address is %ld",newaddr);
	/*
	* 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.
	*/
	int fd1 = open("file.txt",O_RDONLY,S_IRUSR | S_IWUSR);
	struct stat sb;

	if (fstat(fd1,&sb) == -1){
		perror("Coudn't get file size");
	}
	char *file_in_memory =
mmap(NULL,sb.st_size,PROT_READ,MAP_PRIVATE,fd1,0);
	int i ;
	for (i=0;i<sb.st_size;i++){
		printf("%c",file_in_memory[i]);
	}
	show_maps();
	show_va_info((uint64_t)file_in_memory);
	uint64_t addr3;
	addr3 = get_physical_address((uint64_t)file_in_memory);
	printf("Physical address is %ld",addr3);
	//munmap(file_in_memory,sb.st_size);
	/*
	* 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.
        */

        file_shared_buf1 = mmap(NULL,buffer_size,PROT_READ |
PROT_WRITE,MAP_SHARED|MAP_ANONYMOUS,fd,0);
        memset(file_shared_buf1,0,buffer_size);
        //unsigned int j;
        /*for (j = 0; j<buffer_size; j++){
                    file_shared_buf1[j] = 'a';
        }*/
        printf("Buffer size is: %ld\n", buffer_size);
        show_maps();
        show_va_info((uint64_t)file_shared_buf1);
        uint64_t addr4;
        addr4 = get_physical_address((uint64_t)file_shared_buf1);
        printf("Physical Address is %ld\n",addr4);

        press_enter();


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

//      wait(NULL);
        if(-1 == close(fd1)) perror("close1");
//      if (-1 == close(fd)) perror("close");

        return 0;
}
```

Μέρος ΙΙ

1.2.1

1. Παρατηρούμε ότι στην συγκεκριμένη διαδικασία, είναι ταχύτερη η υλοποίηση με νήματα παρά με διεργασίες. Δεν υπάρχει κάποια ουσιαστική διαφορά στον τρόπο υλοποίησης των δύο μεθόδων,

ωστόσο στην περίπτωση των νημάτων, υπάρχει εκ γενετής κοινή μνήμη μεταξύ τους, επομένως αρκεί η χρήση των σημαφόρων για τον συγχρονισμό τους, ενώ στην περίπτωση των διεργασιών, οι οποίες δεν μοιράζονται κοινή μνήμη, πρέπει αυτή να επιτευχθεί μέσω της mmap.

real 0m0.354s
user 0m0.992s
sys 0m0.016s

real 0m0.355s
user 0m0.968s
sys 0m0.044s

2. Στην παρούσα μορφή της άσκησης, δηλαδή με χρήση ανώνυμων σημαφόρων, δεν θα ήταν εφικτή η πραγμάτωσή της, χωρίς την ύπαρξη κοινού προγόνου, μεταξύ των διεργασιών. Στα πλαίσια του κοινού προγόνου, θεωρείται και η περίπτωση, όπου το κάθε παιδί, δημιουργεί ακριβώς μια διεργασία, για παράδειγμα, φτάνοντας έτσι στον επιθυμητό αριθμό των n διεργασιών. Σε κάθε περίπτωση όμως, η χρήση εντελώς ανεξάρτητων διεργασιών, θα ήταν εφικτή μόνο με τη χρήση επώνυμων σημαφόρων.

Κώδικας:

```c
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/wait.h>

/*TODO header file for m(un)map*/
#include <sys/mman.h>
#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

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

```c
/*
 * 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;

int*  NPROC;
sem_t *semaphore;

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

void handle_sigint(int sig) {
      reset_xterm_color(1);
      exit(3);
}
```

```
/*
 * 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);
	}
}



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

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

	/*
	* Determine the number of pages needed, round up the requested number
of
	* pages
	*/
	pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;
	/* Create a shared, anonymous mapping for this number of pages */
	/* TODO:
		addr = mmap(...)
```

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

	return (sem_t*)addr;
}

void destroy_shared_memory_area(void *addr, unsigned int numbytes) {
	int pages;

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

	/*
	* Determine the number of pages needed, round up the requested number
of
	* pages
	*/
	pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;

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

void compute_and_output_mandel_line(int fd, int line)
{
	/*
	* A temporary array, used to hold color values for the line being
drawn
	*/
	int i;
	for(i=line;i<y_chars;i+=(*NPROC)){
		int color_val[x_chars];
		compute_mandel_line(i, color_val);
		sem_wait(&semaphore[i % (*NPROC)]);
		output_mandel_line(fd, color_val);
		sem_post(&semaphore[(i+1) % (*NPROC)]);
	}
```

```
	}

int main(int argc, char **argv)
{
	if(argc!=2){
			perror("Too few args\n");
			exit (3);
	}
	NPROC= (int*) malloc(sizeof(int));
	safe_atoi(argv[1],NPROC);

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

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

	//====Creating the Semaphores======
	semaphore=create_shared_memory_area((*NPROC)*sizeof(sem_t));
	int i;
	for (i=0; i<(*NPROC);i++){
		if(i==0) sem_init(&semaphore[i],1,1);
			else sem_init(&semaphore[i], 1, 0);
	}

	//====Creating forks=======
	int line;
	for (line = 0; line < (*NPROC); line++) {
		if(fork()==0){
			compute_and_output_mandel_line(1, line);
			exit(2);
		}
	}

	//====Waiting for children & freeing mem====
	for (line = 0; line <(*NPROC); line++) {
		wait(NULL);
	}
	destroy_shared_memory_area(semaphore,(*NPROC)*sizeof(sem_t));

	reset_xterm_color(1);
```

```
        return 0;
 }
```

1.2.2

1. Σε περίπτωση που γινόταν μια υλοποίηση της μορφής NPROCS x x_chars, θα χρειαζόταν πάλι να ορίσουμε έναν δυσδιάστατο pointer μεγέθους NPROCS x x_chars .

```
shared_buff= (int**) create_shared_memory_area((*NPROC)*sizeof(int*));
int i=0;
        for (i=0;i<(*NPROC);i++){
            shared_buff[i]= (int*)
create_shared_memory_area((x_chars*y_chars/(*NPROC))*sizeof(int));
```

Και στην συνέχεια θα τροποποιούσαμε κατάλληλα την compute and output mande line, ώστε κάθε φορά να διαβάζει από την σωστή γραμμή, και από την σωστή στήλη και το ίδιο και για την εκτύπωση. Έτσι, θα έπρεπε να χρησιμοποιούμε, κάθε φορά, ένα offset, το οποίο θα επηρέαζε το σημείο του κάθε array των διεργασιών, στο οποίο θα γράφαμε, και αντίστροφα θα έπρεπε να έχει συνυπολογιστεί, αυτό κατά την εκτύπωση. Έτσι, για παράδειγμα, στην χρήση τριών (3), διεργασιών, η πρώτη γραμμή του εν λόγω πίνακα θα πρέπει να περιέχει στη σειρά την 1η υπολογισμένη γραμμή mandel (compute_mandel_line) , την 4η, την 7η και ούτω καθεξής.

Κώδικας:

```
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/wait.h>

/*TODO header file for m(un)map*/
#include <sys/mman.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;

int*  NPROC;
int **shared_buff;

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

void handle_sigint(int sig) {
      reset_xterm_color(1);
      exit(3);
}
```

```
/*
 * 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);
	}
}



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

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

	/*
	* Determine the number of pages needed, round up the requested number
of
	* pages
	*/
	pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;

	/* Create a shared, anonymous mapping for this number of pages */
```

```
	/* TODO:
			addr = mmap(...)
	*/
	addr=mmap(NULL,pages,PROT_READ|PROT_WRITE,MAP_SHARED |
MAP_ANONYMOUS,-1,0);

	return (sem_t*)addr;
}

void destroy_shared_memory_area(void *addr, unsigned int numbytes) {
	int pages;

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

	/*
	* Determine the number of pages needed, round up the requested number
of
	* pages
	*/
	pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;

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

void compute_and_output_mandel_line(int fd, int line)
{
	/*
	* A temporary array, used to hold color values for the line being drawn
	*/
	int i;
	for(i=line;i<y_chars;i+=(*NPROC)){
		//int color_val[x_chars];
		compute_mandel_line(i, shared_buff[i]);
		//output_mandel_line(fd, color_val);
	}
	}
```

```
int main(int argc, char **argv)
{
	if(argc!=2){
			perror("Too few args\n");
			exit (3);
	}
	NPROC= (int*) malloc(sizeof(int));
	safe_atoi(argv[1],NPROC);

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

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

	//====Creating the shared file ======
	shared_buff= (int**) create_shared_memory_area(y_chars*sizeof(int*));
	int i=0;
	for (i=0;i<y_chars;i++){
		shared_buff[i]= (int*)
create_shared_memory_area(x_chars*sizeof(int));
	}


	//====Creating forks=======
	int line;
	for (line = 0; line < (*NPROC); line++) {
		if(fork()==0){
			compute_and_output_mandel_line(1, line);
			exit(2);
		}
	}

	//====Waiting for children & freeing mem====
	for (line = 0; line <(*NPROC); line++) {
		wait(NULL);
	}
	int j;
	for (j = 0 ; j < y_chars; j++) {
		output_mandel_line(1, shared_buff[j]);
```

```
        }

        destroy_shared_memory_area(shared_buff,y_chars*sizeof(int*));

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