Λειτουργικά Συστήματα Υπολογιστών

Αναφορά στην 4η Εργαστηριακή Άσκηση

Αλέξανδρος Σκούρας, 03120105

Ιωάννης Τσαντήλας, 03120883

Εξάμηνο: Εαρινό 2022-23

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

```c
char *heap_private_buf;
char *heap_shared_buf;

char *file_shared_buf;

uint64_t buffer_size;


/*
 * Child process' entry point.
 */
void child(void)
{

        /*
         * Step 7 - Child
         */
        if (0 != raise(SIGSTOP))
                die("raise(SIGSTOP)");
        /*
         * TODO: Write your code here to complete child's part of Step 7.
         */

        printf("Virtual Memory Map For Child:\n");
        show_maps();
```

```c
/*
 * Step 8 - Child
 */
if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
/*
 * TODO: Write your code here to complete child's part of Step 8.
 */
printf("Physical Address of heap_private_buf(child):%ld\n", get_physical_address((uint64_t)heap_private_buf));
```

```
/*
 * Step 9 - Child
 */
if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
/*
 * TODO: Write your code here to complete child's part of Step 9.
 */

int i;
for(i=0; i<(int)buffer_size; i++){
  heap_private_buf[i]=0;
}
printf("Physical Address Of heap_private_buf(child):%ld\n", get_physical_address((uint64_t)heap_private_buf));
```

```
/*
 * Step 10 - Child
 */
if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
/*
 * TODO: Write your code here to complete child's part of Step 10.
 */
 for(i=0; i<(int)buffer_size; i++){
     heap_shared_buf[i]=0;
 }
 printf("Physical Address of heap_shared_buf(child):%ld\n", get_physical_address((uint64_t)heap_shared_buf));
```

```
/*
 * 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("Virtual Memory Map For Child:\n");
show_maps();
show_va_info((uint64_t)heap_shared_buf);
```

```
        /*
         * 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);
        munmap(file_shared_buf, buffer_size);
}
```

```c
void parent(pid_t child_pid)
{
        int status;

        /* Wait for the child to raise its first SIGSTOP. */
        if (-1 == waitpid(child_pid, &status, WUNTRACED))
                die("waitpid");

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

        /*
         * TODO: Write your code here to complete parent's part of Step 7.
         */
        printf("Virtual Shared Memory For Parent:\n");
        show_maps();

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

```c
        /*
         * 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.
         */

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

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

```c
        /*
         * 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("Physical Address of heap_private_buf(parent):%ld\n", get_physical_address((uint64_t)heap_private_buf));

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

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

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

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

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

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

    /*
     * TODO: Write your code here to complete parent's part of Step 11.
     */
    printf("Virtual Memory Map For Parent:\n");
    show_maps();
    show_va_info((uint64_t)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_private_buf, buffer_size);
        munmap(heap_shared_buf, buffer_size);
        munmap(file_shared_buf, buffer_size);
}
```

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


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

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

        show_maps();//print virtual memory map


        //End of step 1
```

```c
        /*
         * 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_READ | PROT_WRITE, MAP_ANONYMOUS | MAP_PRIVATE , -1, 0);
        show_maps();
        show_va_info((uint64_t)heap_private_buf);

        //End of step 2
```

```c
        /*
         * 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.
         */

        printf("Physical Address:%ld:", get_physical_address((uint64_t)heap_private_buf));


        //End of step 3
```

```
/*
 * 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.
 */

int i;
for(i=0; i<(int)buffer_size; i++){
   heap_private_buf[i]=0;
 }
 printf("Physical Address:%ld", get_physical_address((uint64_t)heap_private_buf));

//End of step 4
```

```
 /*
  * 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_SHARED, fd, 0);
 char c;
 for(i=0; i<(int)buffer_size; i++){
         c=file_shared_buf[i];
         if(c!=EOF) putchar(c);
         else break;
 }
 show_maps();
 show_va_info((uint64_t)file_shared_buf);

 //End of step 5
```

```c
        //End of step 5

        /*
         * 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_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
        for(i=0; i<(int)buffer_size; i++){
                heap_shared_buf[i]=0;
        }
        show_maps();
        show_va_info((uint64_t)heap_shared_buf);

        //End of step 6

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

        parent(p);

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

Βήμα 1:Τυπώνουμε τον χάρτη εικονικής μνήμης της τρέχουσας διεργασίας με χρήση της βοηθητικής συνάρτησης show_maps().

```
Step 1: Print the virtual address space map of this process [334130].


Virtual Memory Map of process [334130]:
00400000-00403000 r-xp 00000000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
00602000-00603000 rw-p 00002000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
01d1e000-01d3f000 rw-p 00000000 00:00 0                                  [heap]
7fbd64416000-7fbd64438000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64438000-7fbd64591000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64591000-7fbd645e0000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e0000-7fbd645e4000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e4000-7fbd645e6000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e6000-7fbd645ec000 rw-p 00000000 00:00 0
7fbd645f1000-7fbd645f2000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd645f2000-7fbd64612000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd64612000-7fbd6461a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461b000-7fbd6461c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461c000-7fbd6461d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461d000-7fbd6461e000 rw-p 00000000 00:00 0
7ffdab9b8000-7ffdab9d9000 rw-p 00000000 00:00 0                          [stack]
7ffdab9ee000-7ffdab9f2000 r--p 00000000 00:00 0                          [vvar]
7ffdab9f2000-7ffdab9f4000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------
```

Βήμα 2:Δεσμεύουμε τον heap private buffer μεγέθους μιας σελίδας χρησιμοποιώντας την κλήση συστήματος mmap(), ξανατυπώνουμε τον χάρτη εικονικής μνήμης με τη show_maps() και εντοπίζουμε τον χώρο που δεσμεύσαμε μέσω της show_va_info().

```
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 [334130]:
00400000-00403000 r-xp 00000000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
00602000-00603000 rw-p 00002000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
01d1e000-01d3f000 rw-p 00000000 00:00 0                                  [heap]
7fbd64416000-7fbd64438000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64438000-7fbd64591000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64591000-7fbd645e0000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e0000-7fbd645e4000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e4000-7fbd645e6000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e6000-7fbd645ec000 rw-p 00000000 00:00 0
7fbd645f1000-7fbd645f2000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd645f2000-7fbd64612000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd64612000-7fbd6461a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461a000-7fbd6461b000 rw-p 00000000 00:00 0
7fbd6461b000-7fbd6461c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461c000-7fbd6461d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461d000-7fbd6461e000 rw-p 00000000 00:00 0
7ffdab9b8000-7ffdab9d9000 rw-p 00000000 00:00 0                          [stack]
7ffdab9ee000-7ffdab9f2000 r--p 00000000 00:00 0                          [vvar]
7ffdab9f2000-7ffdab9f4000 r-xp 00000000 00:00 0                          [vdso]
-------------------------------------------------------

7fbd6461a000-7fbd6461b000 rw-p 00000000 00:00 0
```

Βήμα 3:Επιχειρούμε να τυπώσουμε τη φυσική διεύθυνση μνήμης στην οποία απεικονίζεται η εικονική διεύθυνση του buffer μέσω της βοηθητικής συνάρτησης get_physical_address. Παρατηρούμε πως δεν έχει γίνει η απεικόνιση της εικονικής μνήμης σε φυσική. Κάτι τέτοιο είναι λογικό καθώς η φυσική μνήμη δεσμεύεται όταν πρόκειται να την προσπελάσουμε διαφορετικά μπορεί να καλύπταμε μέρος της μνήμης το οποίο εν τέλει να μην χρησιμοποιούσαμε ποτέ.

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

VA[0x7fbd6461a000] is not mapped; no physical memory allocated.
Physical Address:0:
```

Βήμα 4:Γεμίζουμε τον buffer με μηδενικά δηλαδή προσπελαύνουμε την μνήμη με αποτέλεσμα πλέον να γίνεται η απεικόνιση της εικονικής σε φυσική.

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

Physical Address:8520507392
```

Βήμα 5:Με χρήση της mmap() δεσμεύουμε τον file shared buffer μεγέθους μίας σελίδας και απεικονίζουμε το αρχείο file.txt στον χώρο διευθύνσεων της διεργασίας μας τυπώνοντας παράλληλα το περιεχόμενό του.

```
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 [334130]:
00400000-00403000 r-xp 00000000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
00602000-00603000 rw-p 00002000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
01d1e000-01d3f000 rw-p 00000000 00:00 0                                  [heap]
7fbd64416000-7fbd64438000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64438000-7fbd64591000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64591000-7fbd645e0000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e0000-7fbd645e4000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e4000-7fbd645e6000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e6000-7fbd645ec000 rw-p 00000000 00:00 0
7fbd645f0000-7fbd645f1000 r--s 00000000 00:26 7748392                    /home/oslab/oslab16/ask4_alex/file.txt
7fbd645f1000-7fbd645f2000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd645f2000-7fbd64612000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd64612000-7fbd6461a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461a000-7fbd6461b000 rw-p 00000000 00:00 0
7fbd6461b000-7fbd6461c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461c000-7fbd6461d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461d000-7fbd6461e000 rw-p 00000000 00:00 0
7ffdab9b8000-7ffdab9d9000 rw-p 00000000 00:00 0                          [stack]
7ffdab9ee000-7ffdab9f2000 r--p 00000000 00:00 0                          [vvar]
7ffdab9f2000-7ffdab9f4000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------

7fbd645f0000-7fbd645f1000 r--s 00000000 00:26 7748392                    /home/oslab/oslab16/ask4_alex/file.txt
```

Βήμα 6:Δεσμεύουμε τον heap shared buffer(μοιραζόμενος μεταξύ των διεργασιών) μεγέθους μιας σελίδας μέσω της mmap(), τυπώνουμε τον χάρτη μνήμης και εντοπίζουμε την νέα απεικόνιση σε αυτόν.

```
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 [334130]:
00400000-00403000 r-xp 00000000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
00602000-00603000 rw-p 00002000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
01d1e000-01d3f000 rw-p 00000000 00:00 0                                  [heap]
7fbd64416000-7fbd64438000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64438000-7fbd64591000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64591000-7fbd645e0000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e0000-7fbd645e4000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e4000-7fbd645e6000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e6000-7fbd645ec000 rw-p 00000000 00:00 0
7fbd645ef000-7fbd645f0000 rw-s 00000000 00:01 8040                       /dev/zero (deleted)
7fbd645f0000-7fbd645f1000 r--s 00000000 00:26 7748392                    /home/oslab/oslab16/ask4_alex/file.txt
7fbd645f1000-7fbd645f2000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd645f2000-7fbd64612000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd64612000-7fbd6461a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461a000-7fbd6461b000 rw-p 00000000 00:00 0
7fbd6461b000-7fbd6461c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461c000-7fbd6461d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461d000-7fbd6461e000 rw-p 00000000 00:00 0
7ffdab9b8000-7ffdab9d9000 rw-p 00000000 00:00 0                          [stack]
7ffdab9ee000-7ffdab9f2000 r--p 00000000 00:00 0                          [vvar]
7ffdab9f2000-7ffdab9f4000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------

7fbd645ef000-7fbd645f0000 rw-s 00000000 00:01 8040                       /dev/zero (deleted)
```

Βήμα 7:Τυπώνουμε τον χάρτη εικονικής μνήμης της διεργασίας παιδί και της διεργασίας πατέρα και παρατηρούμε πως είναι ίδιοι καθώς όπως περιμέναμε μέσω της fork δημιουργείται ένα αντίγραφο της αρχικής διεργασίας που κληρονομεί τα στοιχεία της.

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

Virtual Shared Memory For Parent:

Virtual Memory Map of process [334130]:
00400000-00403000 r-xp 00000000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
00602000-00603000 rw-p 00002000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
01d1e000-01d3f000 rw-p 00000000 00:00 0                                  [heap]
7fbd64416000-7fbd64438000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64438000-7fbd64591000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64591000-7fbd645e0000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e0000-7fbd645e4000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e4000-7fbd645e6000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e6000-7fbd645ec000 rw-p 00000000 00:00 0
7fbd645ef000-7fbd645f0000 rw-s 00000000 00:01 8040                       /dev/zero (deleted)
7fbd645f0000-7fbd645f1000 r--s 00000000 00:26 7748392                    /home/oslab/oslab16/ask4_alex/file.txt
7fbd645f1000-7fbd645f2000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd645f2000-7fbd64612000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd64612000-7fbd6461a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461a000-7fbd6461b000 rw-p 00000000 00:00 0
7fbd6461b000-7fbd6461c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461c000-7fbd6461d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461d000-7fbd6461e000 rw-p 00000000 00:00 0
7ffdab9b8000-7ffdab9d9000 rw-p 00000000 00:00 0                          [stack]
7ffdab9ee000-7ffdab9f2000 r--p 00000000 00:00 0                          [vvar]
7ffdab9f2000-7ffdab9f4000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------
```

```
Virtual Memory Map For Child:

Virtual Memory Map of process [334132]:
00400000-00403000 r-xp 00000000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
00602000-00603000 rw-p 00002000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
01d1e000-01d3f000 rw-p 00000000 00:00 0                                  [heap]
7fbd64416000-7fbd64438000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64438000-7fbd64591000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64591000-7fbd645e0000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e0000-7fbd645e4000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e4000-7fbd645e6000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e6000-7fbd645ec000 rw-p 00000000 00:00 0
7fbd645ef000-7fbd645f0000 rw-s 00000000 00:01 8040                       /dev/zero (deleted)
7fbd645f0000-7fbd645f1000 r--s 00000000 00:26 7748392                    /home/oslab/oslab16/ask4_alex/file.txt
7fbd645f1000-7fbd645f2000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd645f2000-7fbd64612000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd64612000-7fbd6461a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461a000-7fbd6461b000 rw-p 00000000 00:00 0
7fbd6461b000-7fbd6461c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461c000-7fbd6461d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461d000-7fbd6461e000 rw-p 00000000 00:00 0
7ffdab9b8000-7ffdab9d9000 rw-p 00000000 00:00 0                          [stack]
7ffdab9ee000-7ffdab9f2000 r--p 00000000 00:00 0                          [vvar]
7ffdab9f2000-7ffdab9f4000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------
```

Βήμα 8:Τυπώνουμε την φυσική διεύθυνση μνήμης του private buffer για την διεργασία παιδί και για την διεργασία πατέρα και παρατηρούμε ότι έχουν την ίδια τιμή.

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

Physical Address of heap_private_buf(parent):8520507392
Physical Address of heap_private_buf(child):8520507392
```

Βήμα 9:Γράφουμε στον private buffer από την διεργασία παιδί και παρατηρούμε ότι η απεικόνιση για την φυσική μνήμη του παιδιού αλλάζει. Αυτό συμβαίνει λόγω της χρήσης του flag MAP_PRIVATE που δεν επιτρέπει στην διεργασία πατέρα και στην διεργασία παιδί να γράψουν στο ίδιο σημείο της φυσικής μνήμης καθώς έτσι η μια διεργασία θα μπορούσε να αλλάξει τα πράγματα που γράφει η άλλη.

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

Physical Address of heap_private_buf(parent):8520507392
Physical Address Of heap_private_buf(child):5091311616
```

Βήμα 10:Ακολουθούμε την ίδια διαδικασία για τον shared buffer και βλέπουμε τώρα ότι η απεικόνιση στην φυσική μνήμη είναι ίδια και για την διεργασία παιδί και για την διεργασία πατέρα. Αυτό συμβαίνει λόγω του flag MAP_SHARED που μετατρέπει την σελίδα σε διαμοιραζόμενη και επιτρέπει στις διεργασίες να επεξεργάζονται την ίδια μνήμη

```
Step 10: Write to the shared heap buffer (main) from child and get the physical address for both the parent and the child. What happened?

Physical Address of heap_shared_buf:5113327616
Physical Address of heap_shared_buf(child):5113327616
```

Βήμα 11:Απαγορεύουμε τις εγγραφές για την διεργασία παιδί μέσω της mprotect(). Τυπώνουμε τους χάρτες εικονικής μνήμης και για την διεργασία παιδί και για την διεργασία πατέρα και μέσω της show_va_info() διαπιστώνουμε ότι όντως το δικαίωμα εγγραφής έχει αφαιρεθεί από το παιδί.

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

Virtual Memory Map For Parent:

Virtual Memory Map of process [334130]:
00400000-00403000 r-xp 00000000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
00602000-00603000 rw-p 00002000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
01d1e000-01d3f000 rw-p 00000000 00:00 0                                  [heap]
7fbd64416000-7fbd64438000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64438000-7fbd64591000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64591000-7fbd645e0000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e0000-7fbd645e4000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e4000-7fbd645e6000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e6000-7fbd645ec000 rw-p 00000000 00:00 0
7fbd645ef000-7fbd645f0000 rw-s 00000000 00:01 8040                       /dev/zero (deleted)
7fbd645f0000-7fbd645f1000 r--s 00000000 00:26 7748392                    /home/oslab/oslab16/ask4_alex/file.txt
7fbd645f1000-7fbd645f2000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd645f2000-7fbd64612000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd64612000-7fbd6461a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461a000-7fbd6461b000 rw-p 00000000 00:00 0
7fbd6461b000-7fbd6461c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461c000-7fbd6461d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461d000-7fbd6461e000 rw-p 00000000 00:00 0
7fffdab9b8000-7fffdab9d9000 rw-p 00000000 00:00 0                        [stack]
7fffdab9ee000-7fffdab9f2000 r--p 00000000 00:00 0                        [vvar]
7fffdab9f2000-7fffdab9f4000 r-xp 00000000 00:00 0                        [vdso]
-----------------------------------------------------

7fbd645ef000-7fbd645f0000 rw-s 00000000 00:01 8040                       /dev/zero (deleted)
```

```
Virtual Memory Map For Child:

Virtual Memory Map of process [334132]:
00400000-00403000 r-xp 00000000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
00602000-00603000 rw-p 00002000 00:26 7748320                            /home/oslab/oslab16/ask4_alex/mmap
01d1e000-01d3f000 rw-p 00000000 00:00 0                                  [heap]
7fbd64416000-7fbd64438000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64438000-7fbd64591000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd64591000-7fbd645e0000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e0000-7fbd645e4000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e4000-7fbd645e6000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fbd645e6000-7fbd645ec000 rw-p 00000000 00:00 0
7fbd645ef000-7fbd645f0000 r--s 00000000 00:01 8040                       /dev/zero (deleted)
7fbd645f0000-7fbd645f1000 r--s 00000000 00:26 7748392                    /home/oslab/oslab16/ask4_alex/file.txt
7fbd645f1000-7fbd645f2000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd645f2000-7fbd64612000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd64612000-7fbd6461a000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461a000-7fbd6461b000 rw-p 00000000 00:00 0
7fbd6461b000-7fbd6461c000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461c000-7fbd6461d000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fbd6461d000-7fbd6461e000 rw-p 00000000 00:00 0
7fffdab9b8000-7fffdab9d9000 rw-p 00000000 00:00 0                        [stack]
7fffdab9ee000-7fffdab9f2000 r--p 00000000 00:00 0                        [vvar]
7fffdab9f2000-7fffdab9f4000 r-xp 00000000 00:00 0                        [vdso]
-----------------------------------------------------

7fbd645ef000-7fbd645f0000 r--s 00000000 00:01 8040                       /dev/zero (deleted)
```

Βήμα 12:Αποδεσμεύουμε όλους τους buffers και για τις δύο διεργασίες με χρήση της munmap().

1.2 Παράλληλος υπολογισμός Mandelbrot με διεργασίες αντί για νήματα

1.2.1 Semaphores πάνω από διαμοιραζόμενη μνήμη

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

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

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

sem_t *sema;

/*
 *  * 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;

/*
 *  * This function computes a line of output
 *   * as an array of x_char color values.
 *    */

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", __fu
nc__);
                exit(1);
        }

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

        /* Create a shared, anonymous mapping for this number of pages */
        addr = mmap(NULL, pages * sysconf(_SC_PAGE_SIZE),
                        PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
        if (addr == MAP_FAILED) {
                perror("create_shared_memory_area: mmap failed");
                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", __fu
nc__);
                exit(1);
        }

        pages=(numbytes-1)/sysconf(_SC_PAGE_SIZE)+1;

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



void compute_mandel_line(int line, int color_val[])
{
        /*
         *          * x and y traverse the complex plane.
         *                   */
        double x, y;

        int n;
        int val;

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

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

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

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

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

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

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

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

```
void compute_and_output_mandel_line(int current_line, int nprocs)
{
        int line_num;
        int color_val[x_chars];

        for (line_num=current_line; line_num < y_chars; line_num+=nprocs) {
                compute_mandel_line(line_num, color_val);
                if(sem_wait(&sema[current_line])<0) {
                        perror("sem_wait");
                        exit(1);
                }
                output_mandel_line(1, color_val);
                if(sem_post(&sema[(line_num+1) % nprocs])<0) {
                        perror("sem_post");
                        exit(1);
                }
        }
}

int main(int argc, char *argv[])
{
        int i, nprocs, status;

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

        nprocs = atoi(argv[1]);

        if((argc!=2) || (nprocs<=0)){
                fprintf(stderr, "Only one argument needed:number of procs (must be pos
itive)");
                exit(1);
        }

        sema = create_shared_memory_area(nprocs * sizeof(sem_t)); /* allocate shared m
em to store semaphore array */

        for (i=0; i<nprocs; i++) {
                if(sem_init(&sema[i],1,0)<0) {
                        perror("sem_init");
                        exit(1);
                }
        }

        if(sem_post(&sema[0])<0) {
                perror("sem_post");
                exit(1);
        }

        /*create the processes and call the execution function*/
        pid_t proc_pid;
        for(i=0; i<nprocs; i++) {

                proc_pid=fork();

                if(proc_pid< 0) {
                        perror("error with creation of child");
                        exit(1);
                }

                if(proc_pid== 0) {
                        compute_and_output_mandel_line(i, nprocs);
                        exit(1);
                }
        }
```

```
        for(i=0; i<nprocs; i++) {
                proc_pid = wait(&status);
        }

        for(i=0; i<nprocs; i++) {
                sem_destroy(&sema[i]);
        }

        destroy_shared_memory_area(sema, nprocs * sizeof(sem_t));

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

Ερωτήσεις

Ερώτηση 1

Η υλοποίηση με threads είναι πιο γρήγορη από αυτή των processes, αφού μπορούν να εκμεταλλευτούν καλύτερα την διαμοιραζόμενη μνήμη και να αποφύγουν το overhead της επικοινωνίας των διεργασιών. Επιπλέον, σε multicore συστήματα, η υλοποίηση μπορεί να γίνει αποδοτικότερα παράλληλα μέσω threads.

Η χρήση semaphores για την υλοποίηση της άσκησης προσθέτει παραπάνω overhead, ενώ σε multicore συστήματα, όταν πολλές διεργασίες προσπαθούν να έχουν πρόσβαση στον ίδιο, κάποιες θα χρειαστεί να περιμένουν να ελευθερωθεί, αυξάνοντας ακόμη τον συνολικό χρόνο εκτέλεσης. Επιπλέον αυτή η «επικοινωνία» μεταξύ των διεργασίων, ποια δηλαδή θα έχει πρόσβαση μία δεδομένη χρονική στιγμή στον σημαφόρο, αυξάνει τον χρόνο εκτέλεσης. Τέλος τόσο η δημιουργία όσο και η εναλλαγή μεταξύ των διεργασιών είναι πολύ πιο χρονοβόρα σε σχέση με τα threads.

1.2.2 Υλοποίηση χωρίς semaphores

```c
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>
#include <sys/mman.h>
#include <sys/wait.h>

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000
//Set global shared array and number of procs(input by user)
int **shared_array;
int nr_procs;

/*
 *  * 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;

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

        int n;
        int val;

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

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

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

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

```
/*
 *  * This function outputs an array of x_char color values
 *   * to a 256-color xterm.
 *    */
void output_mandel_line(int fd, int color_val[])
{
        int i;
        char point='@';
        char newline='\n';

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

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

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

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

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

        /* Create a shared, anonymous mapping for this number of pages */
        addr=mmap(NULL, pages * sysconf(_SC_PAGE_SIZE), PROT_READ | PROT_WRITE,
                        MAP_SHARED | MAP_ANONYMOUS, -1,0);

        if(addr==MAP_FAILED) {
                perror("mmap");
                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", __fu
nc__);
                exit(1);
        }
```

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

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

void compute_proc_lines(int line)//Compute lines each proc is responsible for
{
        int i;
        for (i=line ; i < y_chars; i += nr_procs)
                compute_mandel_line(i, shared_array[i]);
        return;
}
```

```
int main(int argc, char *argv[])
{
        int status, i;

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

        nr_procs = atoi(argv[1]);//Get number of procs from input and turn string into
 int

        if((argc!=2) || (nr_procs<=0)){//Check if arguments are ok
                fprintf(stderr, "Only one argument needed:number of procs (must be pos
itive)");
                exit(1);
        }
        //Create a 2d array shared between procs
        shared_array = create_shared_memory_area(y_chars * sizeof(int));

        for (i=0; i<y_chars; i++)
                shared_array[i] = create_shared_memory_area(x_chars * sizeof(char));

        //Create child procs(Responsible for computing mandelbrot lines and place them
 in the shared array)
        pid_t pid;
        for(i=0 ; i<nr_procs ; i++) {
                pid=fork();

                if(pid<0) {//Error check
                        perror("Fork");
                        exit(1);
                }

                if(pid==0) {//Child
                        compute_proc_lines(i);
                        exit(1);
                }
        }
        //Father:Responsible to print mandelbrot

        //Wait for all children procs to terminate(All lines computed)
        for(i=0; i<nr_procs ; i++)
                wait(&status);
        //Output whole mandelbrot(Print 2d array)
        for(i=0; i<y_chars ; i++)
                output_mandel_line(1, shared_array[i]);
        //Destroy shared memory
        for(i=0; i<y_chars; i++)
                destroy_shared_memory_area(shared_array[i], sizeof(shared_array[i]));

        destroy_shared_memory_area(shared_array, sizeof(shared_array));

        reset_xterm_color(1);//reset color
        return 0;
}
```

Ερωτήσεις

Ερώτηση 1

Στην παραπάνω υλοποίηση χωρίς semaphores ο συγχρονισμός επιτυγχάνεται μέσω ενός 2-διάστατου buffer τον οποίον ορίζουμε ως global στην αρχή του προγράμματός μας. Σε αυτόν τον buffer αναλαμβάνει να τοποθετήσει η κάθε διεργασία παιδί τις γραμμές του Mandelbrot που της αναλογούν αφού πρώτα τις υπολογίσει. Η διεργασία πατέρας αναμένει τις διεργασίες παιδιά να ολοκληρωθούν , δηλαδή να υπολογισθούν όλες οι γραμμές και να τοποθετηθούν στον buffer, και έπειτα τυπώνει τον πίνακα των δεδομένων παράγοντας το σύνολο Mandelbrot.

Εάν ο πίνακας είχε μέγεθος x_chars x NPROCS αντί για x_chars x y_chars τότε δεν θα μπορούσαμε να τυπώσουμε όλο το σύνολο Mandelbrot με την μια αλλά ο υπολογισμός θα έπρεπε να γίνει σε δόσεις. Ένας τρόπος λύσης θα ήταν να χρησιμοποιούσαμε σήματα δηλαδή οι διεργασίες παιδιά αφού υπολογίσουν την μία δόση του Mandelbrot να λαμβάνουν σήμα να περιμένουν μέχρι να τυπωθεί από τον πατέρα και στην συνέχεια να συνεχίζουν με την επόμενη δόση.