ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ ΥΠΟΛΟΓΙΣΤΩΝ
ΑΣΚΗΣΗ 4
Μηχανισμοί Εικονικής Μνήμης

Ομάδα oslab39
Δημήτριος Βασιλείου el19830
Ιωάννης Ρόκομος el19061

ΑΣΚΗΣΗ 1.1
Ο πηγαίος κώδικας φαίνεται παρακάτω:

```
/*
 * 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 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 buffer_address_child =
get_physical_address(heap_private_buf);
         printf("Buffer address(from child): %p\n",
buffer_address_child);

        /*
         * Step 9 - Child
         */
        if (0 != raise(SIGSTOP))
                die("raise(SIGSTOP)");
        /*
         * TODO: Write your code here to complete child's part of
Step 9.
         */
         heap_private_buf[0] = 42;
         buffer_address_child =
get_physical_address(heap_private_buf);
         printf("Private buffer address(from child): %p\n",
buffer_address_child);

        /*
         * Step 10 - Child
         */
        if (0 != raise(SIGSTOP))
                die("raise(SIGSTOP)");
        /*
         * TODO: Write your code here to complete child's part of
Step 10.
         */
```

```
        //heap_shared_buf = (char*)malloc(buffer_size *
sizeof(char));

        heap_shared_buf[0] = 42;

        uint64_t shared_buffer_address_child =
get_physical_address(heap_shared_buf);
        printf("Shared buffer address(from child): %p\n",
shared_buffer_address_child);

        /*
         * 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("Shared buffer(from child)\n");
        show_va_info(heap_shared_buf);
        //show_maps();
        /*
         * Step 12 - Child
         */
        /*
         * TODO: Write your code here to complete child's part of
Step 12.
         */

        munmap(heap_private_buf, buffer_size);
        munmap(heap_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 "\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("Parent 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 "\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.
         */
         uint64_t buffer_address_parent =
get_physical_address(heap_private_buf);
         printf("Buffer address(from parent): %p\n",
buffer_address_parent);

        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.
         */
         buffer_address_parent =
get_physical_address(heap_private_buf);
         printf("Private buffer address(from parent): %p\n",
buffer_address_parent);

        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 shared_buffer_address_parent =
get_physical_address(heap_shared_buf);
         printf("Shared buffer address(from parent): %p\n",
shared_buffer_address_parent);


        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.
         */
        printf("Shared buffer(from parent)\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);
}

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 "\nStep 1: Print the virtual address space map
of this "
                "process [%d].\n" RESET, mypid);
        press_enter();
        /*
         * TODO: Write your code here to complete Step 1.
         */
        show_maps();

        /*
         * Step 2: Use mmap to allocate a buffer of 1 page and
print the map
         * again. Store buffer in heap_private_buf.
```

```
         */
        printf(RED "\nStep 2: Use mmap(2) to allocate a private
buffer of "
                "size equal to 1 page and print the VM map again.\
n" RESET);
        press_enter();
        /*
         * TODO: Write your code here to complete Step 2.
         */
         heap_private_buf = mmap(NULL, buffer_size, PROT_WRITE |
PROT_READ, MAP_PRIVATE | MAP_ANONYMOUS, fd, 0);
         show_maps();
        show_va_info(heap_private_buf);
        /*
         * Step 3: Find the physical address of the first page of
your buffer
         * in main memory. What do you see?
         */
        printf(RED "\nStep 3: Find and print the physical address
of the "
                "buffer in main memory. What do you see?\n"
RESET);
        press_enter();
        /*
         * TODO: Write your code here to complete Step 3.
         */

         uint64_t buffer_address =
get_physical_address(heap_private_buf);
         printf("Buffer address: %p", buffer_address);

        /*
         * Step 4: Write zeros to the buffer and repeat Step 3.
         */
        printf(RED "\nStep 4: Initialize your buffer with zeros
and repeat "
                "Step 3. What happened?\n" RESET);
        press_enter();
        /*
         * TODO: Write your code here to complete Step 4.
         */
         //heap_private_buf = (char*)malloc(buffer_size *
sizeof(char));
         int i;
         for (i = 0; i < buffer_size; i++) {
             heap_private_buf[i] = 0;
         }
         buffer_address = get_physical_address(heap_private_buf);
         printf("Buffer address: %p", buffer_address);

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

```
         * its content. Use file_shared_buf.
         */
        printf(RED "\nStep 5: Use mmap(2) to read and print
file.txt. Print "
                "the new mapping information that has been
created.\n" RESET);
        press_enter();
        /*
         * TODO: Write your code here to complete Step 5.
         */
        fd = open("file.txt", O_RDONLY);
        file_shared_buf = mmap(NULL, buffer_size, PROT_READ,
MAP_PRIVATE, fd, 0);
        int j = 0;

        do {
                printf("%c", file_shared_buf[j++]);
        } while (file_shared_buf[j] != '\0');

        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();
        /*
         * TODO: Write your code here to complete Step 6.
         */
        heap_shared_buf = mmap(NULL, buffer_size, PROT_WRITE |
PROT_READ, MAP_SHARED | MAP_ANONYMOUS, fd, 0);
        show_maps();
        show_va_info(heap_shared_buf);
        /* Initialize buffer so that it can be "seen" by the
memory map */
        heap_shared_buf[0] = 10;

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

```
        parent(p);

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

Η έξοδος εκτέλεσης του προγράμματος φαίνεται παρακάτω:

```
Step 1: Print the virtual address space map of this process [21171].


Virtual Memory Map of process [21171]:
00400000-00403000 r-xp 00000000 00:21 9451715                            /home/oslab/oslaba39/Ask4/mmap/mmapTest
00602000-00603000 rw-p 00002000 00:21 9451715                            /home/oslab/oslaba39/Ask4/mmap/mmapTest
01efd000-01fle000 rw-p 00000000 00:00 0                                  [heap]
7fb70d293000-7fb70d434000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d434000-7fb70d634000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d634000-7fb70d638000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d638000-7fb70d63a000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d63a000-7fb70d63e000 rw-p 00000000 00:00 0
7fb70d63e000-7fb70d65f000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d851000-7fb70d854000 rw-p 00000000 00:00 0
7fb70d859000-7fb70d85e000 rw-p 00000000 00:00 0
7fb70d85e000-7fb70d85f000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d85f000-7fb70d860000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d860000-7fb70d861000 rw-p 00000000 00:00 0
7ffeb0ba3000-7ffeb0bc4000 rw-p 00000000 00:00 0                          [stack]
7ffeb0bf7000-7ffeb0bfa000 r--p 00000000 00:00 0                          [vvar]
7ffeb0bfa000-7ffeb0bfc000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------


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


Virtual Memory Map of process [21171]:
00400000-00403000 r-xp 00000000 00:21 9451715                            /home/oslab/oslaba39/Ask4/mmap/mmapTest
00602000-00603000 rw-p 00002000 00:21 9451715                            /home/oslab/oslaba39/Ask4/mmap/mmapTest
01efd000-01fle000 rw-p 00000000 00:00 0                                  [heap]
7fb70d293000-7fb70d434000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d434000-7fb70d634000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d634000-7fb70d638000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d638000-7fb70d63a000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d63a000-7fb70d63e000 rw-p 00000000 00:00 0
7fb70d63e000-7fb70d65f000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d851000-7fb70d854000 rw-p 00000000 00:00 0
7fb70d858000-7fb70d85e000 rw-p 00000000 00:00 0
7fb70d85e000-7fb70d85f000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d85f000-7fb70d860000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d860000-7fb70d861000 rw-p 00000000 00:00 0
7ffeb0ba3000-7ffeb0bc4000 rw-p 00000000 00:00 0                          [stack]
7ffeb0bf7000-7ffeb0bfa000 r--p 00000000 00:00 0                          [vvar]
7ffeb0bfa000-7ffeb0bfc000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------

7fb70d858000-7fb70d85e000 rw-p 00000000 00:00 0

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

VA[0x7fb70d859000] is not mapped; no physical memory allocated.
Buffer address: (nil)
Step 4: Initialize your buffer with zeros and repeat Step 3. What happened?

Buffer address: 0xb8170000
```

```
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 [21171]:
00400000-00403000 r-xp 00000000 00:21 9451715                            /home/oslab/oslaba39/Ask4/mmap/mmapTest
00602000-00603000 rw-p 00002000 00:21 9451715                            /home/oslab/oslaba39/Ask4/mmap/mmapTest
01efd000-01fle000 rw-p 00000000 00:00 0                                  [heap]
7fb70d293000-7fb70d434000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d434000-7fb70d634000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d634000-7fb70d638000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d638000-7fb70d63a000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d63a000-7fb70d63e000 rw-p 00000000 00:00 0
7fb70d63e000-7fb70d65f000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d851000-7fb70d854000 rw-p 00000000 00:00 0
7fb70d857000-7fb70d858000 rw-p 00000000 00:00 0
7fb70d858000-7fb70d859000 r--p 00000000 00:21 9451646                    /home/oslab/oslaba39/Ask4/mmap/file.txt
7fb70d859000-7fb70d85e000 rw-p 00000000 00:00 0
7fb70d85e000-7fb70d85f000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d85f000-7fb70d860000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d860000-7fb70d861000 rw-p 00000000 00:00 0
7ffeb0ba3000-7ffeb0bc4000 rw-p 00000000 00:00 0                          [stack]
7ffeb0bf7000-7ffeb0bfa000 r--p 00000000 00:00 0                          [vvar]
7ffeb0bfa000-7ffeb0bfc000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------


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 [21171]:
00400000-00403000 r-xp 00000000 00:21 9451715                            /home/oslab/oslaba39/Ask4/mmap/mmapTest
00602000-00603000 rw-p 00002000 00:21 9451715                            /home/oslab/oslaba39/Ask4/mmap/mmapTest
01efd000-01fle000 rw-p 00000000 00:00 0                                  [heap]
7fb70d293000-7fb70d434000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d434000-7fb70d634000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d634000-7fb70d638000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d638000-7fb70d63a000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d63a000-7fb70d63e000 rw-p 00000000 00:00 0
7fb70d63e000-7fb70d65f000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d851000-7fb70d854000 rw-p 00000000 00:00 0
7fb70d856000-7fb70d857000 rw-p 00000000 00:00 0
7fb70d857000-7fb70d858000 rw-s 00000000 00:04 3440895                    /dev/zero (deleted)
7fb70d858000-7fb70d859000 r--p 00000000 00:21 9451646                    /home/oslab/oslaba39/Ask4/mmap/file.txt
7fb70d859000-7fb70d85e000 rw-p 00000000 00:00 0
7fb70d85e000-7fb70d85f000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d85f000-7fb70d860000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d860000-7fb70d861000 rw-p 00000000 00:00 0
7ffeb0ba3000-7ffeb0bc4000 rw-p 00000000 00:00 0                          [stack]
7ffeb0bf7000-7ffeb0bfa000 r--p 00000000 00:00 0                          [vvar]
7ffeb0bfa000-7ffeb0bfc000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------

7fb70d857000-7fb70d858000 rw-s 00000000 00:04 3440895                    /dev/zero (deleted)
```

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

Parent map

Virtual Memory Map of process [21171]:
00400000-00403000 r-xp 00000000 00:21 9451715                            /home/oslab/oslaba39/Ask4/mmap/mmapTest
00602000-00603000 rw-p 00002000 00:21 9451715                            /home/oslab/oslaba39/Ask4/mmap/mmapTest
01efd000-01fle000 rw-p 00000000 00:00 0                                  [heap]
7fb70d293000-7fb70d434000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d434000-7fb70d634000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d634000-7fb70d638000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d638000-7fb70d63a000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d63a000-7fb70d63e000 rw-p 00000000 00:00 0
7fb70d63e000-7fb70d65f000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d851000-7fb70d854000 rw-p 00000000 00:00 0
7fb70d856000-7fb70d857000 rw-p 00000000 00:00 0
7fb70d857000-7fb70d858000 rw-s 00000000 00:04 3440895                    /dev/zero (deleted)
7fb70d858000-7fb70d859000 r--p 00000000 00:21 9451646                    /home/oslab/oslaba39/Ask4/mmap/file.txt
7fb70d859000-7fb70d85e000 rw-p 00000000 00:00 0
7fb70d85e000-7fb70d85f000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d85f000-7fb70d860000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d860000-7fb70d861000 rw-p 00000000 00:00 0
7ffeb0ba3000-7ffeb0bc4000 rw-p 00000000 00:00 0                          [stack]
7ffeb0bf7000-7ffeb0bfa000 r--p 00000000 00:00 0                          [vvar]
7ffeb0bfa000-7ffeb0bfc000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------

Child map

Virtual Memory Map of process [21172]:
00400000-00403000 r-xp 00000000 00:21 9451715                            /home/oslab/oslaba39/Ask4/mmap/mmapTest
00602000-00603000 rw-p 00002000 00:21 9451715                            /home/oslab/oslaba39/Ask4/mmap/mmapTest
01efd000-01fle000 rw-p 00000000 00:00 0                                  [heap]
7fb70d293000-7fb70d434000 r-xp 00000000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d434000-7fb70d634000 ---p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d634000-7fb70d638000 r--p 001a1000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d638000-7fb70d63a000 rw-p 001a5000 08:01 6032227                    /lib/x86_64-linux-gnu/libc-2.19.so
7fb70d63a000-7fb70d63e000 rw-p 00000000 00:00 0
7fb70d63e000-7fb70d65f000 r-xp 00000000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d851000-7fb70d854000 rw-p 00000000 00:00 0
7fb70d856000-7fb70d857000 rw-p 00000000 00:00 0
7fb70d857000-7fb70d858000 rw-s 00000000 00:04 3440895                    /dev/zero (deleted)
7fb70d858000-7fb70d859000 r--p 00000000 00:21 9451646                    /home/oslab/oslaba39/Ask4/mmap/file.txt
7fb70d859000-7fb70d85e000 rw-p 00000000 00:00 0
7fb70d85e000-7fb70d85f000 r--p 00020000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d85f000-7fb70d860000 rw-p 00021000 08:01 6032224                    /lib/x86_64-linux-gnu/ld-2.19.so
7fb70d860000-7fb70d861000 rw-p 00000000 00:00 0
7ffeb0ba3000-7ffeb0bc4000 rw-p 00000000 00:00 0                          [stack]
7ffeb0bf7000-7ffeb0bfa000 r--p 00000000 00:00 0                          [vvar]
7ffeb0bfa000-7ffeb0bfc000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
-----------------------------------------------------------


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

Buffer address(from parent): 0xb8170000
Buffer address(from child): 0xb8170000
```

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

Private buffer address(from parent): 0xb8170000
Private buffer address(from child): 0xa91df000

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?

Shared buffer address(from parent): 0xb77a4000
Shared buffer address(from child): 0xb77a4000

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

Shared buffer(from parent)
7fb70d857000-7fb70d858000 rw-s 00000000 00:04 3440895                    /dev/zero (deleted)
Shared buffer(from child)
7fb70d857000-7fb70d858000 r--s 00000000 00:04 3440895                    /dev/zero (deleted)
```

Παρατηρήσεις πάνω στα βήματα:

- Βήμα 3: Με την κλήση της συνάρτησης **`get_physical_address()`** για τον heap_private_buf προσπαθούμε να τυπώσουμε την φυσική διεύθυνση του. Ωστόσο, λαμβάνουμε μήνυμα πως η συγκεκριμένη σελίδα δεν έχει απεικονιστεί σε πλαίσιο φυσικής μνήμης. Αυτό συμβαίνει, διότι ο buffer δεν έχει αρχικοποιηθεί, συνεπώς το λειτουργικό σύστημα δεν έχει παραχωρήσει φυσική μνήμη για την συγκεκριμένη σελίδα.
- Βήμα 4: Σε αυτό το βήμα γεμίζουμε τον buffer με μηδενικά, δηλαδή τον αρχικοποιούμε. Επομένως το λειτουργικό σύστημα του παραχωρεί φυσική μνήμη και αυτή τυπώνεται στο output με την κλήση της συνάρτησης **`get_physical_address()`**.
- Βήμα 7: Τυπώνουμε τους χάρτες εικονικής μνήμης για τις διεργασίες πατέρα και παιδί με την συνάρτηση **`show_maps()`** και παρατηρούμε ότι ο χάρτης μνήμης της διεργασίας παιδιού είναι αντίγραφο του χάρτη της διεργασίας πατέρα. Αυτό συμβαίνει, διότι με την κλήση συστήματος **`fork()`** ένα αντίγραφο του χώρου εικονικών διευθύνσεων της διεργασίας πατέρα περνάει στην διεργασία παιδί.
- Βήμα 8: Παρατηρούμε ότι οι φυσικές διευθύνσεις του heap_private_buf είναι ίδιες μεταξύ των διεργασιών πατέρα-παιδί. Αυτό συμβαίνει, διότι με την κλήση συστήματος **`fork()`** ένα αντίγραφο του χώρου εικονικών διευθύνσεων της διεργασίας πατέρα περνάει στην διεργασία παιδί και οι φυσικές διευθύνσεις στις οποίες αντιστοιχίζονται αυτοί είναι ίδιες.
- Βήμα 9: Γράφοντας σε ένα στοιχείο του buffer από την διεργασία παιδί και τυπώνοντας στην συνέχεια τις φυσικές διευθύνσεις του από την διεργασία παιδί και την διεργασία πατέρα, παρατηρούμε ότι τώρα αυτές είναι διαφορετικές μεταξύ τους. Αυτό γίνεται εξαιτίας της τεχνικής copy-on-write. Οι διεργασίες πατέρας-παιδί αρχικά μοιράζονται τον ίδιο χώρο φυσικής μνήμης ωσότου μία από τις δύο τροποποιήσει το περιεχόμενο του buffer. Σε αυτό το σημείο ανατίθεται ένα νέο πλαίσιο φυσικής μνήμης για τον buffer της διεργασίας παιδιού.
- Βήμα 10: Γράφοντας σε ένα στοιχείο του heap_shared_buf από την διεργασία παιδί και τυπώνοντας στην συνέχεια τις φυσικές διευθύνσεις του από την διεργασία παιδί και την διεργασία πατέρα, παρατηρούμε ότι τώρα αυτές είναι ίδιες μεταξύ τους, σε αντίθεση με τον heap_private_buf. Αυτό συμβαίνει, διότι ορίσαμε τον heap_shared_buf καλώντας την **`mmap()`** με flag **`MAP_SHARED`**. Με αυτόν τον τρόπο η φυσική μνήμη στην οποία αντιστοιχεί ο heap_shared_buf είναι κοινή για όλες τις διεργασίες.

ΑΣΚΗΣΗ 1.2

1.2.1 Semaphores πάνω από διαμοιραζόμενη μνήμη

Ο πηγαίος κώδικας για την φαίνεται παρακάτω:

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

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

/*
 * POSIX thread functions do not return error numbers in errno,
 * but in the actual return value of the function call instead.
 * This macro helps with error reporting in this case.
 */
#define perror_pthread(ret, msg) \
        do { errno = ret; perror(msg); } while (0)

/***************************
* 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;

/* array of semaphores */
sem_t *mutex;

/*--------------Everythig for threads----------------*/

/*
 * A (distinct) instance of this structure
 * is passed to each thread
 */
struct process_info_struct {
        pid_t pid; /* process id, as returned by the library */

        int *color_val; /* Pointer to array to manipulate. Each
process manipulat
es a line and gives each character a color */

        int prid; /* Application-defined process id */
        int nprocs; /* Number of total processes*/
};

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 thread_count array_size\n\n"
                "Exactly two argument required:\n"
                "    thread_count: The number of threads to
create.\n"
                "    array_size: The size of the array to run
with.\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 *arg)
{
    int i;
    struct process_info_struct *proc = arg;
    int *color_val = proc->color_val;

    /* thread i manipulates lines i, i + n, i + 2n ....*/
    for (i = proc->prid; i < y_chars; i += proc->nprocs) {
        compute_mandel_line(i, color_val);
        sem_wait(&mutex[i % proc->nprocs]);
        output_mandel_line(1, color_val);
        sem_post(&mutex[(i + 1)% proc->nprocs]);
    }

    return NULL;
}

void *create_shared_memory_area(unsigned int numbytes)
{
        int pages;
        void *addr;

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

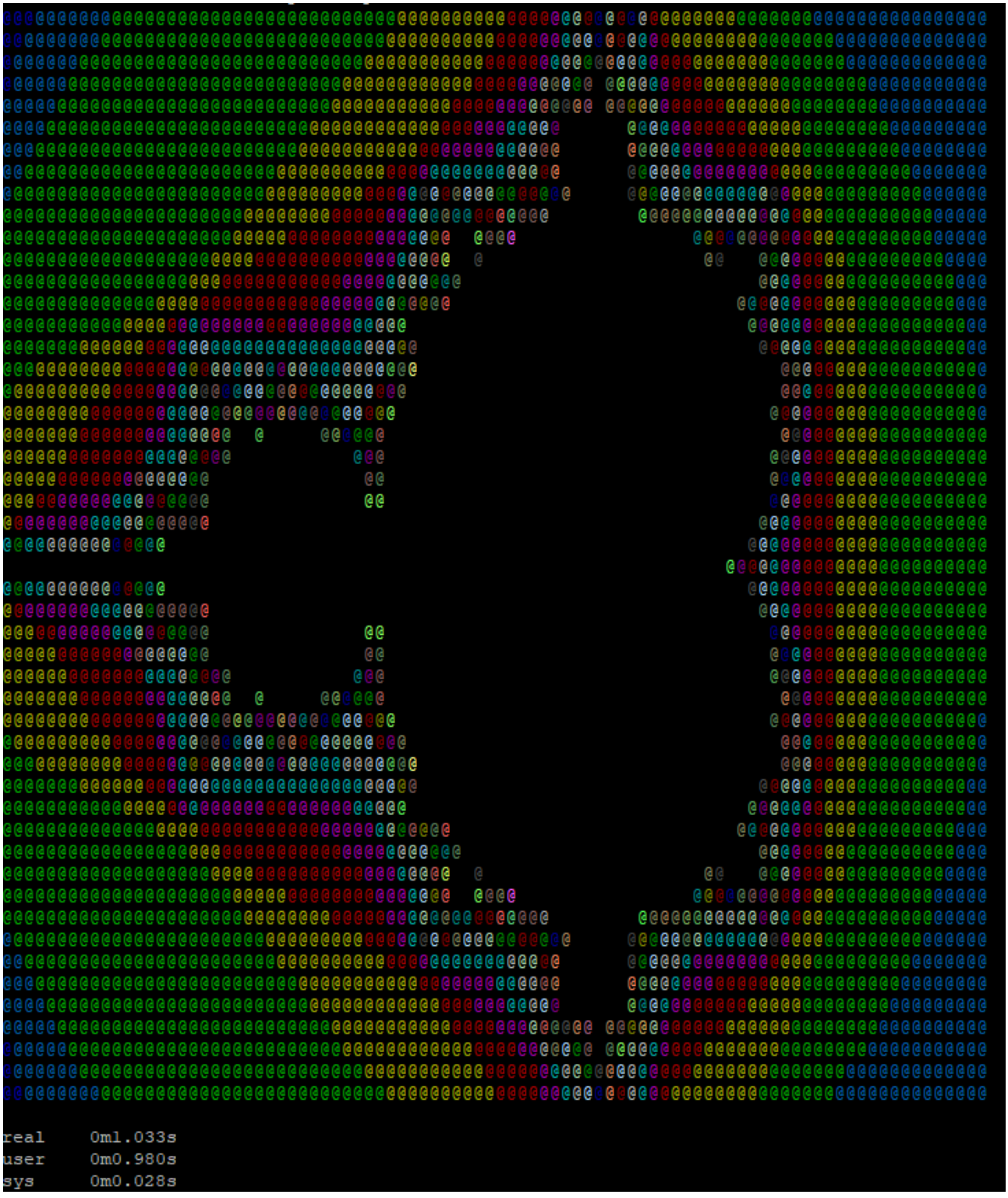
```
        /*
         * Determine the number of pages needed, round up the
requested number of
         * pages
         */
        pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;

        /* Create a shared, anonymous mapping for this number of
pages */
        /* TODO:
                addr = mmap(...)
        */
        addr = mmap(NULL, pages, PROT_READ | PROT_WRITE |
PROT_EXEC, MAP_ANONYMOUS | MAP_SHARED, -1, 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[])
{
    int i, nprocs, status;
    struct process_info_struct *proc;

    /*
     * Parse the command line. User gives number of processes
nprocs
```

```
     */
    if (argc != 2)
            usage(argv[0]);
    if (safe_atoi(argv[1], &nprocs) < 0 || nprocs <= 0) {
            fprintf(stderr, "`%s' is not valid for `thread_count'\
n", argv[1]);
            exit(1);
    }


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

    /* Create array of nprocs processes. Ecah process has a struct
with its info */
    proc = safe_malloc(nprocs * sizeof(*proc));

    mutex = create_shared_memory_area(sizeof(sem_t) * nprocs);

    /*
    * draw the Mandelbrot Set, one line at a time.
    * Output is sent to file descriptor '1', i.e., standard
output.
    */
        /* sem_init 1 to be shared between processes */
        sem_init(&mutex[0], 1, 1);
        for (i = 1; i < nprocs; i++) {
            sem_init(&mutex[i], 1, 0);
        }


        // create nprocs processes
        for (i = 0; i < nprocs; i++) {
            // create new process and initialize fields of process
struct
            proc[i].prid = i;
            proc[i].nprocs = nprocs;
            proc[i].color_val = safe_malloc(x_chars *
sizeof(int));
            proc[i].pid = fork();

            if (proc[i].pid < 0) {
                perror("Fork failed...:(");
                exit(1);
            }
            /* each child process manipulates
            the lines that corresponds to it */
            if(proc[i].pid == 0) {
                compute_and_output_mandel_line(&proc[i]);
                exit(1);
            }

        // parent process waits for the children to finish
```

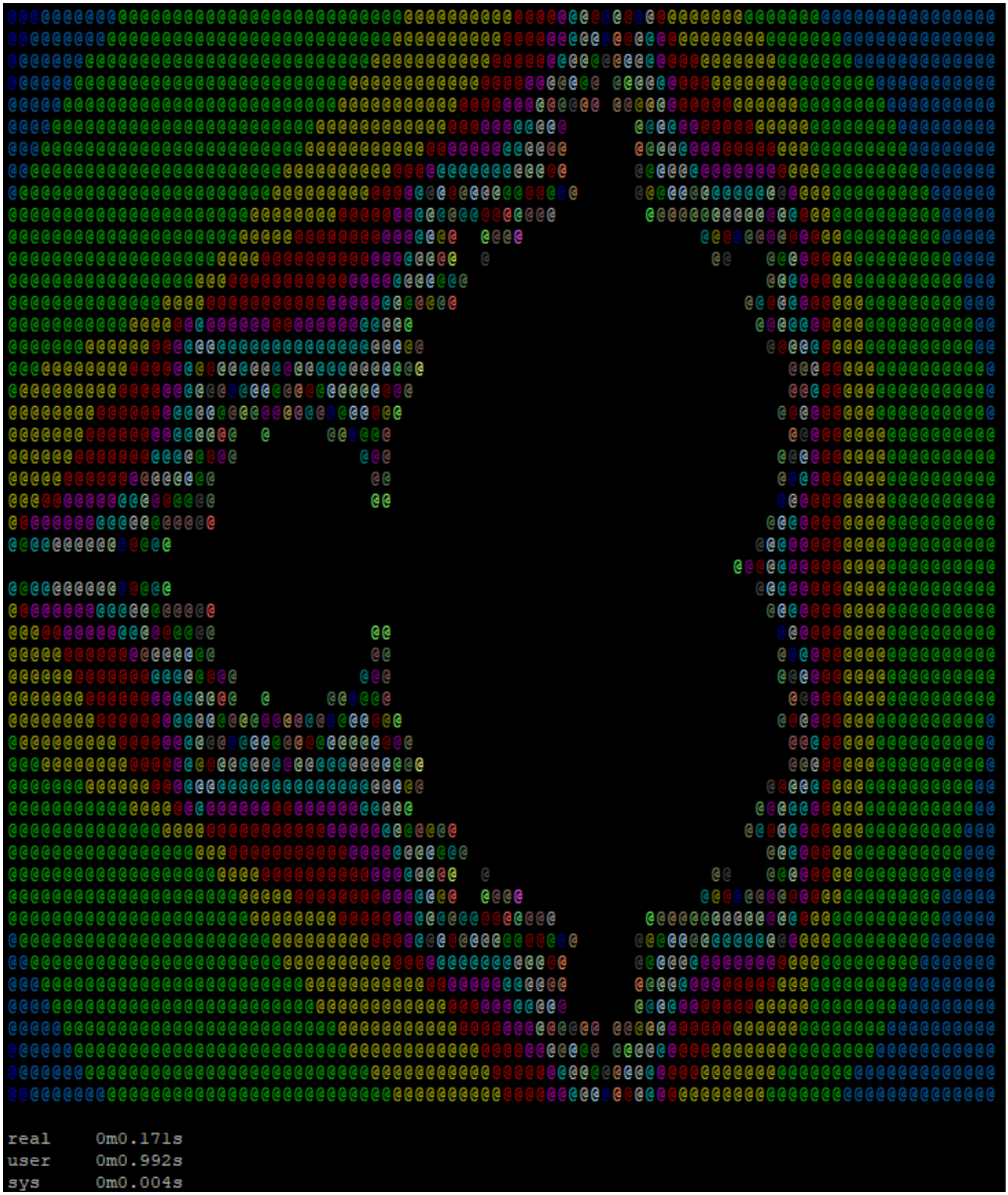
```
        }

        for (i = 0; i < nprocs; i++) {
            proc[i].pid = wait(&status);
        }

    destroy_shared_memory_area(mutex, sizeof(sem_t) * nprocs);
    reset_xterm_color(1);
    return 0;
}
```

Η έξοδος εκτέλεσης του προγράμματος για διάφορες τιμές του N (αριθμός διεργασιών) φαίνεται παρακάτω:

- Για N = 1:



- Για N = 10:



Απαντήσεις στις ερωτήσεις:

1). Περιμένουμε να είναι αποδοτικότερη η υλοποίηση με νήματα αντί για διεργασίες. Αυτό συμβαίνει για τους εξής λόγους.

Αρχικά το κόστος δημιουργίας των νημάτων είναι μικρότερο από αυτό των διεργασιών. Επίσης, τα νήματα μοιράζονται περισσότερους πόρους μνήμης από τις διεργασίες επομένως κατά την δημιουργία τους δεν απαιτείται τόσος χρόνος για την αντιγραφή των μη κοινών τμημάτων μνήμης.

Το γεγονός ότι οι σημαφόροι βρίσκονται σε διαμοιραζόμενη μνήμη μεταξύ των διεργασιών αυξάνει την επίδοση της υλοποίησης, διότι δεν καταναλώνεται χρόνος για την αντιγραφή και ενημέρωση των σημαφόρων για κάθε διεργασία.

Επίσης, γνωρίζουμε ότι τα νήματα έχουν εκ κατασκευής κοινή μνήμη για τους σημαφόρους, ωστόσο για να έχουμε σημαφόρους σε κοινή μνήμη μεταξύ διεργασιών πρέπει να καλέσουμε την κλήση συστήματος **`mmap()`** με flag MAP_SHARED. Επειδή γενικά τα system calls κοστίζουν σε χρόνο, η υλοποίηση με νήματα περιμένουμε να είναι γρηγορότερη.

1.2.2 Υλοποίηση χωρίς semaphores

Ο πηγαίος κώδικας για την φαίνεται παρακάτω:

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

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

/*
 * POSIX thread functions do not return error numbers in errno,
 * but in the actual return value of the function call instead.
 * This macro helps with error reporting in this case.
 */
#define perror_pthread(ret, msg) \
        do { errno = ret; perror(msg); } while (0)

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

/*
* Output at the terminal is is x_chars wide by y_chars long
*/
#define y_chars 50
//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;

/* shared buffer between processes. buffer is array of pointers */
int *buffer[y_chars];

/*--------------Everythig for processes----------------*/

/*
 * A (distinct) instance of this structure
 * is passed to each process
 */
struct process_info_struct {
        pid_t pid; /* process id, as returned by the library */

        int *color_val; /* Pointer to array to manipulate. Each
process manipulates a line and gives each character a color */

        int prid; /* Application-defined process id */
        int nprocs; /* Number of total processes*/
};

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 thread_count array_size\n\n"
                "Exactly two argument required:\n"
                "    thread_count: The number of threads to
create.\n"
                "    array_size: The size of the array to run
with.\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 store_lines_in_buffer(void *arg)
{
    int i, j;
    struct process_info_struct *proc = arg;
    int *color_val = proc->color_val;

    for (i = proc->prid; i < y_chars; i += proc->nprocs) {
        compute_mandel_line(i, color_val);
        for (j = 0; j < x_chars; j++) {
            buffer[i][j] = color_val[j];
        }
    }

}

void *compute_and_output_mandel_line(int fd, int line)
{

   output_mandel_line(fd, buffer[line]);

   return NULL;
}

void *create_shared_memory_area(unsigned int numbytes)
{
        int pages;
        void *addr;

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

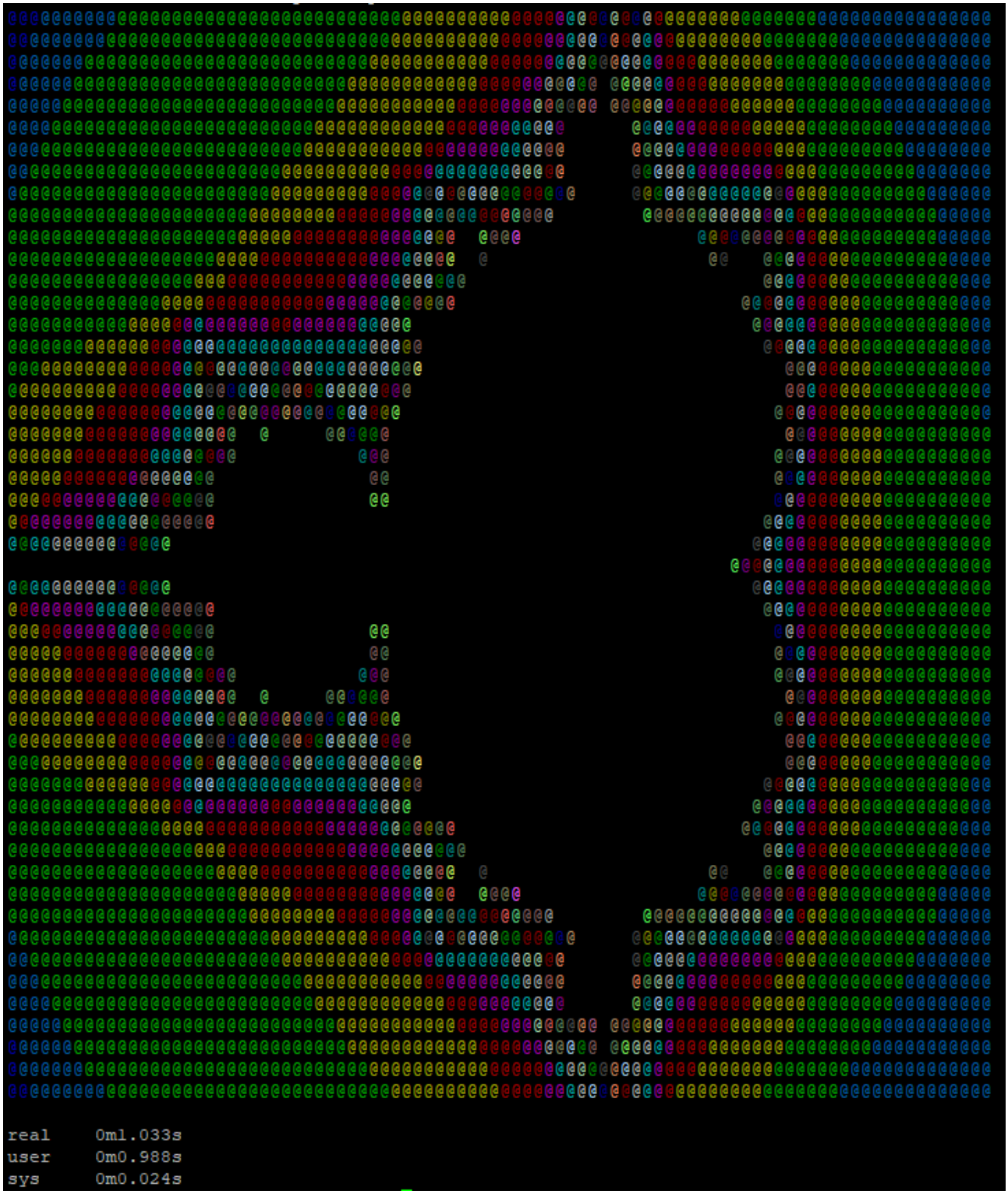
```
                exit(1);
        }

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

        /* Create a shared, anonymous mapping for this number of
pages */
        /* TODO:
                addr = mmap(...)
        */
        addr = mmap(NULL, pages, PROT_READ | PROT_WRITE |
PROT_EXEC, MAP_ANONYMOUS | MAP_SHARED, -1, 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[])
{
    int i, nprocs, status;
    struct process_info_struct *proc;
```

```
    /*
     * Parse the command line. User gives number of processes
nprocs
     */
    if (argc != 2)
            usage(argv[0]);
    if (safe_atoi(argv[1], &nprocs) < 0 || nprocs <= 0) {
            fprintf(stderr, "`%s' is not valid for `thread_count'\
n", argv[1]);
            exit(1);
    }


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

    /* Create array of nprocs processes. Ecah process has a struct
with its info */
    proc = safe_malloc(nprocs * sizeof(*proc));

    /* initialize shared buffer where the mandlebrot will be
stored */
    for (i = 0; i < y_chars; i++) {
        buffer[i] = create_shared_memory_area(sizeof(int) *
x_chars);
    }

        // create nprocs processes
        for (i = 0; i < nprocs; i++) {
            // create new process and initialize fields of process
struct
            proc[i].prid = i;
            proc[i].nprocs = nprocs;
            proc[i].color_val = safe_malloc(x_chars *
sizeof(int));
            proc[i].pid = fork();

            if (proc[i].pid < 0) {
                perror("Fork failed...:(");
                exit(1);
            }
            // child
            if(proc[i].pid == 0) {
                /* each child process stores the lines that
                belong to it to the shared buffer */
                store_lines_in_buffer(&proc[i]);
                exit(1);
            }

        }
        // parent waits for each child process to finish
        for (i = 0; i < nprocs; i++) {
            proc[i].pid = wait(&status);
```

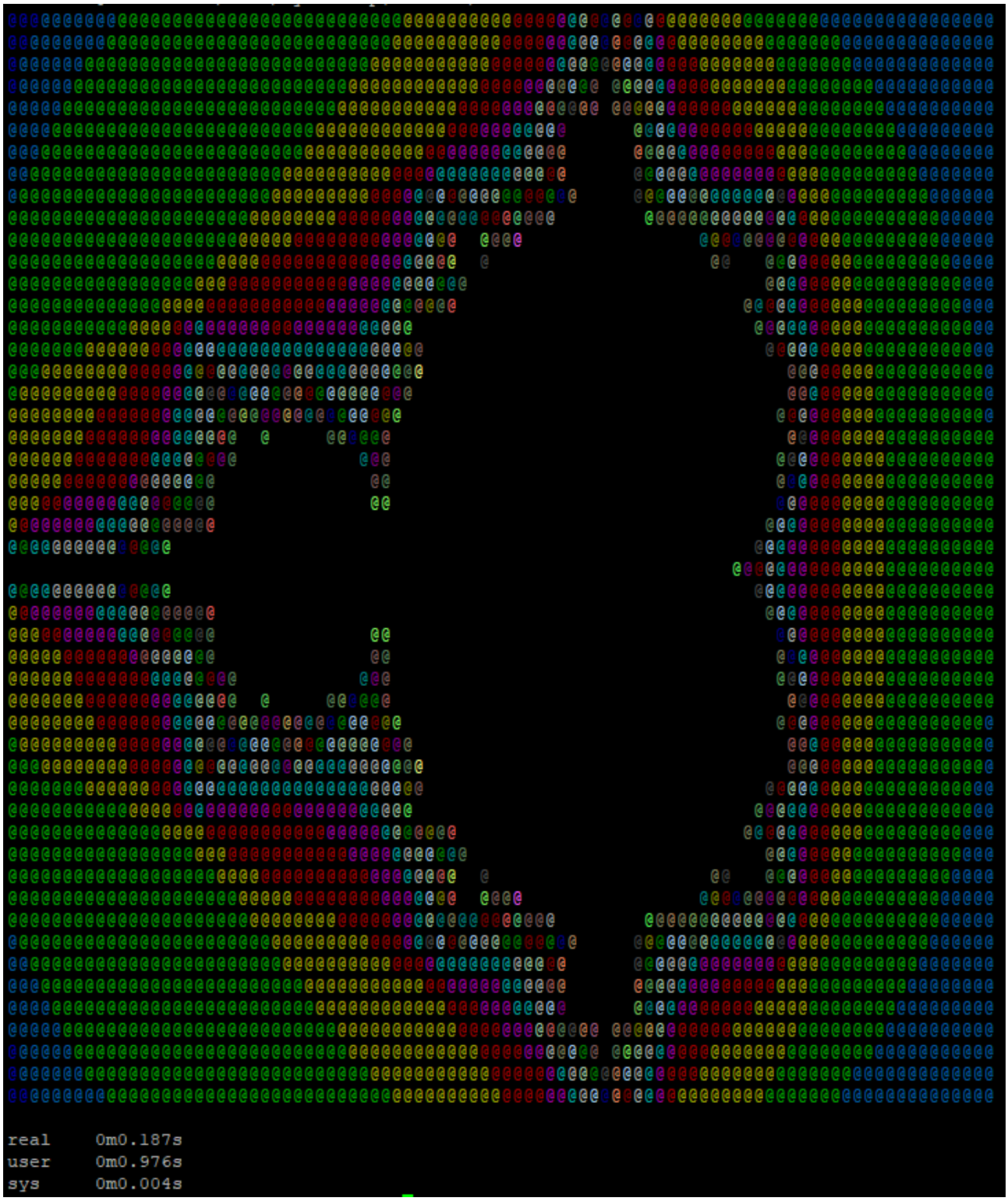
```
        }


    int line;

    /*
     * draw the Mandelbrot Set, one line at a time.
     * Output is sent to file descriptor '1', i.e., standard
output.
     */
    // parent prints the mandlebrot set
    for (line = 0; line < y_chars; line++) {
        compute_and_output_mandel_line(1, line);
    }


    // destroy shared buffer
    for (i = 0; i < y_chars; i++) {
        destroy_shared_memory_area(buffer[i], sizeof(int) *
x_chars);
    }

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

Η έξοδος εκτέλεσης του προγράμματος για διάφορες τιμές του N (αριθμός διεργασιών) φαίνεται παρακάτω:

- Για N = 1:



- Για N = 10:



Απαντήσεις στις ερωτήσεις:

1). Σε αυτήν την υλοποίηση ο συγχρονισμός επιτυγχάνεται με τον εξής τρόπο. Κάθε διεργασία παιδί, μετά την δημιουργία της, υπολογίζει τις γραμμές του Mandelbrot που τις αντιστοιχούν και τις αποθηκεύει σε έναν κοινό χώρο μνήμης μεταξύ όλων των διεργασιών (buffer). Η κύρια διεργασία αναμένει τον τερματισμό όλων των διεργασιών-παιδιά και αναλαμβάνει την εκτύπωση του περιεχομένου της κοινής μνήμης στην οθόνη.

Αν ο buffer είχε διαστάσεις **`NPROCS x x_chars`** τότε κάθε διεργασία-παιδί θα υπολόγιζε μόνο μία γραμμή του Mandelbrot και θα την αποθήκευε στον buffer. Η κύρια διεργασία θα εκτύπωνε το περιεχόμενο του buffer. Οι διεργασίες-παιδιά στη συνέχεια θα υπολόγιζαν τις επόμενες **`NPROCS`** γραμμές, η κύρια διεργασία θα τις εκτύπωνε και αυτή η διαδικασία θα επαναλαμβανόταν μέχρι να εκτυπωθούν όλες οι γραμμές του set.