National Technical University of Athens

School of Electrical and Computer Engineering

Λειτουργικά Συστήματα Υπολογιστών (Τμήμα 1)

Άσκηση 4: Μηχανισμοί Εικονικής Μνήμης

ΧΡΙΣΤΟΦΟΡΟΣ ΒΑΡΔΑΚΗΣ (03118883)

ΓΕΩΡΓΙΟΣ ΛΥΜΠΕΡΑΚΗΣ (03118881)

oslaba69

6 Ιουνίου 2021

Άσκηση 1.1 :

Στην πρώτη άσκηση δοκιμάζουμε διάφορες κλήσεις συστήματος που έχουν να κάνουν με την εικονική μνήμη των διεργασιών και με διαδοχικά βήματα/ερωτήματα εξερευνούμε καλύτερα τις έννοιες που διδαχθήκαμε στην θεωρία ,εξετάζοντας έτσι ένα πραγματικό υπολογιστικό σύστημα ως προς τη διαχείριση της μνήμης του.

Πιο συγκεκριμένα , ακολουθώντας 12 διαδοχικά βήματα θα επιδιώξουμε να αποκτήσουμε μια πρώτη εικόνα για την εικονική ,φυσική μνήμη και τους μηχανισμούς της μέσω κυρίως της κλήσης συστήματος .

```
void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset)

int munmap(void *addr, size_t length)
```

Καθώς , επίσης και των βοηθητικών συναρτήσεων

```
void show_maps(void)

void show_va_info(uint64_t va)

uint64_t get_physical_address(uint64_t va)
```

Επομένως, τώρα παραθέτουμε τον κώδικα της πρώτης άσκησης που είναι υλοποιημένα τα δώδεκα ερωτήματα καθώς επίσης και οι αντίστοιχοι έξοδοι.

Αρχείο mmap.c

```
/*
 * mmap.c
 *
 * Examining the virtual memory of processes.
 *
 * Operating Systems course, CSLab, ECE, NTUA
 *
 */

//----------------------
//Extra Libraries
#include <inttypes.h>
#include <sys/stat.h>
//----------------------
#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("\n\nChild Process:\n");
      show_maps();
//-------------------------------------------------------------------------------
```

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

	uint64_t vc = (uint64_t) heap_private_buf;
	/*
	 * TODO: Write your code here to complete child's part of Step 8.
	 */

	printf("\n\nChild Process:\n");
	printf("\nInformation for the VA of the buffer:\n");
	show_va_info(vc);

	printf("\n\nInformation for the PA of the buffer: %"PRIu64"
\n",get_physical_address(vc));

//-------------------------------------------------------------------------------------------
	/*
	 * Step 9 - Child
	 */
	if (0 != raise(SIGSTOP))
		die("raise(SIGSTOP)");
	/*
	 * TODO: Write your code here to complete child's part of Step 9.
	 */
	unsigned i;
	int * arr = (void *) heap_private_buf;
	long page_size = get_page_size();
	for (i = 0; i < (page_size/sizeof(int)); ++i){
		arr[i]=17;
	}

	printf("\n\nChild Process:\n");
	printf("\nInformation for the VA of the buffer:\n");
	show_va_info(vc);

	printf("\n\nInformation for the PA of the buffer: %"PRIu64"
\n",get_physical_address(vc));
	/*
	 * 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 vcs = (uint64_t) heap_shared_buf;
		unsigned j;
		int * arrshared = (void *) heap_shared_buf;
		for (j = 0; j < (page_size/sizeof(int)); ++j){
		arrshared[j]=8;
		}
	printf("\n\nChild Process:\n");
	printf("\nInformation for the VA of the buffer:\n");
	show_va_info(vcs);

	printf("\n\nInformation for the PA of the buffer: %"PRIu64"
\n",get_physical_address(vcs));
```

```
	/*
	 * 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,page_size,PROT_READ);
	printf("\n\nChild Process:\n");
	printf("\nInformation for the VA of the buffer:\n");
	show_va_info(vcs);

	printf("\n\nInformation for the PA of the buffer: %"PRIu64"
\n",get_physical_address(vcs));
	//show_maps();
	/*
	 * Step 12 - Child
	 */
	/*
	 * TODO: Write your code here to complete child's part of Step 12.
	 */
}

/*
 * 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 "\nStep 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("Father Process:\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
	 */
	uint64_t vp = (uint64_t) heap_private_buf;
	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.
	 */
	printf("Father Process:\n");
	printf("\nInformation for the VA of the buffer:\n");
	show_va_info(vp);
	printf("\n\nInformation for the PA of the buffer: %"PRIu64"
\n",get_physical_address(vp));


	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.
	 */
	printf("Father Process:\n");
	printf("\nInformation for the VA of the buffer:\n");
	show_va_info(vp);
	printf("\n\nInformation for the PA of the buffer: %"PRIu64"
\n",get_physical_address(vp));


	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 vps = (uint64_t) heap_shared_buf;
	printf("Father Process:\n");
	printf("\nInformation for the VA of the buffer:\n");
	show_va_info(vps);
	printf("\n\nInformation for the PA of the buffer: %"PRIu64"
\n",get_physical_address(vps));


	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();

	/*
	 * TODO: Write your code here to complete parent's part of Step 11.
	 */
	uint64_t vps2 = (uint64_t) heap_shared_buf;
	printf("Father Process:\n");
	printf("\nInformation for the VA of the buffer:\n");
	show_va_info(vps2);
	printf("\n\nInformation for the PA of the buffer: %"PRIu64"
\n",get_physical_address(vps2));
	//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
	 */
		if (munmap(heap_shared_buf, sizeof(heap_shared_buf)) == -1) {
			perror("shared buffer munmap failed");
			exit(1);
		}
		if (munmap(heap_private_buf, sizeof(heap_private_buf)) == -1) {
			perror("private buffer munmap failed");
			exit(1);
		}

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

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

	mypid = getpid();
	buffer_size = 1 * get_page_size();

	printf("\nProcess PID: %d \n",mypid);
//-----------------------------------------------------------------------------
	/*
	 * 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();


	long page_size = get_page_size();
	printf("Page Size : %ld Bytes or %ld KBytes\n",page_size,page_size/1024);



	if((heap_private_buf = mmap(NULL,page_size,PROT_READ | PROT_WRITE,MAP_PRIVATE |
MAP_ANONYMOUS,-1,0))==MAP_FAILED){//changed -1 to fd
		printf("Mapping Failed !! \n");
		exit(-1);
	}
	  uint64_t va = (uint64_t) heap_private_buf;

	  printf("\nInformation for the VA of the buffer:\n");
	  show_va_info(va);


	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("\nInformation for the VA of the buffer:\n");
	show_va_info(va);

	printf("\n\nInformation for the PA of the buffer: %"PRIu64"
\n",get_physical_address(va));
//------------------------------------------------------------------------------------
	/*
	 * 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();

	unsigned i;
	int * arr = (void *) heap_private_buf;

	for (i = 0; i < (page_size/sizeof(int)); ++i){
		arr[i]=0;
	}
/*
	for(i=0; i <(page_size/sizeof(int)); ++i){
		 printf("Buffer[%d] = %d \n",i,arr[i]);
	}
*/
	  printf("\nInformation for the VA of the buffer:\n");
	  show_va_info(va);
```

```
        printf("\n\nInformation for the PA of the buffer: %"PRIu64"
\n",get_physical_address(va));

        show_maps();

//-------------------------------------------------------------------------------------------
      /*
       * 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();

      char * filename ="file.txt";//"/home/oslab/oslaba69/ALL/G/mmap/file.txt";

      if((fd = open(filename,O_RDONLY)) == -1){
            perror("OPEN");
            exit(-1);
      }


      struct stat st;

      if(fstat(fd,&st)==-1){
              perror("Get File Size");
            exit(-1);
      }

      unsigned length = st.st_size;


      if(length  == 0){
            printf("\nEmpty File\n");
      }

      char * text;

      if((text = mmap(NULL,length,PROT_READ,MAP_SHARED,fd,0))==MAP_FAILED){
            printf("Mapping Failed!\n");
            close(fd);
            exit(-1);
      }


      for(i = 0; i<length; ++i){
            printf("Text[%d] = %c\n",i,text[i]);
      }

        va = (uint64_t) text;

        printf("\nInformation for the VA of the buffer:\n");
        show_va_info(va);

        printf("\n\nInformation for the PA of the buffer: %"PRIu64"
\n",get_physical_address(va));
      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();


	if((heap_shared_buf = mmap(NULL,page_size,PROT_READ | PROT_WRITE,MAP_SHARED|
MAP_ANONYMOUS,-1,0))==MAP_FAILED){
		printf("Mapping Failed !! \n");
		exit(-1);
	}
	uint64_t v = (uint64_t) heap_shared_buf;

	printf("\nInformation for the VA of the buffer:\n");
	show_va_info(v);

	printf("\n\nInformation for the PA of the buffer: %"PRIu64"
\n",get_physical_address(v));

	show_maps();

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

	parent(p);
	if (-1 == close(fd))//fd
		perror("close");
	return 0;
}
```

Έξοδος Εκτελέσιμου *mmap*

```
Step 1: Print the virtual address space map of this process [19919].


Virtual Memory Map of process [19919]:
00400000-00403000 r-xp 00000000 00:21 20324262                           /home/oslab/oslaba69/ALL/mmap/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 20324262                           /home/oslab/oslaba69/ALL/mmap/mmap/mmap
01876000-01897000 rw-p 00000000 00:00 0                                  [heap]
7f5630302000-7f56304a3000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56304a3000-7f56306a3000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a3000-7f56306a7000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a7000-7f56306a9000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a9000-7f56306ad000 rw-p 00000000 00:00 0
7f56306ad000-7f56306ce000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308c0000-7f56308c3000 rw-p 00000000 00:00 0
7f56308c8000-7f56308cd000 rw-p 00000000 00:00 0
7f56308cd000-7f56308ce000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308ce000-7f56308cf000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308cf000-7f56308d0000 rw-p 00000000 00:00 0
7fff310ff000-7fff31120000 rw-p 00000000 00:00 0                          [stack]
7fff311af000-7fff311b2000 r--p 00000000 00:00 0                          [vvar]
7fff311b2000-7fff311b4000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

Στο παραπάνω βήμα εκτυπώνουμε το χάρτη μνήμης του προγράμματος (pid = 19919). Κάθε γραμμή του χάρτη μνήμης αντιστοιχεί σε μία περιοχή εικονικής μνήμης ενώ κάθε στήλη περιέχει ορισμένες πληροφορίες για την αντίστοιχη περιοχής. Πιο συγκεκριμένα, η πρώτη στήλη αντιστοιχεί στην πρώτη και την τελευταία διεύθυνση της εικονικής μνήμης του εύρους διευθύνσεων, η δεύτερη στα δικαιώματα πρόσβασης που έχει η διεργασία στο συγκεκριμένο χώρο μνήμης (δικαίωμα εγγραφής, ανάγνωσης ή/και εκτέλεσης).Αν το εύρος αποτελεί απεικόνιση (mapping) αρχείου η τρίτη στήλη αντιστοιχεί στο offset από την αρχή του αρχείου, η τέταρτη στήλη αντιστοιχεί στη συσκευή που βρίσκετε το αρχείο ενώ η τελευταία στο inode.

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

Page Size : 4096 Bytes or 4 KBytes

Information for the VA of the buffer:
7f56308c7000-7f56308cd000 rw-p 00000000 00:00 0

Virtual Memory Map of process [19919]:
00400000-00403000 r-xp 00000000 00:21 20324262                           /home/oslab/oslaba69/ALL/mmap/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 20324262                           /home/oslab/oslaba69/ALL/mmap/mmap/mmap
01876000-01897000 rw-p 00000000 00:00 0                                  [heap]
7f5630302000-7f56304a3000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56304a3000-7f56306a3000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a3000-7f56306a7000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a7000-7f56306a9000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a9000-7f56306ad000 rw-p 00000000 00:00 0
7f56306ad000-7f56306ce000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308c0000-7f56308c3000 rw-p 00000000 00:00 0
7f56308c7000-7f56308cd000 rw-p 00000000 00:00 0
7f56308cd000-7f56308ce000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308ce000-7f56308cf000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308cf000-7f56308d0000 rw-p 00000000 00:00 0
7fff310ff000-7fff31120000 rw-p 00000000 00:00 0                          [stack]
7fff311af000-7fff311b2000 r--p 00000000 00:00 0                          [vvar]
7fff311b2000-7fff311b4000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-------------------------------------------------------
```

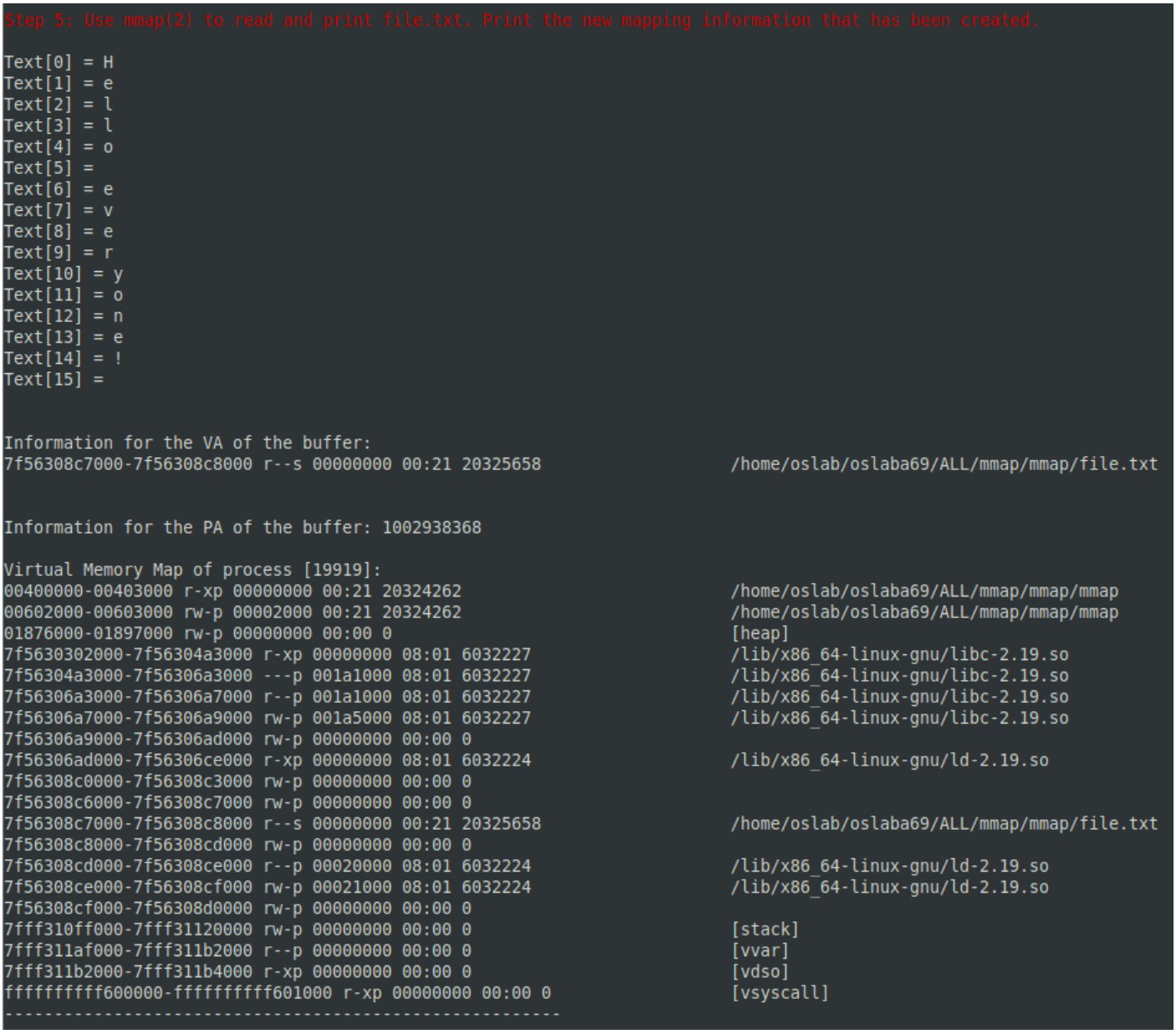
Στο επόμενο βήμα δεσμεύουμε έναν private buffer μεγέθους μίας σελίδας (4KB). Βλέπουμε ότι ο buffer αντιστοιχεί στις εικονικές διευθύνσεις 7f56308c7000 – 7f56308cd000. Παρατηρούμε στο χάρτη μνήμης ότι το offset το device και το inode του εύρους διευθύνσεων έγιναν 0, γεγονός που υποδηλώνει ότι το συγκεκριμένο εύρος δεν αντιστοιχεί σε αρχείο.

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


Information for the VA of the buffer:
7f56308c7000-7f56308cd000 rw-p 00000000 00:00 0
VA[0x7f56308c8000] is not mapped; no physical memory allocated.


Information for the PA of the buffer: 0
```

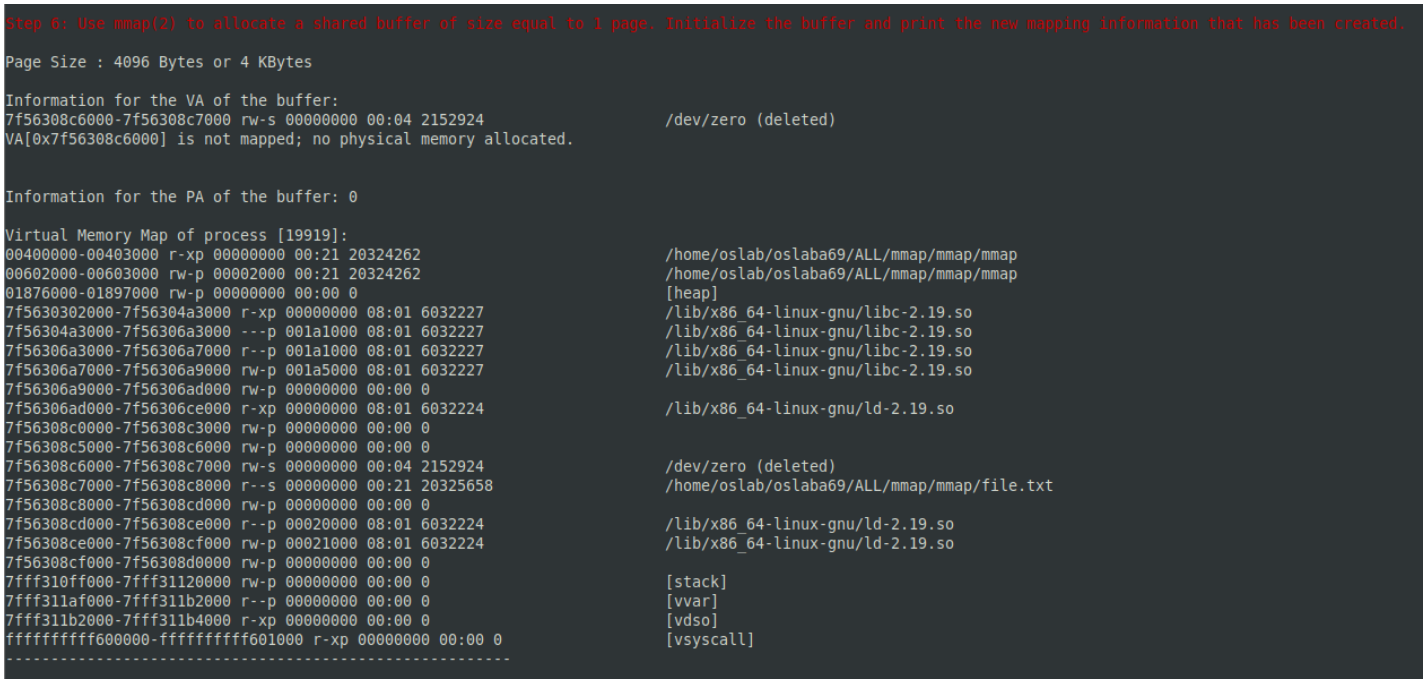
Στο βήμα αυτό επιχειρούμε να εκτυπώσουμε τη διεύθυνση του buffer στη φυσική μνήμη του συστήματος. Παρατηρούμε ότι στο σημείο αυτό ο buffer δεν αντιστοιχεί σε πραγματική διεύθυνση στη μνήμη. Αυτό είναι αναμενόμενο, μιας και το πρόγραμμα δεν έχει ακόμα προσπελάσει τον συγκεκριμένο χώρο.

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


Information for the VA of the buffer:
7f56308c7000-7f56308cd000 rw-p 00000000 00:00 0


Information for the PA of the buffer: 5022351360

Virtual Memory Map of process [19919]:
00400000-00403000 r-xp 00000000 00:21 20324262                           /home/oslab/oslaba69/ALL/mmap/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 20324262                           /home/oslab/oslaba69/ALL/mmap/mmap/mmap
01876000-01897000 rw-p 00000000 00:00 0                                  [heap]
7f5630302000-7f56304a3000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56304a3000-7f56306a3000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a3000-7f56306a7000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a7000-7f56306a9000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a9000-7f56306ad000 rw-p 00000000 00:00 0
7f56306ad000-7f56306ce000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308c0000-7f56308c3000 rw-p 00000000 00:00 0
7f56308c7000-7f56308cd000 rw-p 00000000 00:00 0
7f56308cd000-7f56308ce000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308ce000-7f56308cf000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308cf000-7f56308d0000 rw-p 00000000 00:00 0
7fff310ff000-7fff31120000 rw-p 00000000 00:00 0                          [stack]
7fff311af000-7fff311b2000 r--p 00000000 00:00 0                          [vvar]
7fff311b2000-7fff311b4000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

Στο ερώτημα αυτό επιχειρούμε ξανά να εκτυπώσουμε τη φυσική διεύθυνση του buffer, αφου ομως πρώτα γράψουμε σε αυτόν (προσπέλαση του χώρου). Παρατηρούμε ότι μετά την εγγραφή ο buffer αντιστοιχεί στη φυσική διεύθυνση 5022351360.



Στο βήμα αυτό απεικονίζουμε στη μνήμη ένα αρχείο με όνομα file.txt. Όπως βλέπουμε το περιεχόμενο του αρχείου (Η φράση “Hello everyone!”) βρίσκετε αποθηκευμένη στις 15 πρώτες θέσεις του buffer. Μάλιστα αν ανατρέξουμε στον χάρτη μνήμης και συγκεκριμένα στη γραμμή που αντιστοιχεί στον buffer, θα παρατηρήσουμε ότι στις 2 τελευταίες στήλες, δηλαδή τις device και inode, υπάρχουν πλέον οι τιμές που αντιστοιχούν στο αρχείο file.txt. Επίσης στη δεξιά μεριά του χάρτη μνήμης στη γραμμή αυτή βρίσκετε το μονοπάτι (path) όπου βρίσκετε αποθηκευμένο το αρχείο.



Στο ερώτημα αυτό καλούμαστε να δεσμεύσουμε έναν shared buffer, δηλαδή έναν buffer διαμοιραζόμενο ανάμεσα σε δύο διεργασίες του προγράμματος. Ο buffer που δεσμεύσαμε αντιστοιχεί στο εύρος εικονικών διευθύνσεων 7f56308c6000 – 7f56308c7000. Παρατηρούμε ότι στον χάρτη μνήμης ο buffer αποτελεί απεικόνιση του αρχείου /dev/zero. Αυτό συμβαίνει επειδή το λειτουργικό σύστημα χρησιμοποιεί το αρχείο αυτό κατά τη δημιουργία του shared buffer, όταν αυτός δεν αντιστοιχεί σε κάποιο άλλο αρχείο κατά τη δημιουργία του. Μάλιστα με το που θα δημιουργηθεί ο buffer το λειτουργικό διαγράφει το αρχείο μιας και δεν το χεριάζετε πια, και για το λόγο αυτό εμφανίζετε ως /dev/zero(deleted).

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

Father Process:

Virtual Memory Map of process [19919]:
00400000-00403000 r-xp 00000000 00:21 20324262                           /home/oslab/oslaba69/ALL/mmap/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 20324262                           /home/oslab/oslaba69/ALL/mmap/mmap/mmap
01876000-01897000 rw-p 00000000 00:00 0                                  [heap]
7f5630302000-7f56304a3000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56304a3000-7f56306a3000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a3000-7f56306a7000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a7000-7f56306a9000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a9000-7f56306ad000 rw-p 00000000 00:00 0
7f56306ad000-7f56306ce000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308c0000-7f56308c3000 rw-p 00000000 00:00 0
7f56308c5000-7f56308c6000 rw-p 00000000 00:00 0
7f56308c6000-7f56308c7000 rw-s 00000000 00:04 2152924                    /dev/zero (deleted)
7f56308c7000-7f56308c8000 r--s 00000000 00:21 20325658                   /home/oslab/oslaba69/ALL/mmap/mmap/file.txt
7f56308c8000-7f56308cd000 rw-p 00000000 00:00 0
7f56308cd000-7f56308ce000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308ce000-7f56308cf000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308cf000-7f56308d0000 rw-p 00000000 00:00 0
7fff310ff000-7fff31120000 rw-p 00000000 00:00 0                          [stack]
7fff311af000-7fff311b2000 r--p 00000000 00:00 0                          [vvar]
7fff311b2000-7fff311b4000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------



Child Process:

Virtual Memory Map of process [19920]:
00400000-00403000 r-xp 00000000 00:21 20324262                           /home/oslab/oslaba69/ALL/mmap/mmap/mmap
00602000-00603000 rw-p 00002000 00:21 20324262                           /home/oslab/oslaba69/ALL/mmap/mmap/mmap
01876000-01897000 rw-p 00000000 00:00 0                                  [heap]
7f5630302000-7f56304a3000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56304a3000-7f56306a3000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a3000-7f56306a7000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a7000-7f56306a9000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7f56306a9000-7f56306ad000 rw-p 00000000 00:00 0
7f56306ad000-7f56306ce000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308c0000-7f56308c3000 rw-p 00000000 00:00 0
7f56308c5000-7f56308c6000 rw-p 00000000 00:00 0
7f56308c6000-7f56308c7000 rw-s 00000000 00:04 2152924                    /dev/zero (deleted)
7f56308c7000-7f56308c8000 r--s 00000000 00:21 20325658                   /home/oslab/oslaba69/ALL/mmap/mmap/file.txt
7f56308c8000-7f56308cd000 rw-p 00000000 00:00 0
7f56308cd000-7f56308ce000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308ce000-7f56308cf000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7f56308cf000-7f56308d0000 rw-p 00000000 00:00 0
7fff310ff000-7fff31120000 rw-p 00000000 00:00 0                          [stack]
7fff311af000-7fff311b2000 r--p 00000000 00:00 0                          [vvar]
7fff311b2000-7fff311b4000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------
```

Στο ερώτημα αυτό εκτελούμε την κλήση συστήματος fork() δημιουργώντας μια νέα διεργασία (pid = 19920) στην οποία θα αναφερόμαστε στο εξής ως διεργασία-παιδί, ενώ για την ήδη υπάρχουσα διεργασία θα χρησιμοποιούμε τον όρο διεργασία-πατέρας. Παρατηρούμε ότι ο χάρτης μνήμης των δύο αυτών διεργασιών είναι ο ίδιος.

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

Father Process:

Information for the VA of the buffer:
7f56308c8000-7f56308cd000 rw-p 00000000 00:00 0


Information for the PA of the buffer: 5022351360


Child Process:

Information for the VA of the buffer:
7f56308c8000-7f56308cd000 rw-p 00000000 00:00 0


Information for the PA of the buffer: 5022351360
```

Παρατηρούμε ότι η φυσική διεύθυνση του buffer είναι ίδια και στον πατέρα και στο παιδί. Αυτό είναι αναμενόμενο, μιας και μετά το fork() η διεργασία δεν έχει επιχειρήσει να γράψει στον buffer οπότε διατηρεί τη φυσική διεύθυνση που είχε πριν το fork().

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

Father Process:

Information for the VA of the buffer:
7f56308c8000-7f56308cd000 rw-p 00000000 00:00 0


Information for the PA of the buffer: 5022351360


Child Process:

Information for the VA of the buffer:
7f56308c8000-7f56308cd000 rw-p 00000000 00:00 0


Information for the PA of the buffer: 4874911744
```

Στο ερώτημα αυτό η διεργασία παιδί πραγματοποιεί εγγραφή στον buffer. Παρατηρούμε ότι μετά την εγγραφή η φυσική διεύθυνση του buffer του παιδιού αλλάζει, μιας και ο buffer είναι private οπότε η αλλαγές της διεργασίας-παιδιού δεν πρέπει να επηρεάζουν το περιεχόμενο του buffer για τη διεργασία-πατέρα.

```
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?

Father Process:

Information for the VA of the buffer:
7f56308c6000-7f56308c7000 rw-s 00000000 00:04 2152924                    /dev/zero (deleted)
VA[0x7f56308c6000] is not mapped; no physical memory allocated.


Information for the PA of the buffer: 0


Child Process:

Information for the VA of the buffer:
7f56308c6000-7f56308c7000 rw-s 00000000 00:04 2152924                    /dev/zero (deleted)


Information for the PA of the buffer: 4450480128
```

Στο ερώτημα αυτό η διεργασία παιδί πραγματοποιεί εγγραφή στον κοινό buffer. Βλέπουμε ότι για τη διεργασία παιδί ο shared buffer αντιστοιχίζεται στην φυσική διεύθυνση 450480128. Όμως η διεργασία πατέρας , λόγω του copy on write, δεν εντοπίζει την αντιστοίχιση. Για να γίνει αυτό πρέπει και η διεργασία-πατέρας να επιχειρήσει να προσπελάσει τον buffer. Μάλιστα δοκιμάσαμε να γράψουμε στον shared buffer από τη διεργασία-πατέρα και να εκτυπώσουμε την φυσική διεύθυνση του. Παρατηρήσαμε ότι σε αντίθεση με τον private buffer, η τιμή της ήταν ίδια με αυτή της διεργασίας παιδιού. Όμως μιας και αυτό δεν συμφωνεί με τα ζητούμενα της άσκησης ο παραδοτέος κώδικας δεν εκτελεί την λειτουργία αυτή.

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

Father Process:

Information for the VA of the buffer:
7f56308c6000-7f56308c7000 rw-s 00000000 00:04 2152924                    /dev/zero (deleted)
VA[0x7f56308c6000] is not mapped; no physical memory allocated.


Information for the PA of the buffer: 0


Child Process:

Information for the VA of the buffer:
7f56308c6000-7f56308c7000 r--s 00000000 00:04 2152924                    /dev/zero (deleted)


Information for the PA of the buffer: 4450480128
```

Τέλος καλούμαστε να αφαιρέσουμε το δικαίωμα εγγραφής στον shared buffer για τη διεργασία παιδί. Παρατηρούμε ότι στην απεικόνιση αλλάζει η τιμή της δεύτερης στήλης (permissions) και από rw δηλαδή ανάγνωση και εγγραφή (read and write) έχει αλλάξει σε r δηλαδή μόνο ανάγνωση (read). Αυτό δεν επηρεάζει τα δικαιώματα της διεργασίας-πατέρα, που συνεχίζει να έχει δικαίωμα ανάγνωσης και εγγραφής (rw).

Τέλος αποδεσμεύουμε τους buffers εκτελώντας την κλήση συστήματος munmap().

Άσκηση 1.2.1 :

Στην άσκηση αυτή καλούμαστε να υπολογίσουμε το σύνολο Mandelbrot χρησιμοποιώντας ένα πλήθος διεργασιών. Για το συγχρονισμό θα χρησιμοποιήσουμε ένα σύνολο από σημαφόρους ισάριθμο με το πλήθος των διεργασιών.

Η διαφορά με την προηγούμενη υλοποίηση όμως έγκειται στο γεγονός ότι τώρα οι σημαφόροι βρίσκονται αποθηκευμένοι σε κοινό (shared) τμήμα της μνήμης, ούτως ώστε να είναι προσβάσιμοι από τις διεργασίες.

Πιο συγκεκριμένα, όπως ακριβώς και με τα threads κάθε διεργασία υπολογίζει μια σειρά του πίνακα και έπειτα περιμένει να ενεργοποιηθεί ο σημαφόρος της, για να μπει στο κρίσιμο σημείο και να εκτυπώσει το αποτέλεσμα. Αφού εκτυπώσει, ενεργοποιεί τον σημαφόρο της επόμενης γραμμής και επαναλαμβάνει τη διαδικασία μέχρις ότου να εκτυπώσει τις γραμμές που της αναλογούν.

Παρακάτω φαίνεται ο κώδικας που υλοποιεί τα παραπάνω:

Αρχείο mandel-fork.c

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

#include <sys/mman.h>
#include <sys/types.h>
#include <sys/wait.h>

#include <semaphore.h>

#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

/*TODO header file for m(un)map*/

#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;

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

struct Proc_Card {
      int turn,NPROCS;
      pid_t p;
      sem_t * arr;
};

typedef struct Proc_Card * proc_ptr;
sem_t *sema;
```

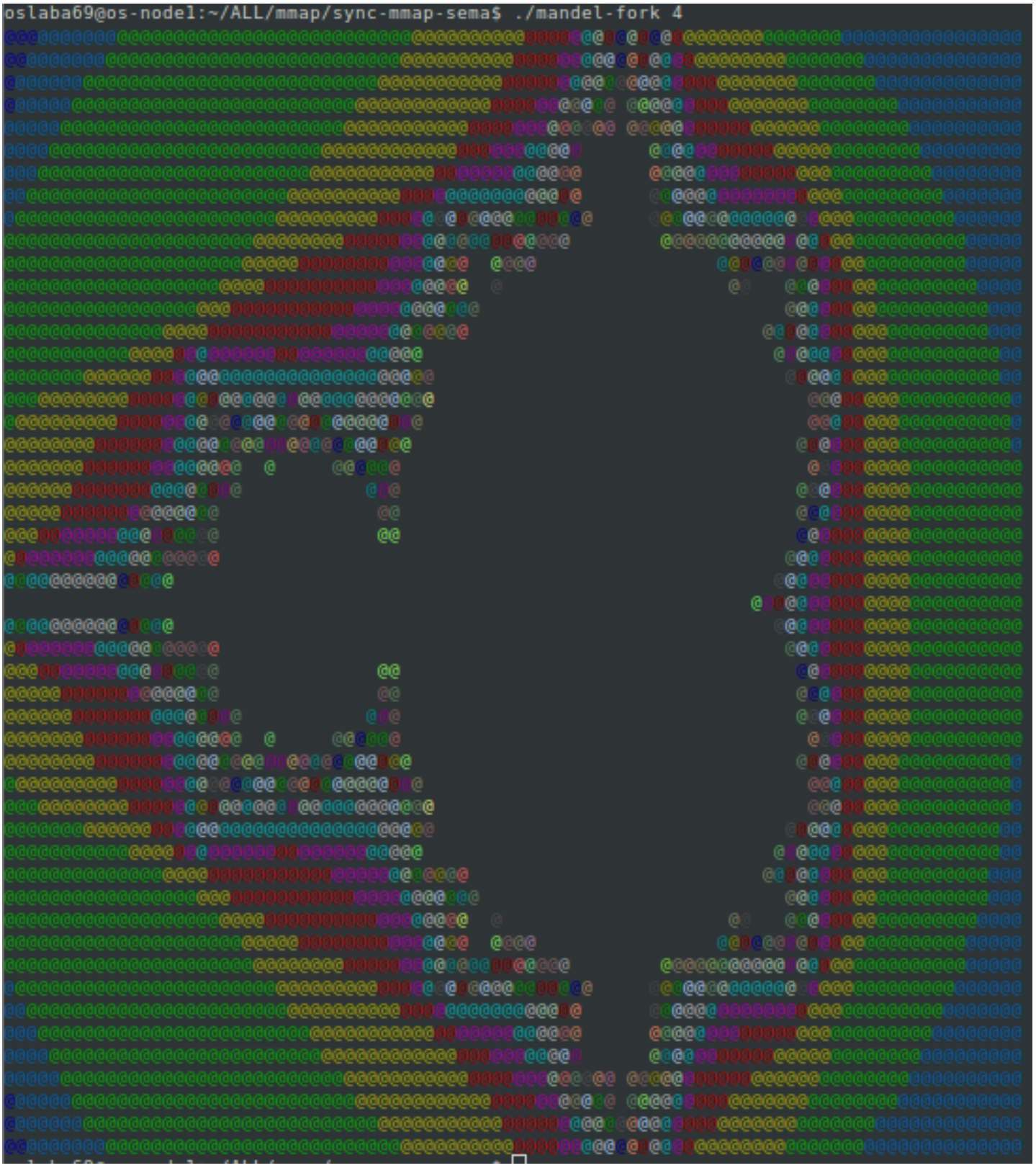
```c
int safe_atoi(char *s, int *val){
	long l;
	char *endp;

	l = strtol(s,&endp,10);
	if( s!=endp && *endp =='\0'){
		*val = l;
		return 0;
	}
	else return -1;
}


void * safe_malloc(size_t size){
	void *p;

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




void usage(char *argv0) {
	fprintf(stderr, "Usage: %s proc_count array_size\n\n"
		"Exactly one argument required:\n"
		" proc_count: The number of processes to create.\n",
		argv0);
	exit(1);
}

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

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

	proc_ptr ptr=(proc_ptr) p;

	int turn = (ptr)->turn;
	int fd = 1;
	int line;
	int Num_Procs = (ptr)->NPROCS;
	sem_t * arr = (ptr)->arr;

	for(line = turn; line < y_chars; line+=Num_Procs){

	  compute_mandel_line(line , color_val);
	  sem_wait(&arr[(line)%Num_Procs]); //wait until one thread wakes you

	  //Start of Critical Seciton

	  output_mandel_line(fd , color_val);  //print the your line
	  //End of Critical Section
	  //printf("Process: %d,inside critical,after print\n",line);

	  sem_post(&arr[(line +1)% Num_Procs]);  //wake the thread for the next line
		//printf("Process: %d,outside critical\n",line);

	}

	exit(1);
}
```

```
/*
 * Create a shared memory area, usable by all descendants of the calling
 * process.
 */
#define PROT (PROT_READ | PROT_WRITE)
#define FLAGS (MAP_SHARED | MAP_ANONYMOUS)

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;


	if((addr = mmap(NULL,pages*sysconf(_SC_PAGE_SIZE),
			PROT,FLAGS ,-1,0))==MAP_FAILED){
		perror("Problem with memory");
		exit(-1);
	}

	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[])
{
	int line,NPROCS;
	proc_ptr ptr;
	pid_t p;
	if (argc != 2) usage(argv[0]);
	if (safe_atoi(argv[1], &NPROCS) < 0 || NPROCS <= 0) { //convert input string to
//integer by using "atoi"
		fprintf(stderr, "`%s' is not valid for `process_count'\n", argv[1]);
		exit(1);
	}
```

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

	if (NPROCS > y_chars ){//if the threads are more than the lines then compress them
		NPROCS = y_chars;
	}


	ptr = safe_malloc(NPROCS * sizeof(*ptr));


	sema = create_shared_memory_area(NPROCS * sizeof(*sema)); //semaphroes array

	unsigned i;

	for (i = 1; i < NPROCS; ++i){//for loop that initializes all apart from the first
//semaphore

		if((sem_init(&sema[i],1,0) == -1)){
			fprintf(stderr,"Error with semaphores initialization");

		}
	}
	sem_init(&sema[0],1,1);//semaphore of first thread is equal to one so it can be in
//critical section
	//prints(sema,NPROCS);


	//printf("Before fork\n");

	for (i = 0; i < NPROCS; ++i){

		if((p = fork())<0){
			perror("fork");
			exit(-1);
		}

		if(p > 0){
			continue;
		}

		if(p==0){
			ptr[i].NPROCS = NPROCS;
			ptr[i].turn = i;
			ptr[i].arr = sema;

			compute_and_output_mandel_line((void*) (ptr + i));
		}
	}
	/*
	 * draw the Mandelbrot Set, one line at a time.
	 * Output is sent to file descriptor '1', i.e., standard output.
	 */
	int status;

	for (i = 0; i < NPROCS; ++i)//wait to join
		wait(&status);

	for (i = 0; i < NPROCS; ++i)//destroy every semaphore
		sem_destroy(&sema[i]);

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

Έξοδος Εκτελέσιμου *mandel-fork*



Ερωτήσεις:

1.

Στο κομμάτι της εκτέλεσης οι δυο υλοποιήσεις είναι ισοδύναμες. Η χρήση κοινής μνήμης για την αποθήκευση των σημαφόρων κάνει τις διεργασίες να συμπεριφέρονται ως threads.
Όμως μια καθυστέρηση εις βάρος των διεργασιών είναι αναμενόμενη , μιας και είναι πιο αργές στη δημιουργία τους σε σύγκριση με τα threads. Αυτό συμβαίνει επειδή η εξ ορισμού κοινή μνήμη των threads δεν χεριάζετε αντιγραφές δεδομένων όπως συμβαίνει στης διεργασίες, επιταχύνοντας έτσι τη δημιουργία τους. Αξίζει να σημειωθεί βέβαια ότι στο δικό μας παράδειγμα, ακόμα και αν χρησιμοποιούσαμε το μέγιστο πλήθος διεργασιών η διαφορά ήταν αμελητέα. Όμως σε περιπτώσεις όπου το πλήθος των διεργασιών είναι πολύ μεγαλύτερο αναμένετε σημαντική διαφορά.

Άσκηση 1.2.2. :

Στην τελευταία άσκηση καλούμαστε να υπολογίσουμε ξανά το Mandelbrot χρησιμοποιώντας ένα πλήθος διεργασιών. Ωστόσο τώρα χωρίς την χρήση των σημαφόρων, και ειδικότερα θα δεσμεύσουμε έναν πίνακα στην μνήμη όπου κάθε διεργασία αφού υπολογίζει την γραμμή που της αντιστοιχεί τότε θα πηγαίνει στην κατάλληλη θέση στον πίνακα Buffer[γραμμές*στήλες] για να αποθηκεύσει σε διαδοχικές θέσεις μνήμης το αποτέλεσμα της.

Επομένως , αφού υπολογιστούν όλες οι γραμμές η πατρική διεργασία θα διατρέξει τον πίνακα εκτυπώνοντας το αποτέλεσμα/εικόνα στο τερματικό μας (standard output) . Άρα, σε αυτήν την υλοποίηση δεν απαιτείται άμεσος συγχρονισμός μεταξύ των διεργασιών κατά το τύπωμα μια γραμμής καθώς μια τυχαία διεργασία έχει την ελευθερία να αποθηκεύσει τον πίνακα της (που εμπεριέχει έναν αριθμό για κάθε χαρακτήρα της γραμμής που αντιστοιχίζει στο χρώμα του) στο διαμοιραζόμενο χώρος μνήμης όποτε αυτή είναι έτοιμη ,καθώς το τύπωμα στο τέλος θα γίνει αποκλειστικά από μια διεργασία και δε χρειάζεται κάθε διεργασία να περιμένει να εκτυπώσει αυτό που υπολογίζει (άρα συγχρονισμός μέσω της κλήσης συστήματος ***wait()*** που εκτελεί η πατρική διεργασία για κάθε παιδί της).

Οπότε, τώρα παραθέτουμε τον απαιτούμενο κώδικα ώστε να επιτύχουμε το ζητούμενο αποτέλεσμα.

Αρχείο mandel-fork.c

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

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

struct Proc_Card {
      int turn,NPROCS;
      pid_t p;
      int * arr;
};
typedef struct Proc_Card * proc_ptr;

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;
}
```

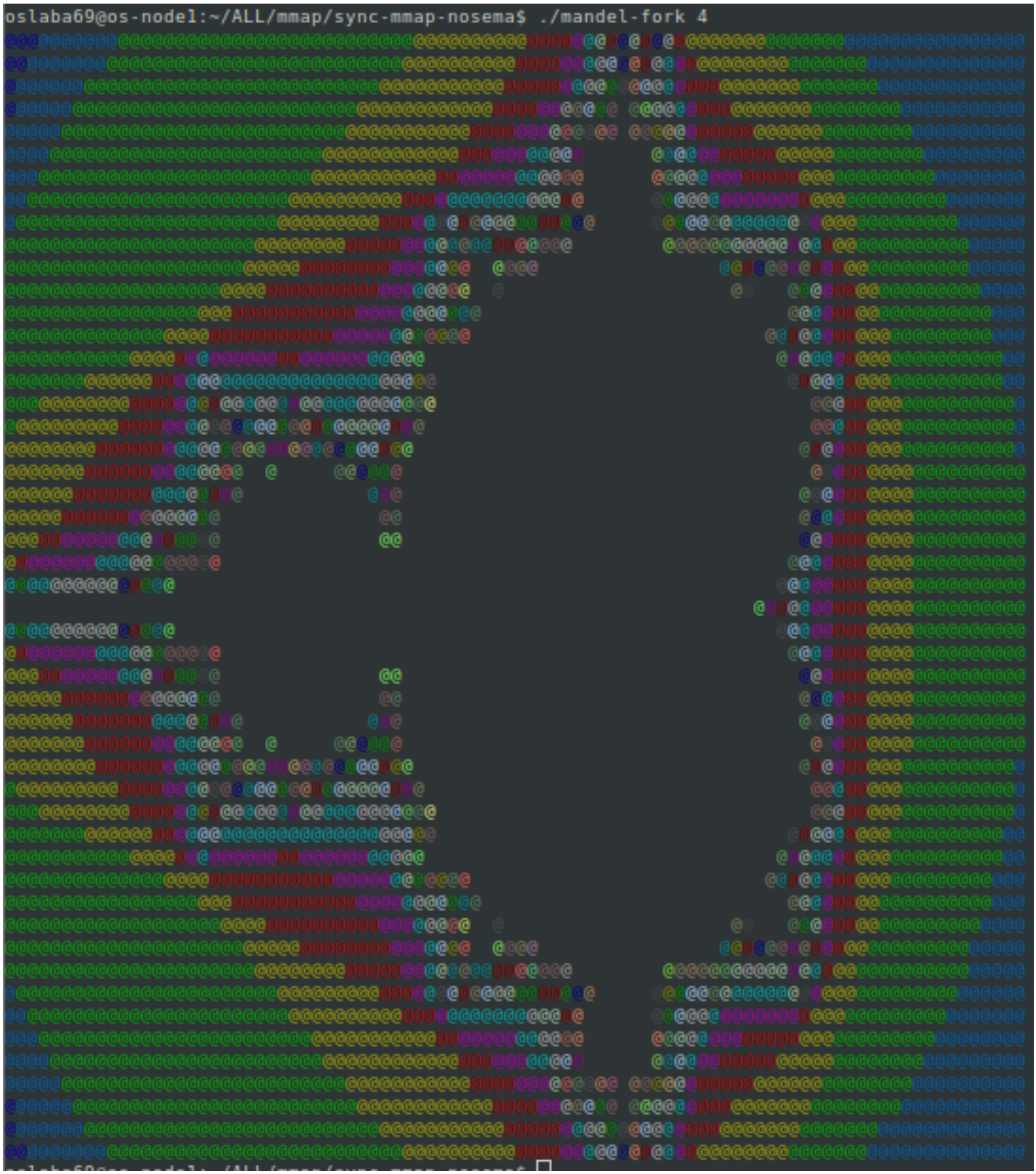
```
void * safe_malloc(size_t size){
	void *p;

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



void usage(char *argv0) {
	fprintf(stderr, "Usage: %s proc_count array_size\n\n"
		"Exactly one argument required:\n"
		" proc_count: The number of processes to create.\n",
		argv0);
	exit(1);
}

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

	int 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(void * p)
{
	/*
	 * A temporary array, used to hold color values for the line being drawn
	 */
	proc_ptr ptr=(proc_ptr) p;
	int turn = (ptr)->turn,line,
	Num_Procs = (ptr)->NPROCS,
	*arr=(ptr)->arr;

	for(line = turn; line < y_chars; line+=Num_Procs){
		compute_mandel_line(line,arr + line*x_chars);
	}
	exit(1);
}

/*
 * Create a shared memory area, usable by all descendants of the calling
 * process.
 */

#define PROTECT PROT_READ | PROT_WRITE
#define FLAGS MAP_SHARED | MAP_ANONYMOUS
#define LENGTH pages*sysconf(_SC_PAGE_SIZE)
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;


	if((addr = mmap(NULL,LENGTH,PROTECT, FLAGS,-1,0))==MAP_FAILED){
		perror("Problem with memory");
		exit(-1);
	}

	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, LENGTH) == -1) {
		perror("destroy_shared_memory_area: munmap failed");
		exit(1);
	}
}

int main(int argc,char *argv[])
{

	int line,NPROCS;
	proc_ptr ptr;
	pid_t p;

	if (argc != 2) usage(argv[0]);

	if (safe_atoi(argv[1], &NPROCS) < 0 || NPROCS <= 0) { //convert input string to
integer by using "atoi"
		fprintf(stderr, "`%s' is not valid for `process_count'\n", argv[1]);
		exit(1);
	}


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

	if (NPROCS > y_chars ){//if the threads are more than the lines then compress them
		NPROCS = y_chars;
	}


	ptr = safe_malloc(NPROCS * sizeof(*ptr));


	int * buffer = create_shared_memory_area(sizeof(int)*y_chars*x_chars);

	unsigned i;
```

```
	for (i = 0; i < NPROCS; ++i){

		if((p = fork())<0){
			perror("fork");
			exit(-1);
		}

		if(p > 0){
			continue;
		}

		if(p==0){
			ptr[i].NPROCS = NPROCS;
			ptr[i].turn = i;
			ptr[i].arr = buffer;

			compute_and_output_mandel_line((void*) (ptr + i));
		}
	}

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

	int status;

	for (i = 0; i < NPROCS; ++i)
		wait(&status);

	for (line = 0; line < y_chars; line++)
		output_mandel_line(1 ,(int *) (buffer + line * x_chars) );

	reset_xterm_color(1);


	 destroy_shared_memory_area(buffer,sizeof(int)*y_chars*x_chars );


	return 0;
}
```

Έξοδος Εκτελέσιμου *mandel-fork*



Ερωτήσεις:

1.

Στην τρέχουσα υλοποίηση δεν εμφανίζεται η ανάγκη για άμεσο συγχρονισμό μεταξύ των διεργασιών αλλά γίνεται με έμμεσο τρόπο. Πιο συγκεκριμένα, η κάθε διεργασία δεν χρειάζεται να περιμένει κάποια άλλη διεργασία για να εκτυπώσει το αποτέλεσμα της καθώς αυτό δεν εκτυπώνεται ακαριαία στο τερματικό μας αλλά αποθηκεύεται σε συγκεκριμένη θέση μέσα στον διαμοιραζόμενο χώρο μνήμης όπου στο τέλος η πατρική διεργασία ,γνωρίζοντας ότι τα παιδιά της έχουν τερματίσει, αναλαμβάνει το έργο της εκτύπωσης (συγχρονισμός έμμεσα μέσω του ***wait()***).

Τώρα, στην περίπτωση που ο διαμοιραζόμενο χώρος μνήμης δεν έχει την ικανότητα να εμπεριέχει ολόκληρη την εικόνα ($y\_chars * x\_chars$) αλλά το μέρος $NPROCS * x\_chars$,το σχήμα συγχρονισμού απαιτεί αλλαγή.

Πιο συγκεκριμένα, υπάρχει αυτονομία μεταξύ των $NPROCS$ γραμμών όπου αφού υπολογιστούν και εκτυπωθούν από την πατρική τότε επαναλαμβάνεται η ίδια διαδικασία έως ότου υπολογιστούν όλες οι γραμμές, δηλαδή απαιτούνται συνολικά $\boldsymbol{times} = \left\lceil \frac{\boldsymbol{y\_chars}}{\boldsymbol{NPROCS}} \right\rceil$ εκτυπώσεις του buffer. Να σημειωθεί ότι μόλις μια γραμμή υπολογισθεί τότε η αντίστοιχη διεργασία σταματάει τη ροή της μέσω της κλήσης συστήματος $\boldsymbol{kill}()$ και περιμένει από την πατρική διεργασία σήμα ώστε να ξανά μπει στο βρόχο υπολογισμού των γραμμών .

Άρα έχουμε για τον πατέρα,

```
for (i = 1; i <= times; ++i){

      for (j = 0; j < NPROCS; ++j)//wait for every process to stop
            wait(&status);

      for (j = 0; (j < NPROC) && (((i*times) + (j + 1)) <= y_chars); ++j)
            output_mandel_line(1 ,(int *) (buffer + j* x_chars) );
//print the buffer in stdout

      for (j = 0; j < NPROC; ++j)
            kill(arr_pid[i],SIGCONT);//awake all processes to fill again
//the buffer

}

//arr_pid: an array with length=NPROCS that each element is the
//pid of child
```

Ενώ για το παιδί είναι

```
for(line = turn; line < y_chars; line+=Num_Procs){

     compute_mandel_line(line,arr + (line % NPROCS)* x_chars);

     kill(getpid(),SIGSTOP);

}
```