

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

**4η εργαστηριακή αναφορά**

στο μάθημα **“Λειτουργικά συστήματα”** 6ου Εξαμήνου

Από τους φοιτητές

Άγγελος Γκίκας , ΑΜ: 03118218

Νικήτας Τσίννας, ΑΜ: 03118187

Ομάδα: oslaba14

Ημερομηνία εξέτασης: 08/6/2021

Ημερομηνία παράδοσης: 15/6/2021

**Άσκηση 1.1: Κλήσεις συστήματος και βασικοί μηχανισμοί του ΛΣ για τη διαχείριση της εικονικής μνήμης (Virtual Memory – VM)**

*Source code:*

Mε ‘Bold’ έχουν σημειωθεί οι γραμμές κώδικα που άλλαξαν/προστέθηκαν.

*/\**

*\* mmap.c*

*\**

*\* Examining the virtual memory of processes.*

*\**

*\* Operating Systems course, CSLab, ECE, NTUA*

*\**

*\*/*

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <sys/mman.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <errno.h>

#include <stdint.h>

#include <signal.h>

#include <sys/wait.h>

#include "help.h"

#define RED "\033[31m"

#define RESET "\033[0m"

char \*heap\_private\_buf;

char \*heap\_shared\_buf;

char \*file\_shared\_buf;

uint64\_t buffer\_size;

*/\**

*\* Child process' entry point.*

*\*/*

void child(void)

{

uint64\_t pa;

*/\**

*\* Step 7 - Child*

*\*/*

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

die("raise(SIGSTOP)");

*/\**

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

*\*/*

**printf("Child's VM Map\n");**

**show\_maps();**

*/\**

*\* Step 8 - Child*

*\*/*

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

die("raise(SIGSTOP)");

*/\**

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

*\*/*

**uint64\_t paddr;**

**paddr = get\_physical\_address((uint64\_t)heap\_private\_buf);**

**printf("Physical addr in child: %ld\n", paddr);**

*/\**

*\* Step 9 - Child*

*\*/*

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

die("raise(SIGSTOP)");

*/\**

*\* TODO: Write your code here to complete child's part of Step 9.*

*\*/*

**memset(heap\_private\_buf, 0, buffer\_size);**

**paddr = get\_physical\_address((uint64\_t)heap\_private\_buf);**

**printf("Physical addr in child: %ld\n", paddr);**

*/\**

*\* Step 10 - Child*

*\*/*

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

die("raise(SIGSTOP)");

*/\**

*\* TODO: Write your code here to complete child's part of Step 10.*

*\*/*

**memset(heap\_shared\_buf, 0, buffer\_size);**

**paddr = get\_physical\_address((uint64\_t)heap\_shared\_buf);**

**printf("Physical addr in child: %ld\n", paddr);**

*/\**

*\* Step 11 - Child*

*\*/*

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

die("raise(SIGSTOP)");

*/\**

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

*\*/*

**mprotect(heap\_shared\_buf,buffer\_size,PROT\_READ);**

**printf("Child's VM map:\n");**

**show\_va\_info(heap\_shared\_buf)**;

*/\**

*\* Step 12 - Child*

*\*/*

*/\**

*\* TODO: Write your code here to complete child's part of Step 12.*

*\*/*

*//munmap(heap\_shared\_buf,buffer\_size);*

munmap(heap\_private\_buf,buffer\_size);

*//munmap(file\_shared\_buf,buffer\_size);*

}

*/\**

*\* Parent process' entry point.*

*\*/*

void parent(pid\_t child\_pid)

{

uint64\_t pa;

int status;

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

**if** (-1 == waitpid(child\_pid, &status, WUNTRACED))

die("waitpid");

*/\**

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

*\* Step 7 - Parent*

*\*/*

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

press\_enter();

*/\**

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

*\*/*

**printf("Parent's VM Map\n");**

**show\_maps();**

**if** (-1 == kill(child\_pid, SIGCONT))

die("kill");

**if** (-1 == waitpid(child\_pid, &status, WUNTRACED))

die("waitpid");

*/\**

*\* Step 8: Get the physical memory address for heap\_private\_buf.*

*\* Step 8 - Parent*

*\*/*

printf(RED "**\n**Step 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.*

*\*/*

**uint64\_t paddr;**

**paddr = get\_physical\_address((uint64\_t)heap\_private\_buf);**

**printf("Physical addr in parent: %ld\n", paddr);**

**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 "**\n**Step 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.*

*\*/*

**paddr = get\_physical\_address((uint64\_t)heap\_private\_buf);**

**printf("Physical addr in child: %ld\n", paddr);**

**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 "**\n**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?**\n**" RESET);

press\_enter();

*/\**

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

*\*/*

**paddr = get\_physical\_address((uint64\_t)heap\_shared\_buf);**

**printf("Physical addr in parent: %ld\n", paddr);**

**if** (-1 == kill(child\_pid, SIGCONT))

die("kill");

**if** (-1 == waitpid(child\_pid, &status, WUNTRACED))

die("waitpid");

*/\**

*\* Step 11: Disable writing on the shared buffer for the child*

*\* (hint: mprotect(2)).*

*\* Step 11 - Parent*

*\*/*

printf(RED "**\n**Step 11: Disable writing on the shared buffer for the "

"child. Verify through the maps for the parent and the "

"child.**\n**" RESET);

press\_enter();

*/\**

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

*\*/*

**printf("Parent's VM map:\n");**

**show\_va\_info(heap\_shared\_buf);**

**if** (-1 == kill(child\_pid, SIGCONT))

die("kill");

**if** (-1 == waitpid(child\_pid, &status, 0))

die("waitpid");

*/\**

*\* Step 12: Free all buffers for parent and child.*

*\* Step 12 - Parent*

*\*/*

*/\**

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

*\*/*

munmap(heap\_shared\_buf,buffer\_size);

*//munmap(heap\_private\_buf,buffer\_size);*

*//munmap(file\_shared\_buf,buffer\_size);*

}

int main(void)

{

pid\_t mypid, p;

int fd = -1;

uint64\_t pa;

mypid = getpid();

buffer\_size = 1 \* get\_page\_size();

*/\**

*\* Step 1: Print the virtual address space layout of this process.*

*\*/*

printf(RED "**\n**Step 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 "**\n**Step 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("Error in mapping");**

**return 1;**

}

show\_maps();

show\_va\_info((uint64\_t)pa);

heap\_private\_buf = pa;

*/\**

*\* Step 3: Find the physical address of the first page of your buffer*

*\* in main memory. What do you see?*

*\*/*

printf(RED "**\n**Step 3: Find and print the physical address of the "

"buffer in main memory. What do you see?**\n**" RESET);

press\_enter();

*/\**

*\* TODO: Write your code here to complete Step 3.*

*\*/*

**get\_physical\_address((unsigned long)pa);**

*/\**

*\* Step 4: Write zeros to the buffer and repeat Step 3.*

*\*/*

printf(RED "**\n**Step 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);**

*//paddr =*

printf("Physical Address: %ld**\n**",get\_physical\_address((uint64\_t)pa));

*//printf("Physical addr: %ld", paddr);*

*/\**

*\* Step 5: Use mmap(2) to map file.txt (memory-mapped files) and print*

*\* its content. Use file\_shared\_buf.*

*\*/*

printf(RED "**\n**Step 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.*

*\*/*

***//open file:***

**int fileno = open("file.txt", O\_RDONLY);**

**if (fileno == -1) {**

**printf("error in opening file");**

**return 1;**

**}**

***//find size of file:***

**struct stat buf;**

**fstat(fileno, &buf);**

**int size = buf.st\_size;**

***//allocate VM:***

**file\_shared\_buf = mmap(NULL, size, PROT\_READ, MAP\_PRIVATE, fileno, 0);**

**if (file\_shared\_buf == MAP\_FAILED){**

**printf("Error in mapping");**

**return 1;**

**}**

***//print VM map:***

**show\_maps();**

***//print file content from VM map:***

**int i;**

**for(i=0; i<size; i++){**

**char c;**

**c = file\_shared\_buf[i];**

**printf("%c",c);**

**}**

*/\**

*\* Step 6: Use mmap(2) to allocate a shared buffer of 1 page. Use*

*\* heap\_shared\_buf.*

*\*/*

printf(RED "**\n**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.**\n**" RESET);

press\_enter();

*/\**

*\* TODO: Write your code here to complete Step 6.*

*\*/*

**heap\_shared\_buf = mmap(NULL, buffer\_size, (PROT\_READ|PROT\_WRITE), MAP\_SHARED|MAP\_ANONYMOUS, fd, 0);**

**if (heap\_shared\_buf == MAP\_FAILED){**

**printf("Error in mapping");**

**return 1;**

**}**

**show\_maps();**

**show\_va\_info((uint64\_t)heap\_shared\_buf);**

**memset(heap\_shared\_buf, 0, buffer\_size);**

p = fork();

**if** (p < 0)

die("fork");

**if** (p == 0) {

child();

**return** 0;

}

parent(p);

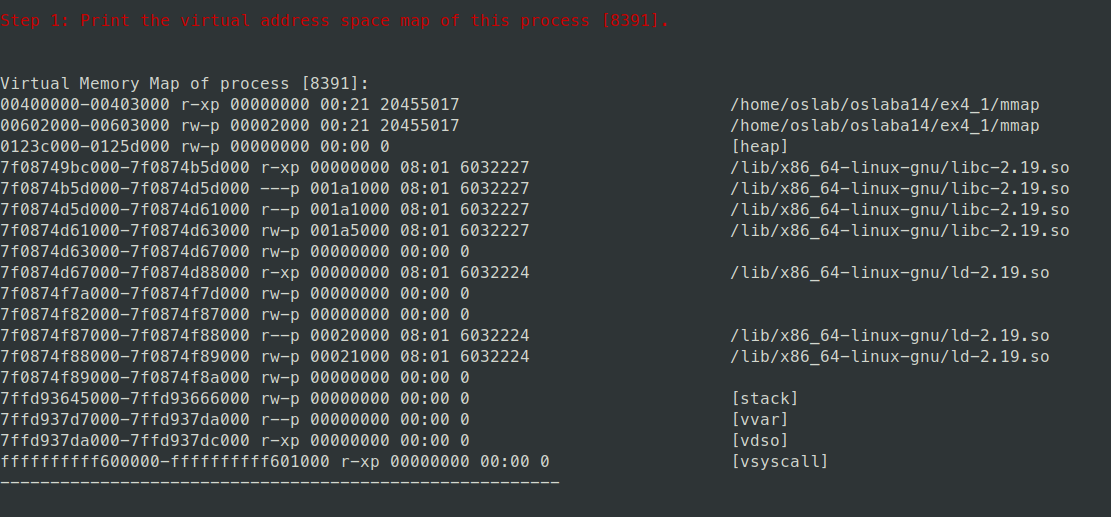
**if** (-1 == close(fd))

perror("close");

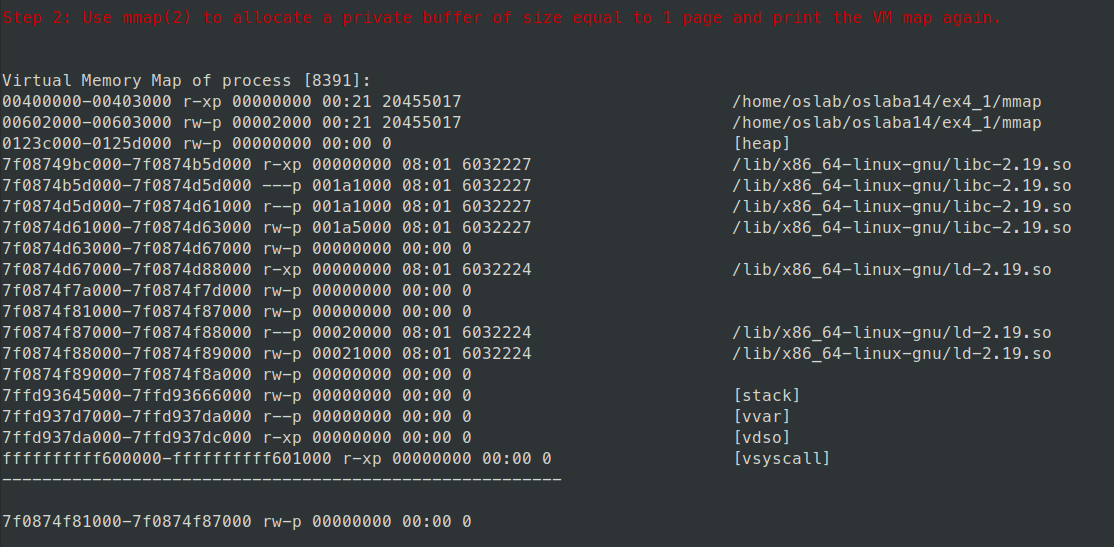
**return** 0;

}

1. Με την χρήση της βοηθητικής εντολής show\_maps() τυπώνεται στο τερματικό ο χάρτης μνήμης της τρέχουσας διεργασίας όπως φαίνεται παρακάτω:



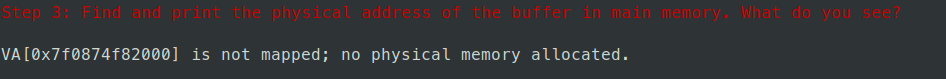
1. Μετά την κλήση συστήματος mmap(), αφού δεσμεύσουμε buffer μεγέθους μίας σελίδας, τυπώνουμε ξανά τον χάρτη της virtual memory:



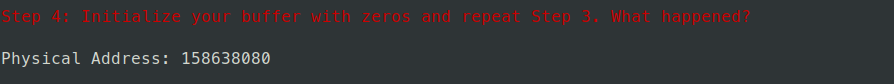
Παρατηρούμε πως έχει δεσμευτεί ο εξής νέος χώρος στην εικονική μνήμη:

**7f0874f81000-7f0874f87000 rw-p 00000000 00:00 0**

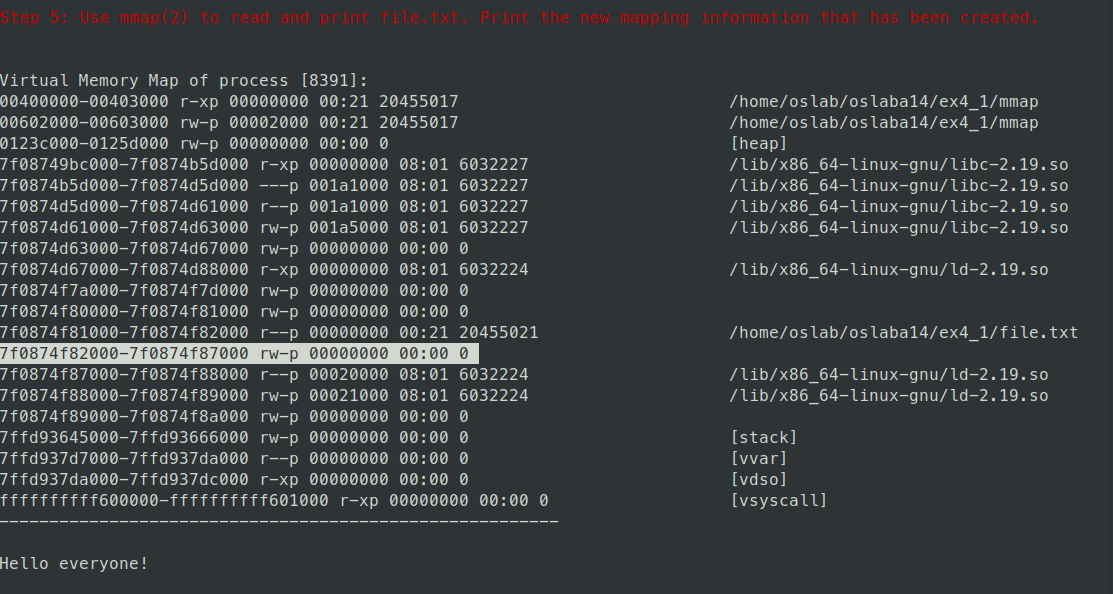
1. Με την χρήση της get\_physical\_address() εκτυπώνουμε την φυσική διεύθυνση μνήμης που αντιστοιχεί η εικονική διεύθυνση του buffer. Παρατηρούμε πως δεν υπάρχει κάποιοα απεικόνηση ακόμα. Αυτό συμβαίνει λόγω της πολτικής του λειτουργικού “**demand paging**”. Πιο συγκεκριμένα, εφόσον δεν έχει προσπελαστεί η εικονική μνήμη του buffer, ο kernel δεν την έχει συνδέσει με κάποια φυσική διεύθυνση.



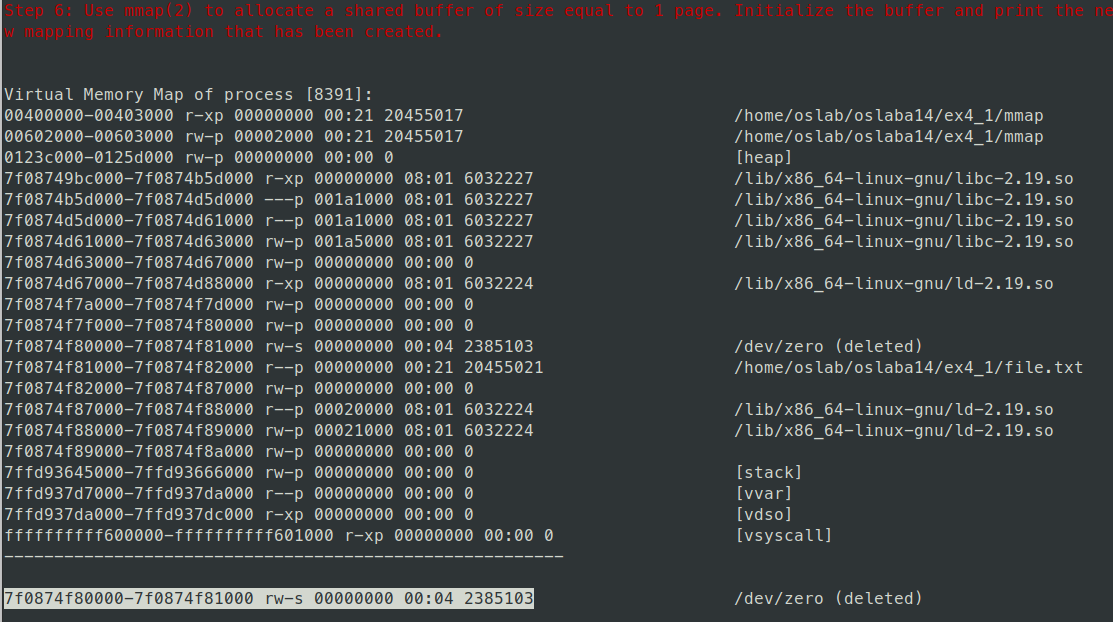
1. Σε αυτό το βήμα, μέσω της κλήσης mmap, γεμίζουμε με μηδενικά τον buffer. Αυτό έχει ως αποτέλεσμα την προσπέλαση του και επομένως την αντιστοίχισή του με κάποια φυσική διεύθυνση μνήμης, όπως φαίνεται παρακάτω:



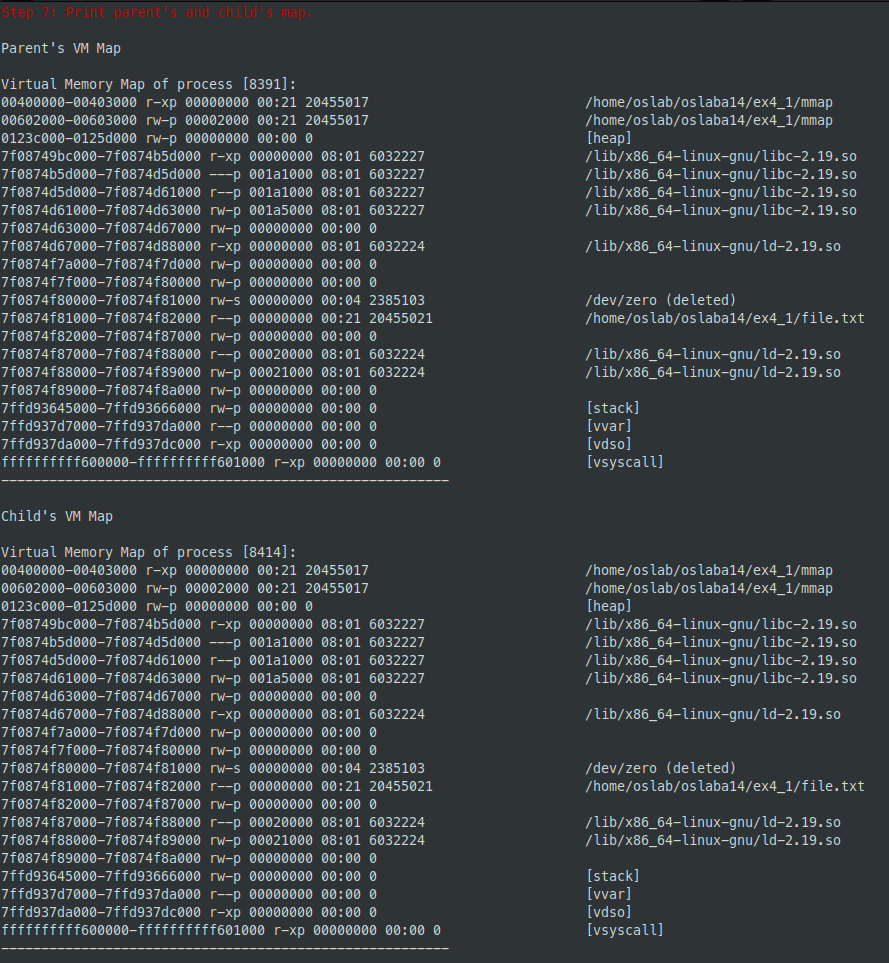
1. Χρησιμοποιώντας την κλήση mmap, απεικονίζουμε το αρχείο file.txt στον εικονικό χώρο της μνήμης της διεργασίας. Παρακάτω φαίνεται υπογραμμισμένη η νέα εικονική διεύθυνση που αντιστοιχεί στο περιεχόμενο του αρχείου. Επίσης, έχει εκτυπωθεί το περιεχόμενο του αρχείου. Για να επιτευχθεί αυτό, υπολογίσαμε πρώτα το μέγεθος του αρχείου για να απεικονίσουμε το αντίστοιχο μέγεθος στην εικονική μνήμη και έπειτα να την προσπελάσουμε σωστά:



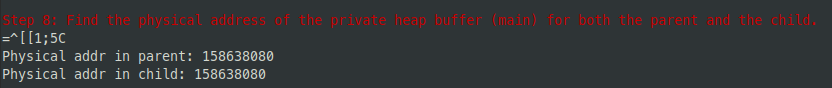
1. Χρησιμοποιώντας την mmap με το flag MAP\_SHARED δημιουργούμε διαμοιραζόμενο μεταξύ των διεργασιών buffer ο οποίος φαίνεται υπογραμμισμένος στην παρακάτω εικόνα. Παρατηρούμε τα αντίστοιχα permissions, όπου το s αντιστοιχεί στο shared:



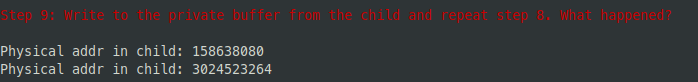
1. Αμέσως μετά την fork, οι πίνακες μνήμης των δύο διεργασιών είναι ίδιοι όπως ήταν αναμενόμενο εφόσον το παιδί δεν έχει αλληλεπιδράσει με τον χώρο μνήμης του ακόμα:



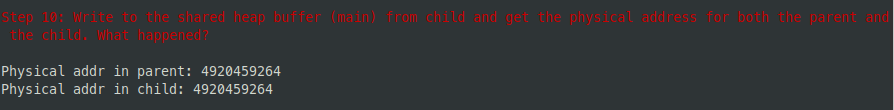
1. Παρατηρούμε ότι οι δύο φυσικές αντιστοιχίσεις μνήμης δεν διαφέρουν διότι ακολουθείται η πολιτική copy-on-write, οπότε αφού το παιδί δεν έχει γράψει στον private buffer, δεν έχει γίνει ακόμη δέσμευση φυσικής μνήμης για το αντίγραφο του buffer που ανήκει στο παιδί.



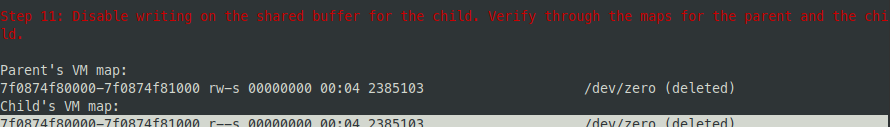
1. Τώρα, λόγω της αιτίας που εξηγήθηκε παραπάνω, οι δύο απεικονίσεις διαφέρουν, εφόσον άλλαξε αυτή του παιδιού:



1. Εφόσον γράφουμε στον shared\_buffer, οι φυσικές απεικονίσεις δεν αλλάζουν για κάθε διεργασία. Σε αντίθεση με προηγουμένως, αυτό συμβαίνει επειδή ο buffer είναι διαμοιραζόμενος:



1. Χρησιμοποιώντας την mprotect() αλλάζουμε τα permissions για την διεργασία παιδί και έτσι του απαγορεύουμε να γράψει σε αυτόν. Παρακάτω φαίνεται στην υπογραμμισμένη γραμμή πως τα δικαιώματα του παιδιού για τον buffer άλλαξαν από rw σε μόνο r, δηλαδή μπορεί πλέον μόνο να διαβάσει αλλά όχι να γράψει στον buffer:



1. Χρησιμοποιώντας την munmap() αποδεσμεύουμε όλους τους buffers (heap\_private\_buf, heap\_shared\_buf, file\_shared\_buf) και για τις δύο διεργασίες.