



Λειτουργικά Συστήματα Υπολογιστών

Κακούρης Δημήτριος (03119019)
Μαρδίκης Κωνσταντίνος (03119867)

4η Εργαστηριακή Άσκηση

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

Κάνουμε make με τη βοήθεια του Makefile που δίνεται και παράγουμε το εκτελέσιμο `mmap`. Προκειμένου να τυπωθούν σωστά σε τοπικό μηχάνημα τα επόμενα υποερωτήματα (όπως τα physical memory addresses) τρέχουμε το εκτελέσιμο με root access δηλαδή, `sudo ./mmap`. Αυτό οφείλεται στο ότι τοπικά τρέχουμε πυρήνα linux 6.2 που φαίνεται να μην αφήνει access σε απλό user σε physical memory addresses.

Έκδοση πυρήνα:

```
brewed@brewed-laptop:~$ uname -r  
6.2.0-20-generic
```

Πηγαίος κώδικας:

```
/*  
 * 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 GREEN "\033[32m" /* Green */
#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)
{

/*
 * Step 7 - Child
 */
if (0 != raise(SIGSTOP))
die("raise(SIGSTOP)");

printf(GREEN "\n Child's map: \n" RESET);
show_maps();

/*
 * Step 8 - Child
 */
if (0 != raise(SIGSTOP))
die("raise(SIGSTOP)");

printf("Physical Address of the private buffer of the child: %ld\n",
get_physical_address((uint64_t)heap_private_buf));

/*
 * Step 9 - Child
 */
if (0 != raise(SIGSTOP))
die("raise(SIGSTOP)");

memset(heap_private_buf, 0, buffer_size);

printf("Physical Address of the private buffer of the child: %ld\n",
get_physical_address((uint64_t)heap_private_buf));

```

```

/*
 * Step 10 - Child
 */
if (0 != raise(SIGSTOP))
die("raise(SIGSTOP)");

memset(heap_shared_buf, 0, buffer_size);
printf("Physical Address of the shared buffer of the child: %ld\n",
get_physical_address((uint64_t) heap_shared_buf));

/*
 * Step 11 - Child
 */
if (0 != raise(SIGSTOP))
die("raise(SIGSTOP)");

mprotect(heap_shared_buf, buffer_size, PROT_READ);

show_va_info((uint64_t) heap_shared_buf);
printf(GREEN "\n Child's map: \n" RESET);
show_maps();

/*
 * Step 12 - Child
 */
munmap(heap_shared_buf, buffer_size);
munmap(heap_private_buf, 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();

printf(GREEN "\n Parent's map: \n" RESET);
show_maps();

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

```

```

die("waitpid");

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

printf("Physical Address of the private buffer of the parent: %ld\n",
get_physical_address((uint64_t) heap_private_buf));

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

/*
 * Step 9: Write to heap_private_buf. What happened?
 * Step 9 - Parent
 */
printf(RED "\nStep 9: Write to the private buffer from the child and "
"repeat step 8. What happened?\n" RESET);
press_enter();

printf("Physical Address of the private buffer of the parent: %ld\n",
get_physical_address((uint64_t)heap_private_buf));

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

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

printf("Physical Address of the shared buffer of the parent: %ld\n",
get_physical_address((uint64_t)heap_shared_buf));

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

```

```

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

show_va_info((uint64_t)heap_shared_buf);
printf(GREEN "\n Parent's map: \n" RESET);
show_maps();

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

/*
 * Step 12: Free all buffers for parent and child.
 * Step 12 - Parent */
munmap(heap_shared_buf,buffer_size);
munmap(heap_private_buf,buffer_size);

}

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();

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();

// Allocate memory for the buffer using mmap
heap_private_buf = mmap(NULL, buffer_size, PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS, -1, 0);
if (heap_private_buf == MAP_FAILED) {
perror("mmap");
return 1;
}

show_va_info((uint64_t)heap_private_buf);
show_maps();

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

printf("Physical Address of the buffer: %ld\n",
get_physical_address((uint64_t)heap_private_buf));

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

if (heap_private_buf == MAP_FAILED) {
perror("mmap");
return 1;
}

memset(heap_private_buf, 0, buffer_size);

printf("Physical Address of the buffer: %ld\n",
get_physical_address((uint64_t)heap_private_buf));

/*
 * 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();

```

```

fd = open ("file.txt", O_RDONLY);
if (fd == -1) {
perror("fopen");
return 1;
}

file_shared_buf = mmap(NULL, buffer_size, PROT_READ, MAP_PRIVATE, fd, 0);
if (file_shared_buf == MAP_FAILED) {
perror("mmap");
return 1;
}
printf("%s\n", file_shared_buf);

show_maps();

/*
 * 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();

heap_shared_buf= mmap(NULL,buffer_size, PROT_READ| PROT_WRITE,MAP_SHARED|
MAP_ANONYMOUS,-1,0);

memset(heap_shared_buf,0,buffer_size);

show_maps();

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

parent(p);

if (-1 == close(fd))
perror("close");
return 0;
}

```

Βήμα 1:

Ο εικονικός χάρτης χάρτης της διεργασίας:

```
Step 1: Print the virtual address space map of this process [33323].
```

```
Virtual Memory Map of process [33323]:
55a7b9165000-55a7b9166000 r--p 00000000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9166000-55a7b9167000 r-xp 00001000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9167000-55a7b9168000 r--p 00002000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9168000-55a7b9169000 r--p 00002000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9169000-55a7b916a000 rw-p 00003000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7ba3ad000-55a7ba3ce000 rw-p 00000000 00:00 0 [heap]
7f9ab4000000-7f9ab4022000 r--p 00000000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab4022000-7f9ab419a000 r-xp 00022000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab419a000-7f9ab41f2000 r--p 0019a000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f2000-7f9ab41f6000 r--p 001f1000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f6000-7f9ab41f8000 rw-p 001f5000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f8000-7f9ab4205000 rw-p 00000000 00:00 0
7f9ab4365000-7f9ab4368000 rw-p 00000000 00:00 0
7f9ab4382000-7f9ab4384000 rw-p 00000000 00:00 0
7f9ab4384000-7f9ab4385000 r--p 00000000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab4385000-7f9ab43ad000 r-xp 00001000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43ad000-7f9ab43b7000 r--p 00029000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b7000-7f9ab43b9000 r--p 00033000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b9000-7f9ab43bb000 rw-p 00035000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffd19901000-7ffd19922000 rw-p 00000000 00:00 0 [stack]
7ffd199e3000-7ffd199e7000 r--p 00000000 00:00 0 [vvar]
7ffd199e7000-7ffd199e9000 r-xp 00000000 00:00 0 [vdso]
fffffffff6000000-fffffffff6010000 --xp 00000000 00:00 0 [vsyscall]
-----
```

Βήμα 2:

Κάνουμε allocate εικονική μνήμη μέσω `mmap()` και χρησιμοποιούμε για pointer τον `heap_private_buff`.

```
Step 2: Use mmap(2) to allocate a private buffer of size equal to 1 page and print the VM map again.
```

```
7f9ab4381000-7f9ab4384000 rw-p 00000000 00:00 0
```

```
Virtual Memory Map of process [33323]:
55a7b9165000-55a7b9166000 r--p 00000000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9166000-55a7b9167000 r-xp 00001000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9167000-55a7b9168000 r--p 00002000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9168000-55a7b9169000 r--p 00002000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9169000-55a7b916a000 rw-p 00003000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7ba3ad000-55a7ba3ce000 rw-p 00000000 00:00 0 [heap]
7f9ab4000000-7f9ab4022000 r--p 00000000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab4022000-7f9ab419a000 r-xp 00022000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab419a000-7f9ab41f2000 r--p 0019a000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f2000-7f9ab41f6000 r--p 001f1000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f6000-7f9ab41f8000 rw-p 001f5000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f8000-7f9ab4205000 rw-p 00000000 00:00 0
7f9ab4365000-7f9ab4368000 rw-p 00000000 00:00 0
7f9ab4381000-7f9ab4384000 rw-p 00000000 00:00 0
7f9ab4384000-7f9ab4385000 r--p 00000000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab4385000-7f9ab43ad000 r-xp 00001000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43ad000-7f9ab43b7000 r--p 00029000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b7000-7f9ab43b9000 r--p 00033000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b9000-7f9ab43bb000 rw-p 00035000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffd19901000-7ffd19922000 rw-p 00000000 00:00 0 [stack]
7ffd199e3000-7ffd199e7000 r--p 00000000 00:00 0 [vvar]
7ffd199e7000-7ffd199e9000 r-xp 00000000 00:00 0 [vdso]
fffffffff6000000-fffffffff6010000 --xp 00000000 00:00 0 [vsyscall]
-----
```


Βήμα 3:

Όταν τυπώνουμε τη διεύθυνση φυσικής μνήμης για τον buffer βλέπουμε πως επιστρέφει τη τιμή 0. Ουσιαστικά αυτο που έχει συμβεί είναι πως επειδή το ΛΣ λειτουργεί με on demand paging, έχουμε αντιστοίχιση σε εικονική διεύθυνση αλλά όχι ακόμα σε φυσική μνήμη. Για να γίνει αντιστοιχία σε φυσική μνήμη μέσω page map θα έπρεπε πρώτα να ζητηθεί ένα page, το οποίο θα οδηγούσε αρχικά σε page fault και ύστερα το ΛΣ θα ορίσει ένα frame για την αντιστοίχιση και αυτή η διεύθυνση θα εγγραφεί στο πίνακα σελίδων (page map).

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

```
VA[0x7f9ab4381000] is not mapped; no physical memory allocated.  
Physical Address of the buffer: 0
```

Βήμα 4:

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

```
Physical Address of the buffer: 9181057024
```

Παρατηρούμε πως τώρα που γράψαμε μηδενικά στον buffer έγινε η διαδικασία που περιγράψαμε παραπάνω λόγω on demand paging και πλέον έχουμε αντιστοιχία σε φυσική διεύθυνση.

Βήμα 5:

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

```
Hello everyone!
```

```
Virtual Memory Map of process [33323]:
```

55a7b9165000-55a7b9166000	r--p	00000000	103:04	2622682	/home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9166000-55a7b9167000	r-xp	00001000	103:04	2622682	/home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9167000-55a7b9168000	r--p	00002000	103:04	2622682	/home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9168000-55a7b9169000	r--p	00002000	103:04	2622682	/home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9169000-55a7b916a000	rw-p	00003000	103:04	2622682	/home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7ba3ad000-55a7ba3ce000	rw-p	00000000	00:00	0	[heap]
7f9ab4000000-7f9ab4022000	r--p	00000000	103:04	526442	/usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab4022000-7f9ab419a000	r-xp	00022000	103:04	526442	/usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab419a000-7f9ab41f2000	r--p	0019a000	103:04	526442	/usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f2000-7f9ab41f6000	r--p	001f1000	103:04	526442	/usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f6000-7f9ab41f8000	rw-p	001f5000	103:04	526442	/usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f8000-7f9ab4205000	rw-p	00000000	00:00	0	
7f9ab4365000-7f9ab4368000	rw-p	00000000	00:00	0	
7f9ab4380000-7f9ab4381000	r--p	00000000	103:04	2628067	/home/brewed/Desktop/OSLab-2022-23/ex4/file.txt
7f9ab4381000-7f9ab4384000	rw-p	00000000	00:00	0	
7f9ab4384000-7f9ab4385000	r--p	00000000	103:04	526436	/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab4385000-7f9ab43ad000	r-xp	00001000	103:04	526436	/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43ad000-7f9ab43b7000	r--p	00029000	103:04	526436	/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b7000-7f9ab43b9000	r--p	00033000	103:04	526436	/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b9000-7f9ab43bb000	rw-p	00035000	103:04	526436	/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffd19901000-7ffd19922000	rw-p	00000000	00:00	0	[stack]
7ffd199e3000-7ffd199e7000	r--p	00000000	00:00	0	[vvar]
7ffd199e7000-7ffd199e9000	r-xp	00000000	00:00	0	[vdso]
ffffffff600000-ffffffff601000	--xp	00000000	00:00	0	[vsyscall]

Έχοντας απεικονίσει το `file.txt` στον εικονικό χώρο διευθύνσεων της διεργασίας μέσω file descriptor που αντιστοιχεί στο `file.txt`, τυπώνουμε με `printf` το περιεχόμενο του εικονικού χώρου και βλέπουμε το περιεχόμενο του `file.txt`.

Τυπώνοντας το χώρο εικονικών διευθύνσεων παρατηρούμε την virtual map address του `file.txt` στο σωρό.

Βήμα 6:

Δεσμεύουμε ξανά buffer, αυτή τη φορά shared (`heap_shared_buff`) και τυπώνουμε τον χάρτη εικονικών διευθύνσεων. Ο buffer αυτός θα είναι διαμοιραζόμενος ανάμεσα στις δύο διεργασίες που ακολουθούν. Βλέπουμε τον buffer στο heap ως:

7f9ab437f000-7f9ab4380000

καθώς και μία από τις άδειες της διεύθυνσης αυτής είναι 's' (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 [33323]:
55a7b9165000-55a7b9166000 r--p 00000000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9166000-55a7b9167000 r-xp 00001000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9167000-55a7b9168000 r--p 00002000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9168000-55a7b9169000 r--p 00002000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9169000-55a7b916a000 rw-p 00003000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7ba3ad000-55a7ba3ce000 rw-p 00000000 00:00 0 [heap]
7f9ab4000000-7f9ab4022000 r--p 00000000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab4022000-7f9ab419a000 r-xp 00022000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab419a000-7f9ab41f2000 r--p 0019a000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f2000-7f9ab41f6000 r--p 001f1000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f6000-7f9ab41f8000 rw-p 001f5000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f8000-7f9ab4205000 rw-p 00000000 00:00 0
7f9ab4365000-7f9ab4368000 rw-p 00000000 00:00 0
7f9ab437f000-7f9ab4380000 rw-s 00000000 00:01 16746 /dev/zero (deleted)
7f9ab4380000-7f9ab4381000 r--p 00000000 103:04 2628067 /home/brewed/Desktop/OSLab-2022-23/ex4/file.txt
7f9ab4381000-7f9ab4384000 rw-p 00000000 00:00 0
7f9ab4384000-7f9ab4385000 r--p 00000000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab4385000-7f9ab43ad000 r-xp 00001000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43ad000-7f9ab43b7000 r--p 00029000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b7000-7f9ab43b9000 r--p 00033000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b9000-7f9ab43bb000 rw-p 00035000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffd19901000-7ffd19922000 rw-p 00000000 00:00 0 [stack]
7ffd199e3000-7ffd199e7000 r--p 00000000 00:00 0 [vvar]
7ffd199e7000-7ffd199e9000 r-xp 00000000 00:00 0 [vdso]
fffffffff60000-fffffffff601000 --xp 00000000 00:00 0 [vsyscall]
-----
```

Βήμα 7:

VM Map of the child process:

Child's map:

Virtual Memory Map of process [33390]:

```
55a7b9165000-55a7b9166000 r--p 00000000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9166000-55a7b9167000 r-xp 00001000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9167000-55a7b9168000 r--p 00002000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9168000-55a7b9169000 r--p 00002000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9169000-55a7b916a000 rw-p 00003000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7ba3ad000-55a7ba3ce000 rw-p 00000000 00:00 0 [heap]
7f9ab4000000-7f9ab4022000 r--p 00000000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab4022000-7f9ab419a000 r-xp 00022000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab419a000-7f9ab41f2000 r--p 0019a000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f2000-7f9ab41f6000 r--p 001f1000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f6000-7f9ab41f8000 rw-p 001f5000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f8000-7f9ab4205000 rw-p 00000000 00:00 0
7f9ab4365000-7f9ab4368000 rw-p 00000000 00:00 0
7f9ab437f000-7f9ab4380000 rw-s 00000000 00:01 16746 /dev/zero (deleted)
7f9ab4380000-7f9ab4381000 r--p 00000000 103:04 2628067 /home/brewed/Desktop/OSLab-2022-23/ex4/file.txt
7f9ab4381000-7f9ab4384000 rw-p 00000000 00:00 0
7f9ab4384000-7f9ab4385000 r--p 00000000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab4385000-7f9ab43ad000 r-xp 00001000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43ad000-7f9ab43b7000 r--p 00029000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b7000-7f9ab43b9000 r--p 00033000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b9000-7f9ab43bb000 rw-p 00035000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffd19901000-7ffd19922000 rw-p 00000000 00:00 0 [stack]
7ffd199e3000-7ffd199e7000 r--p 00000000 00:00 0 [vvar]
7ffd199e7000-7ffd199e9000 r-xp 00000000 00:00 0 [vdso]
fffffffff600000-fffffffff601000 --xp 00000000 00:00 0 [vsyscall]
-----
```

VM Map of the parent process:

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

Parent's map:

Virtual Memory Map of process [33323]:

```
55a7b9165000-55a7b9166000 r--p 00000000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9166000-55a7b9167000 r-xp 00001000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9167000-55a7b9168000 r--p 00002000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9168000-55a7b9169000 r--p 00002000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9169000-55a7b916a000 rw-p 00003000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7ba3ad000-55a7ba3ce000 rw-p 00000000 00:00 0 [heap]
7f9ab4000000-7f9ab4022000 r--p 00000000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab4022000-7f9ab419a000 r-xp 00022000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab419a000-7f9ab41f2000 r--p 0019a000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f2000-7f9ab41f6000 r--p 001f1000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f6000-7f9ab41f8000 rw-p 001f5000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f8000-7f9ab4205000 rw-p 00000000 00:00 0
7f9ab4365000-7f9ab4368000 rw-p 00000000 00:00 0
7f9ab437f000-7f9ab4380000 rw-s 00000000 00:01 16746 /dev/zero (deleted)
7f9ab4380000-7f9ab4381000 r--p 00000000 103:04 2628067 /home/brewed/Desktop/OSLab-2022-23/ex4/file.txt
7f9ab4381000-7f9ab4384000 rw-p 00000000 00:00 0
7f9ab4384000-7f9ab4385000 r--p 00000000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab4385000-7f9ab43ad000 r-xp 00001000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43ad000-7f9ab43b7000 r--p 00029000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b7000-7f9ab43b9000 r--p 00033000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b9000-7f9ab43bb000 rw-p 00035000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffd19901000-7ffd19922000 rw-p 00000000 00:00 0 [stack]
7ffd199e3000-7ffd199e7000 r--p 00000000 00:00 0 [vvar]
7ffd199e7000-7ffd199e9000 r-xp 00000000 00:00 0 [vdso]
fffffffff600000-fffffffff601000 --xp 00000000 00:00 0 [vsyscall]
-----
```

Ο εικονικός χάρτης μνημών είναι ίδιος ανάμεσα σε διεργασία πατέρα και παιδί που δημιουργήθηκε μέσω `fork()`. Απο θεωρία θα πρέπει επίσης οι σελίδες να έχουν μόνο read access. Αν πάμε να γράφουμε σε μία από αυτές τότε θα δημιουργηθεί αντίγραφο σε ξεχωριστή θέση και τότε θα μπορούμε να γράφουμε σε αυτές (Copy on Write).

Βήμα 8:

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

```
Physical Address of the private buffer of the parent: 10707718144  
Physical Address of the private buffer of the child: 10707718144
```

Η φυσική μνήμη του buffer για το παιδί και τον πατέρα είναι ίδια (κοινό resource).

Βήμα 9:

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

```
Physical Address of the private buffer of the parent: 10707718144  
Physical Address of the private buffer of the child: 4840423424
```

Παρατηρούμε πως μετά την εγγραφή των μηδενικών έχει αλλάξει η φυσική διεύθυνση για τις δυο διεργασίες και πλέον δεν είναι ίδια.

Όταν δημιουργείται η διεργασία-παιδί, δεν λαμβάνει αμέσως ένα ξεχωριστό αντίγραφο ολόκληρου του χώρου μνήμης του γονέα. Αντ' αυτού, και οι δύο διεργασίες μοιράζονται αρχικά τα ίδια pages, τα οποία όμως χαρακτηρίζονται ως μόνο για ανάγνωση στις άδειες τους. Αν κάποια από τις διεργασίες προσπαθήσει να γράψει σε μια σελίδα, δημιουργείται ένα ξεχωριστό αντίγραφο αυτής της σελίδας για τη διεργασία που γράφει και για αυτό βλέπουμε αλλαγή στη φυσική διεύθυνση του παιδιού.

Βήμα 10:

```
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?
```

```
Physical Address of the shared buffer of the parent: 4846460928  
Physical Address of the shared buffer of the child: 4846460928
```

Στη περίπτωση του shared buffer η φυσική διεύθυνση δεν αλλάζει και οι δυο διεργασίες βλέπουν το ίδιο frame φυσικής μνήμης κατι που επιτρέπει τη διεργασιακή επικοινωνία (IPC).

Βήμα 11:

Απενεργοποιούμε τα δικαιώματα write απο πλευράς του παιδιού με `mprotect()`

```
mprotect( addr: heap_shared_buf, len: buffer_size, prot: PROT_READ);
```

VM Map of the child process:

```
7f9ab437f000-7f9ab4380000 r--s 00000000 00:01 16746 /dev/zero (deleted)

Child's map:

Virtual Memory Map of process [33390]:
55a7b9165000-55a7b9166000 r--p 00000000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9166000-55a7b9167000 r-xp 00001000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9167000-55a7b9168000 r--p 00002000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9168000-55a7b9169000 r--p 00002000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9169000-55a7b916a000 rw-p 00003000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7ba3ad000-55a7ba3ce000 rw-p 00000000 00:00 0 [heap]
7f9ab4000000-7f9ab4022000 r--p 00000000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab4022000-7f9ab419a000 r-xp 00022000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab419a000-7f9ab41f2000 r--p 0019a000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f2000-7f9ab41f6000 r--p 001f1000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f6000-7f9ab41f8000 rw-p 001f5000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f8000-7f9ab4205000 rw-p 00000000 00:00 0
7f9ab4365000-7f9ab4368000 rw-p 00000000 00:00 0
7f9ab437f000-7f9ab4380000 r--s 00000000 00:01 16746 /dev/zero (deleted)
7f9ab4380000-7f9ab4381000 r--p 00000000 103:04 2628067 /home/brewed/Desktop/OSLab-2022-23/ex4/file.txt
7f9ab4381000-7f9ab4384000 rw-p 00000000 00:00 0
7f9ab4384000-7f9ab4385000 r--p 00000000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab4385000-7f9ab43ad000 r-xp 00001000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43ad000-7f9ab43b7000 r--p 00029000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b7000-7f9ab43b9000 r--p 00033000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b9000-7f9ab43bb000 rw-p 00035000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffd19901000-7ffd19922000 rw-p 00000000 00:00 0 [stack]
7ffd199e3000-7ffd199e7000 r--p 00000000 00:00 0 [vvar]
7ffd199e7000-7ffd199e9000 r-xp 00000000 00:00 0 [vdso]
fffffffff600000-fffffffff601000 --xp 00000000 00:00 0 [vsyscall]
-----
```

VM Map of the parent process:

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

7f9ab437f000-7f9ab4380000 rw-s 00000000 00:01 16746 /dev/zero (deleted)

Parent's map:

Virtual Memory Map of process [33323]:
55a7b9165000-55a7b9166000 r--p 00000000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9166000-55a7b9167000 r-xp 00001000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9167000-55a7b9168000 r--p 00002000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9168000-55a7b9169000 r--p 00002000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7b9169000-55a7b916a000 rw-p 00003000 103:04 2622682 /home/brewed/Desktop/OSLab-2022-23/ex4/mmap
55a7ba3ad000-55a7ba3ce000 rw-p 00000000 00:00 0 [heap]
7f9ab4000000-7f9ab4022000 r--p 00000000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab4022000-7f9ab419a000 r-xp 00022000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab419a000-7f9ab41f2000 r--p 0019a000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f2000-7f9ab41f6000 r--p 001f1000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f6000-7f9ab41f8000 rw-p 001f5000 103:04 526442 /usr/lib/x86_64-linux-gnu/libc.so.6
7f9ab41f8000-7f9ab4205000 rw-p 00000000 00:00 0
7f9ab4365000-7f9ab4368000 rw-p 00000000 00:00 0
7f9ab437f000-7f9ab4380000 rw-s 00000000 00:01 16746 /dev/zero (deleted)
7f9ab4380000-7f9ab4381000 r--p 00000000 103:04 2628067 /home/brewed/Desktop/OSLab-2022-23/ex4/file.txt
7f9ab4381000-7f9ab4384000 rw-p 00000000 00:00 0
7f9ab4384000-7f9ab4385000 r--p 00000000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab4385000-7f9ab43ad000 r-xp 00001000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43ad000-7f9ab43b7000 r--p 00029000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b7000-7f9ab43b9000 r--p 00033000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f9ab43b9000-7f9ab43bb000 rw-p 00035000 103:04 526436 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffd19901000-7ffd19922000 rw-p 00000000 00:00 0 [stack]
7ffd199e3000-7ffd199e7000 r--p 00000000 00:00 0 [vvar]
7ffd199e7000-7ffd199e9000 r-xp 00000000 00:00 0 [vdso]
fffffffff600000-fffffffff601000 --xp 00000000 00:00 0 [vsyscall]
-----
```

Βήμα 12:

Αποδεσμεύουμε τους δεσμευμένους buffers μέσω της `munmap()`, αν δε το κάναμε εμείς θα το έκανε μόνο του το ΛΣ μετά το πέρας του εκτελέσιμου (όταν κληθεί η `exit()`), ωστόσο είναι καλή πρακτική να το κάνουμε όταν δε χρησιμοποιούμε πλέον τον εικονικό χώρο.

```
munmap( addr: heap_shared_buf, len: buffer_size);  
munmap( addr: heap_private_buf, len: buffer_size);
```

1.2 Παράλληλος υπολογισμός Mandelbrot με διεργασίες αντί για νήματα

1.2.1 Semaphores πάνω από διαμοιραζόμενη μνήμη

Στην υλοποίηση έχουμε ανάγκη συγχρονισμού, όμως έχουμε διεργασίες οι οποίες δεν έχουν κοινή μνήμη και πρέπει να δημιουργήσουμε ένα κοινό χώρο για να βλέπουν τους ίδιους semaphores. Έτσι δημιουργούμε με `create_shared_memory_area()` και τελικά όταν τελειώσουμε τους υπολογισμούς τον καταστρέφουμε με `destroy_shared_memory_area()`.

Πηγαίος Κώδικας:

```
/*  
 * mandel.c  
 *  
 * A program to draw the Mandelbrot Set on a 256-color xterm.  
 *  
 */  
  
#include <stdio.h>  
#include <unistd.h>  
#include <assert.h>  
#include <string.h>  
#include <math.h>  
#include <semaphore.h>  
#include <stdlib.h>  
#include <wait.h>
```



```

#include <sys/mman.h>

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

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

sem_t *shared;
/*
 * Output at the terminal is x_chars wide by y_chars long
 */
int y_chars = 50;
int x_chars = 90;

/*
 * The part of the complex plane to be drawn:
 * upper left corner is (xmin, ymax), lower right corner is (xmax, ymin)
 */
double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;

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

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

    int n;
    int val;

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

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

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

```

```

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

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

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

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

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

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

    int i;
    for(i=proc_id; i<y_chars; i+=number_of_procs) {
        compute_mandel_line(i, color_val);

        sem_wait(&shared[(proc_id) % number_of_procs]);
        output_mandel_line(1, color_val);
        sem_post(&shared[(proc_id + 1) % number_of_procs]);
    }
}

/*
 * 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 */

    addr = mmap(NULL, pages, PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS, 0, 0);

    return addr;
}

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

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

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

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

int main(int argc, char *argv[])
{
    pid_t procs;
    int NPROCS;
    int i;
    /* The number of threads is read from the program's arguments */
    NPROCS = atoi(argv[1]);

    shared=create_shared_memory_area(sizeof(sem_t)*NPROCS);

    if (argc != 2) {
        printf("Usage: %s <NPROCS> \n", argv[0]);
        exit(1);
    }
}

```

```

}
for(i=0; i<NPROCS; i++){
    if (i==0){
        sem_init(&shared[i],1,1);
    }
    else{
        sem_init(&shared[i],1,0);
    }
}

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.
 */

for (i=0; i<NPROCS; i++){
    procs=fork();
    if (procs < 0) {
        perror("main: fork");
        exit(1);
    }
    if (procs == 0) {
        compute_and_output_mandel_line(i,NPROCS);
        exit(10);
    }
}

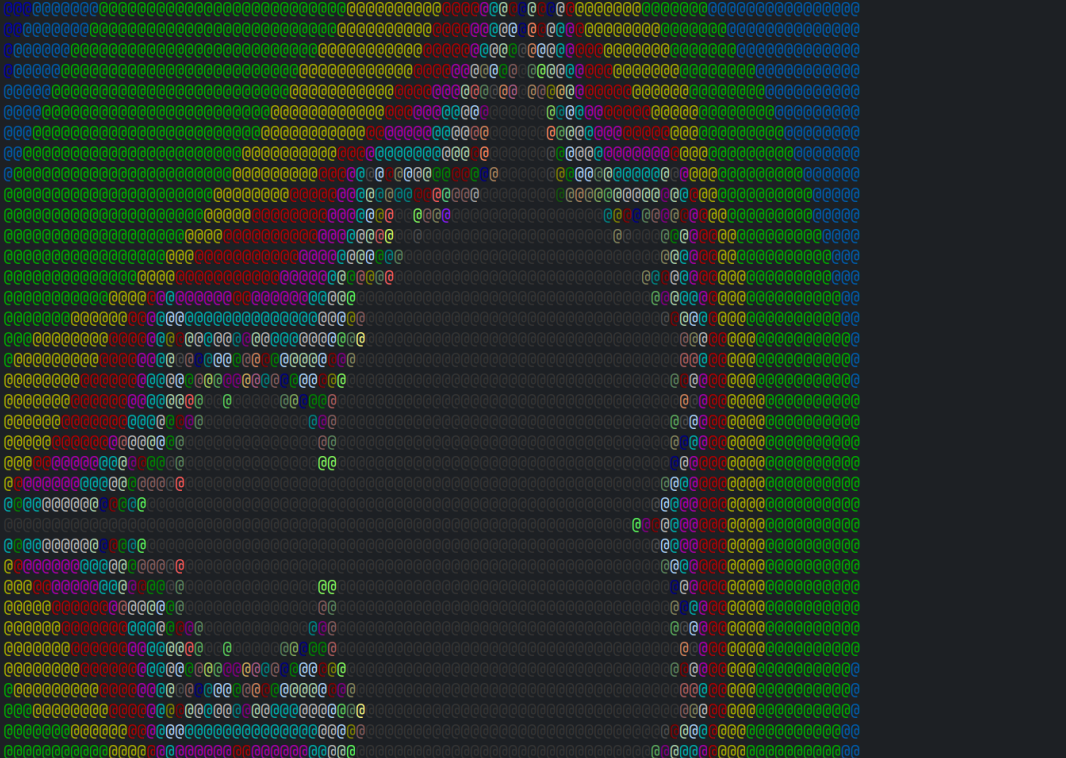
for (i=0; i<NPROCS; i++){
    wait(NULL);
}
destroy_shared_memory_area(shared,sizeof(sem_t)*NPROCS);

for(i=0; i<NPROCS; i++){
    sem_destroy(&shared[i]);
}
reset_xterm_color(1);
return 0;
}

```

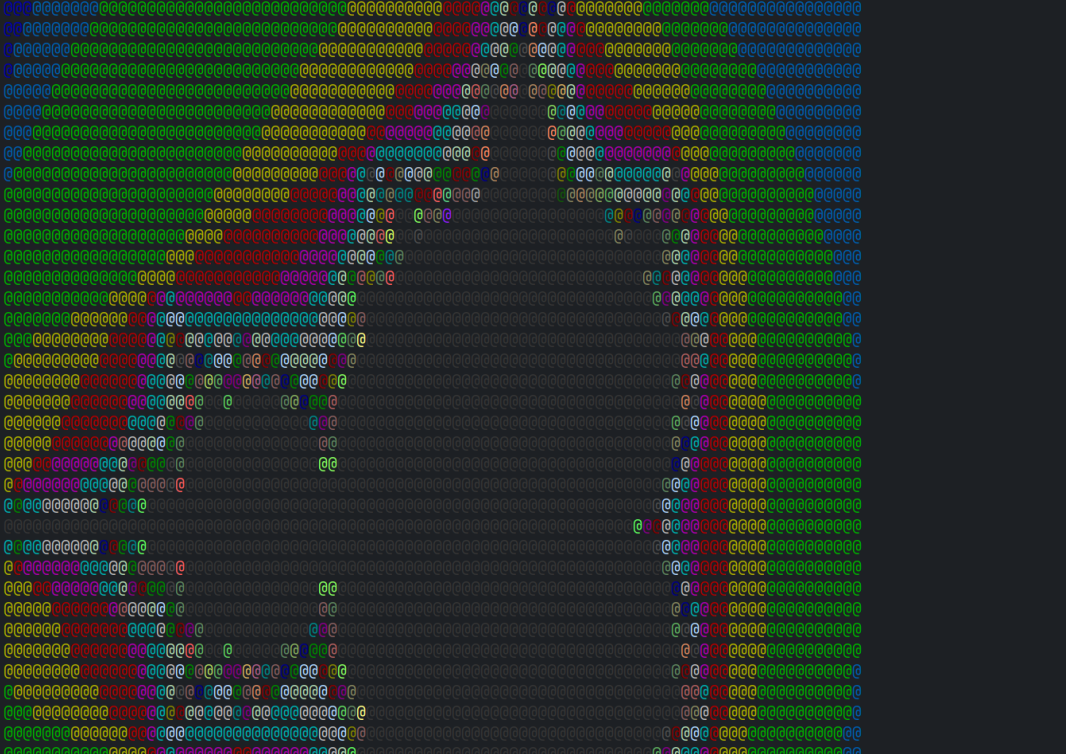
Παρακάτω εκτελούμε το πρόγραμμα με 1 process (σειριακός υπολογισμός):

```
brewed@brewed-laptop:~/Desktop/0SLab-2022-23/ex4/mandel$ ./mandel-sem-fork 1
```



Παρακάτω εκτελούμε το πρόγραμμα με 8 processes (παράλληλος υπολογισμός, βέλτιστος για το σύστημα μας):

```
brewed@brewed-laptop:~/Desktop/0SLab-2022-23/ex4/mandel$ ./mandel-sem-fork 8
```



Ερωτήσεις:

1. Η υλοποίηση με threads είναι πιο γρήγορη από αυτή με διεργασίες. Αυτό οφείλεται στο ότι τα threads έχουν by default κοινά resources, εκτός από το stack. Προφανώς η συνάρτηση `created_shared_memory_area()` δεν είναι τόσο optimized για χρήση κοινής μνήμης όσο το να έχουμε natively κοινή μνήμη. Στη περίπτωση επίσης των διεργασιών το context switch που θα έκανε η CPU θα ήταν χρονοβόρο διότι θα έπρεπε να μεταφέρει τα PCBs κάτι που δε χρειάζεται στη περίπτωση των threads.

1.2.2 Υλοποίηση χωρίς semaphores

Στην υλοποίηση αυτή ουσιαστικά δεν απαιτείται συγχρονισμός και δεν υπάρχει κρίσιμο τμήμα, κάθε διεργασία αναλαμβάνει κάποιες γραμμές του mandelbrot και δεν υπάρχει conflict με κάποια άλλη. Τελικά τυπώνεται όλος ο buffer στην οθόνη συνολικά.

Πηγαίος κώδικας:

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

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

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

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

int **shared;
/*
 * Output at the terminal is x_chars wide by y_chars long
 */
```

```

int y_chars = 50;
int x_chars = 90;

/*
 * The part of the complex plane to be drawn:
 * upper left corner is (xmin, ymax), lower right corner is (xmax, ymin)
 */
double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;

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

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

    int n;
    int val;

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

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

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

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

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

```

```

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

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

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

void compute_and_output_mandel_line(int proc_id, int number_of_procs) {
    /*
     * A temporary array, used to hold color values for the line being drawn
     */
    int i;
    for (i = proc_id; i < y_chars; i += number_of_procs) {
        compute_mandel_line(i, shared[i]);
    }
}

/*
 * 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 */

    addr = mmap(NULL, pages, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, 0, 0);

    return addr;
}

```

```

}

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

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

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

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

int main(int argc, char *argv[])
{
pid_t root;
int NPROCS;
int i;
/* The number of threads is read from the program's arguments */
NPROCS = atoi(argv[1]);

shared = create_shared_memory_area(sizeof(unsigned char *) * y_chars);

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

if (argc != 2) {
printf("Usage: %s <NPROCS> \n", argv[0]);
exit(1);
}

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.
 */

for (i=0; i<NPROCS; i++){
root=fork();
if (root < 0) {

```

```

perror("main: fork");
exit(1);
}
if (root == 0) {
compute_and_output_mandel_line(i,NPROCS);
exit(10);
}
}

for (i=0; i<NPROCS; i++){
wait(NULL);
}
for(i=0; i<y_chars; i++){
output_mandel_line(1,shared[i]);
}

destroy_shared_memory_area(shared, sizeof(unsigned char *) * y_chars);

reset_xterm_color(1);
return 0;
}


```

Παρακάτω εκτελούμε το πρόγραμμα με 8 processes (παράλληλος υπολογισμός, βέλτιστος για το σύστημα μας, αλλά συνολική εκτύπωση στο τέλος):

```

brewed@brewed-laptop:~/Desktop/OSLab-2022-23/ex4/mandel$ ./mandel-fork 8

```



Ερωτήσεις:

1. Αν αλλάξουμε το μέγεθος του buffer και έχει NPROCS γραμμές τότε προφανώς δε μπορούμε να τυπώσουμε συνολικά το mandelbrot. Θα έπρεπε να τυπώνουμε ανα μέρη το συνολικό αδειάζοντας κάθε φορά τον buffer στην οθόνη και ξαναγεμίζοντας τον.