Λειτουργικά Συστήματα Υπολογιστών

6ο εξάμηνο, Ακαδημαϊκή περίοδος 2023-2024

3η Εργαστηριακή Άσκηση

Μηχανισμοί Εικονικής Μνήμης

Δημήτρης Βατάλας: 03121093

Αγγελική Ζέρβα: 03121101

Άσκηση 1.1

Στο συγκεκριμένο μέρος της άσκησης θα πειραματιστούμε με βασικές κλήσεις συστήματος και θα μελετήσουμε βασικούς μηχανικούς του ΛΣ που αφορούν την εικονική μνήμη.

Ερώτημα 1

Για να τυπώσουμε τον χάρτη μνήμης της τρέχουσας διεργασίας χρησιμοποιούμε την εντολή **show_maps()** και το αποτέλεσμα φαίνεται παρακάτω:

```
Step 1: Print the virtual address space map of this process [2318217].


Virtual Memory Map of process [2318217]:
561aa498a000-561aa498b000 r--p 00000000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498b000-561aa498c000 r-xp 00001000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498c000-561aa498d000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498d000-561aa498e000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498e000-561aa498f000 rw-p 00003000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa4a6b000-561aa4a8c000 rw-p 00000000 00:00 0                          [heap]
7fd512e36000-7fd512e58000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512e58000-7fd512fb1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512fb1000-7fd513000000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513000000-7fd513004000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513004000-7fd513006000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513006000-7fd51300c000 rw-p 00000000 00:00 0
7fd513012000-7fd513013000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513013000-7fd513033000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513033000-7fd51303b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303c000-7fd51303d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303d000-7fd51303e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303e000-7fd51303f000 rw-p 00000000 00:00 0
7ffdef3cc000-7ffdef3ed000 rw-p 00000000 00:00 0                          [stack]
7ffdef3f1000-7ffdef3f5000 r--p 00000000 00:00 0                          [vvar]
7ffdef3f5000-7ffdef3f7000 r-xp 00000000 00:00 0                          [vdso]
-------------------------------------------------------
```

Ερώτημα 2

Η δέσμευση buffer μεγέθους μιας σελίδας με τη χρήση της κλήσης συστήματος mmap() γίνεται όπως φαίνεται παρακάτω:

```c
/* Create private buffer*/
heap_private_buf = mmap(NULL, buffer_size, PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS, -1, 0);
/*Check for error*/
if(heap_private_buf == MAP_FAILED) {
    perror("mmap");
    exit(1);
}
show_maps();
show_va_info((uint64_t)heap_private_buf); //Show mapping info only for the buffer
```

Ο νέος χάρτης και οι πληροφορίες αποκλειστικά για τον buffer (τελευταία γραμμή) είναι οι εξής:

```
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 [2318217]:
561aa498a000-561aa498b000 r--p 00000000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498b000-561aa498c000 r-xp 00001000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498c000-561aa498d000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498d000-561aa498e000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498e000-561aa498f000 rw-p 00003000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa4a6b000-561aa4a8c000 rw-p 00000000 00:00 0                          [heap]
7fd512e36000-7fd512e58000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512e58000-7fd512fb1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512fb1000-7fd513000000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513000000-7fd513004000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513004000-7fd513006000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513006000-7fd51300c000 rw-p 00000000 00:00 0
7fd513012000-7fd513013000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513013000-7fd513033000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513033000-7fd51303b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303b000-7fd51303c000 rw-p 00000000 00:00 0
7fd51303c000-7fd51303d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303d000-7fd51303e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303e000-7fd51303f000 rw-p 00000000 00:00 0
7fffdef3cc000-7fffdef3ed000 rw-p 00000000 00:00 0                        [stack]
7fffdef3f1000-7fffdef3f5000 r--p 00000000 00:00 0                        [vvar]
7fffdef3f5000-7fffdef3f7000 r-xp 00000000 00:00 0                        [vdso]
-------------------------------------------------------

7fd51303b000-7fd51303c000 rw-p 00000000 00:00 0
```

Ερώτημα 3

Για να απεικονίσουμε τη φυσική διεύθυνση μνήμης στην οποία απεικονίζεται η εικονική διεύθυνση του buffer χρησιμοποιούμε την παρακάτω εντολή:

```
printf("The physical address is: %ld",
get_physical_address((uint64_t)heap_private_buf));
```

Το αποτέλεσμα της εκτέλεσης της παραπάνω εντολής είναι το εξής:

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

VA[0x7fd51303b000] is not mapped; no physical memory allocated.
The physical address is: 0
```

Παρατηρούμε ότι παρόλο που έχουμε δεσμεύσει μνήμη μεγέθους μιας σελίδας στην εικονική μνήμη, αυτή δεν απεικονίζεται στη φυσική μνήμη. Αυτό οφείλεται στην τεχνική του **demand paging** με βάση την οποία μια σελίδα απεικονίζεται στη φυσική (πραγματική) μνήμη μόνο όταν πάει να προσπελαστεί. Τότε γίνεται page fault και το λειτουργικό σύστημα είναι υπεύθυνο για την πραγματοποίηση της απεικόνισης σε ένα πλάισιο της μνήμης

Ερώτημα 4

Γεμίζουμε τον ίδιο buffer με μηδενικά και τυπώνουμε τη φυσική διεύθυνση της μνήμης στην οποία απεικονίζεται η εικονική διεύθυνση του buffer:

```
for(int i=0; i<(int)buffer_size; i++) heap_private_buf[i] = 0;
printf("The physical address is: %ld",
get_physical_address((uint64_t)heap_private_buf));
```

Το αποτέλεσμα της παραπάνω εντολής σε συνδυασμό με τον μηδενισμό του buffer φαίνεται παρακάτω:

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

The physical address is: 8418598912
```

Σε αυτή την περίπτωση παρατηρούμε ότι η απεικόνιση τη φυσική μνήμη έγινε με επιτυχία και εμφανίζεται η αντίστοιχη διεύθυνση, καθώς ο buffer χρησιμοποιήθηκε πριν την εκτέλεση της εντολής **get_physical_address()**, δηλαδή έγινε μια απαίτηση ώστε το λειτουργικό σύστημα να φέρει τον buffer στη φυσική μνήμη.

Ερώτημα 5

Χρησιμοποιώντας την εντολή **mmap()** έχοντας ως όρισμα τον file descriptor του αρχείου **file.txt** για να απεικονίσουμε την εικονική μνήμη που δεσμεύει το αρχείο στον χάρτη μνήμης και ταυτόχρονα τυπώνουμε το περιεχόμενο.

```
if((fd = open("file.txt", O_RDONLY)) == -1) {
    perror("Problem opening file to read\n");
    exit(1);
}

file_shared_buf = mmap(NULL, buffer_size, PROT_READ, MAP_PRIVATE, fd, 0);

if(file_shared_buf == MAP_FAILED) {
    perror("mmap");
    exit(1);
}

show_maps();
show_va_info((uint64_t)file_shared_buf);

printf("\nThe content of file.txt is: ");
for(int i=0; i<(int)buffer_size; i++) {
    printf("%c", (char)file_shared_buf[i]);
}
```

Παρακάτω παρουσιάζεται ο χάρτης της εικονικής μνήμης συμπεριλαμβάνοντας την εικονική μνήμη που δεσμεύει το αρχείο file.txt μέσω του **file_shared_buf**. Σημειώνεται ότι τυπώνουμε τη συγκεκριμένη διεύθυνση και ξεχωριστά παρακάτω με σκοπό να εντοπιστεί πιο εύκολα.

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


Virtual Memory Map of process [2318217]:
561aa498a000-561aa498b000 r--p 00000000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498b000-561aa498c000 r-xp 00001000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498c000-561aa498d000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498d000-561aa498e000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498e000-561aa498f000 rw-p 00003000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa4a6b000-561aa4a8c000 rw-p 00000000 00:00 0                          [heap]
7fd512e36000-7fd512e58000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512e58000-7fd512fb1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512fb1000-7fd513000000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513000000-7fd513004000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513004000-7fd513006000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513006000-7fd51300c000 rw-p 00000000 00:00 0
7fd513011000-7fd513012000 r--p 00000000 00:27 662575                     /home/oslab/oslab093/ex3/mmap/file.txt
7fd513012000-7fd513013000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513013000-7fd513033000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513033000-7fd51303b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303b000-7fd51303c000 rw-p 00000000 00:00 0
7fd51303c000-7fd51303d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303d000-7fd51303e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303e000-7fd51303f000 rw-p 00000000 00:00 0
7ffdef3cc000-7ffdef3ed000 rw-p 00000000 00:00 0                          [stack]
7ffdef3f1000-7ffdef3f5000 r--p 00000000 00:00 0                          [vvar]
7ffdef3f5000-7ffdef3f7000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------------

7fd513011000-7fd513012000 r--p 00000000 00:27 662575                     /home/oslab/oslab093/ex3/mmap/file.txt

The content of file.txt is: Hello everyone!
```

Ερώτημα 6

Δεσμεύουμε έναν νέο buffer μέσω της εντολής **mmap()** αυτή τη φορά όμως με όρισμα το flag **MAP_SHARED**, αντί του flag **MAP_PRIVATE** που χρησιμοποιούσαμε στα προηγούμενα ερωτήματα.

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

if(heap_shared_buf == MAP_FAILED) {
    perror("mmap");
    exit(1);
}

for(int i=0; i<(int)buffer_size; i++) heap_shared_buf[i] = 0;

show_maps();
show_va_info((uint64_t)heap_shared_buf);
```

Τυπώνουμε το χάρτη της εικονικής μνήμης και παρατηρούμε ότι η σελίδα της μνήμης που δεσμεύαμε χαρακτηρίζεται από τα flag **rw-s**, όπου το "**s**" δηλώνει ότι αυτό το κομμάτι της μνήμης είναι διαμοιραζόμενο.

```
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 [2318217]:
561aa498a000-561aa498b000 r--p 00000000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498b000-561aa498c000 r-xp 00001000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498c000-561aa498d000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498d000-561aa498e000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498e000-561aa498f000 rw-p 00003000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa4a6b000-561aa4a8c000 rw-p 00000000 00:00 0                          [heap]
7fd512e36000-7fd512e58000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512e58000-7fd512fb1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512fb1000-7fd513000000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513000000-7fd513004000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513004000-7fd513006000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513006000-7fd51300c000 rw-p 00000000 00:00 0
7fd513010000-7fd513011000 rw-s 00000000 00:01 61601                      /dev/zero (deleted)
7fd513011000-7fd513012000 r--p 00000000 00:27 662575                     /home/oslab/oslab093/ex3/mmap/file.txt
7fd513012000-7fd513013000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513013000-7fd513033000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513033000-7fd51303b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303b000-7fd51303c000 rw-p 00000000 00:00 0
7fd51303c000-7fd51303d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303d000-7fd51303e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303e000-7fd51303f000 rw-p 00000000 00:00 0
7ffdef3cc000-7ffdef3ed000 rw-p 00000000 00:00 0                          [stack]
7ffdef3f1000-7ffdef3f5000 r--p 00000000 00:00 0                          [vvar]
7ffdef3f5000-7ffdef3f7000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------------

7fd513010000-7fd513011000 rw-s 00000000 00:01 61601                      /dev/zero (deleted)
```

Ερώτημα 7

Στο σημείο αυτό καλείται η **fork()** και δημιουργείται μια νέα διεργασία παιδί. Μέσω της εντολής **show_maps()** τυπώνουμε τον χάρτη της εικονικής μνήμης της διεργασίας παιδί και του πατέρα:

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

Map of parent's virtual memory:

Virtual Memory Map of process [2318217]:
561aa498a000-561aa498b000 r--p 00000000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498b000-561aa498c000 r-xp 00001000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498c000-561aa498d000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498d000-561aa498e000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498e000-561aa498f000 rw-p 00003000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa4a6b000-561aa4a8c000 rw-p 00000000 00:00 0                          [heap]
7fd512e36000-7fd512e58000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512e58000-7fd512fb1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512fb1000-7fd513000000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513000000-7fd513004000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513004000-7fd513006000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513006000-7fd51300c000 rw-p 00000000 00:00 0
7fd513010000-7fd513011000 rw-s 00000000 00:01 61601                      /dev/zero (deleted)
7fd513011000-7fd513012000 r--p 00000000 00:27 662575                     /home/oslab/oslab093/ex3/mmap/file.txt
7fd513012000-7fd513013000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513013000-7fd513033000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513033000-7fd51303b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303b000-7fd51303c000 rw-p 00000000 00:00 0
7fd51303c000-7fd51303d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303d000-7fd51303e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303e000-7fd51303f000 rw-p 00000000 00:00 0
7ffdef3cc000-7ffdef3ed000 rw-p 00000000 00:00 0                          [stack]
7ffdef3f1000-7ffdef3f5000 r--p 00000000 00:00 0                          [vvar]
7ffdef3f5000-7ffdef3f7000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------------
```

```
Map of child's virtual memory:

Virtual Memory Map of process [2318252]:
561aa498a000-561aa498b000 r--p 00000000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498b000-561aa498c000 r-xp 00001000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498c000-561aa498d000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498d000-561aa498e000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498e000-561aa498f000 rw-p 00003000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa4a6b000-561aa4a8c000 rw-p 00000000 00:00 0                          [heap]
7fd512e36000-7fd512e58000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512e58000-7fd512fb1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512fb1000-7fd513000000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513000000-7fd513004000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513004000-7fd513006000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513006000-7fd51300c000 rw-p 00000000 00:00 0
7fd513010000-7fd513011000 rw-s 00000000 00:01 61601                      /dev/zero (deleted)
7fd513011000-7fd513012000 r--p 00000000 00:27 662575                     /home/oslab/oslab093/ex3/mmap/file.txt
7fd513012000-7fd513013000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513013000-7fd513033000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513033000-7fd51303b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303b000-7fd51303c000 rw-p 00000000 00:00 0
7fd51303c000-7fd51303d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303d000-7fd51303e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303e000-7fd51303f000 rw-p 00000000 00:00 0
7ffdef3cc000-7ffdef3ed000 rw-p 00000000 00:00 0                          [stack]
7ffdef3f1000-7ffdef3f5000 r--p 00000000 00:00 0                          [vvar]
7ffdef3f5000-7ffdef3f7000 r-xp 00000000 00:00 0                          [vdso]
-------------------------------------------------------
```

Παρατηρούμε ότι οι δύο χάρτες είναι πανομοιότυποι. Αυτό συμβαίνει καθώς το παιδί μετά την εντολής της **fork()** κληρονομεί την εικονική μνήμη του πατέρα, ενώ παράλληλα το λειτουργικό σύστημα αφαιρεί τα δικαιώματα εγγραφής και από τον πατέρα και από το παιδί στον πίνακα σελίδων.

Ερώτημα 8

Μέσω της εντολής **get_physical_address()** στο παιδί και στον πατέρα τυπώνουμε τη φυσική διεύθυνση της μνήμης που απεικονίζεται ο private buffer

```
printf("Child: physical address of the private buffer: %ld\n",
get_physical_address((uint64_t)heap_private_buf));
```

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

Parent: physical address of the private buffer: 8418598912
Child: physical address of the private buffer: : 8418598912
```

Παρατηρούμε ότι οι διευθύνσεις απεικόνισης στη φυσική μνήμη του private buffer είναι ακριβώς ίδιες, αυτό εξηγείται, όπως αναφέρθηκε και στο προηγούμενο ερώτημα από το γεγονός ότι η διεργασία παιδί κληρονομεί τις διευθύνεις στη μνήμη του πατέρα.

Ερώτημα 9

Σε αυτή την περίπτωση πριν τυπώσουμε τις διευθύνσεις στη φυσική μνήμη του private buffer από το παιδί και από τον γονέα, γεμίζουμε τον buffer με άσους στο παιδί.

```
for(int i=0; i<(int)buffer_size; i++) heap_private_buf[i] = 1;
printf("Child: physical address of the private buffer: : %ld\n",
get_physical_address((uint64_t)heap_private_buf));
```

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

Parent: physical address of the private buffer: 8418598912
Child: physical address of the private buffer: : 7909625856
```

Παρατηρούμε ότι σε αυτή την περίπτωση η διεύθυνση του private buffer των δύο διεργασιών είναι διαφορετική. Αυτό συμβαίνει επειδή το λειτουργικό σύστημα χρησιμοποιεί την τεχνική **Copy-On-Write**, κατά την οποία η διεργασία παιδί μετά την εντολή **fork()** κληρονομεί τον πίνακα σελίδας και όχι όλη τη μνήμη της γονικής διεργασίας. Όμως, αν το παιδί προσπαθήσει να κάνει μια αλλαγή στα δεδομένα της μνήμης τότε το λειτουργικό σύστημα αντιγράφει το συγκεκριμένο πλαίσιο μνήμης σε μια νέα διεύθυνση, με σκοπό να μη γίνεται άσκοπη σπατάλη χώρου κάθε φορά που δημιουργούμε μια νέα διεργασία παιδί.

Ερώτημα 10

Γράφουμε στον shared buffer από τη διεργασία παιδί και τυπώνουμε τη φυσική της διεύθυνση από το παιδί και από τον πατέρα αντίστοιχα:

```
for(int i=0; i<(int)buffer_size; i++) heap_shared_buf[i] = 1;
printf("Child: physical address of the private buffer: : %ld\n",
get_physical_address((uint64_t)heap_shared_buf));
```

```
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?

Parent: physical address of the shared buffer: 5515837440
Child: physical address of the shared buffer: : 5515837440
```

Παρατηρούμε ότι σε αυτήν την περίπτωση η διεύθυνση στη φυσική μνήμη του buffer είναι πάλι η ίδια, παρόλο που το παιδί έγραψε στον buffer. Αυτό συμβαίνει καθώς η σελίδα του συγκεκριμένου buffer είναι διαμοιραζόμενη, καθώς χρησιμοποιήθηκε το flag **MAP_SHARED** κατά τη δημιουργία του στην κλήση της εντολής **mmap()**.

Ερώτημα 11

Μέσω της εντολής **mprotect()** και του flag **PROT_READ** αφαιρούμε τα δικαιώματα εγγραφής του παιδιού στον shared buffer.

```
mprotect(heap_shared_buf, buffer_size, PROT_READ);
```

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

Map of parent's virtual memory:

Virtual Memory Map of process [2318217]:
561aa498a000-561aa498b000 r--p 00000000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498b000-561aa498c000 r-xp 00001000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498c000-561aa498d000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498d000-561aa498e000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498e000-561aa498f000 rw-p 00003000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa4a6b000-561aa4a8c000 rw-p 00000000 00:00 0                          [heap]
7fd512e36000-7fd512e58000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512e58000-7fd512fb1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512fb1000-7fd513000000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513000000-7fd513004000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513004000-7fd513006000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513006000-7fd51300c000 rw-p 00000000 00:00 0
7fd513010000-7fd513011000 rw-s 00000000 00:01 61601                      /dev/zero (deleted)
7fd513011000-7fd513012000 r--p 00000000 00:27 662575                     /home/oslab/oslab093/ex3/mmap/file.txt
7fd513012000-7fd513013000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513013000-7fd513033000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513033000-7fd51303b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303b000-7fd51303c000 rw-p 00000000 00:00 0
7fd51303c000-7fd51303d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303d000-7fd51303e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303e000-7fd51303f000 rw-p 00000000 00:00 0
7ffdef3cc000-7ffdef3ed000 rw-p 00000000 00:00 0                          [stack]
7ffdef3f1000-7ffdef3f5000 r--p 00000000 00:00 0                          [vvar]
7ffdef3f5000-7ffdef3f7000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------------

7fd513010000-7fd513011000 rw-s 00000000 00:01 61601                      /dev/zero (deleted)
```

```
Map of child's virtual memory:

Virtual Memory Map of process [2318252]:
561aa498a000-561aa498b000 r--p 00000000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498b000-561aa498c000 r-xp 00001000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498c000-561aa498d000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498d000-561aa498e000 r--p 00002000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa498e000-561aa498f000 rw-p 00003000 00:27 661325                     /home/oslab/oslab093/ex3/mmap/mmap
561aa4a6b000-561aa4a8c000 rw-p 00000000 00:00 0                          [heap]
7fd512e36000-7fd512e58000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512e58000-7fd512fb1000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd512fb1000-7fd513000000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513000000-7fd513004000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513004000-7fd513006000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fd513006000-7fd51300c000 rw-p 00000000 00:00 0
7fd513010000-7fd513011000 r--s 00000000 00:01 61601                      /dev/zero (deleted)
7fd513011000-7fd513012000 r--p 00000000 00:27 662575                     /home/oslab/oslab093/ex3/mmap/file.txt
7fd513012000-7fd513013000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513013000-7fd513033000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd513033000-7fd51303b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303b000-7fd51303c000 rw-p 00000000 00:00 0
7fd51303c000-7fd51303d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303d000-7fd51303e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fd51303e000-7fd51303f000 rw-p 00000000 00:00 0
7ffdef3cc000-7ffdef3ed000 rw-p 00000000 00:00 0                          [stack]
7ffdef3f1000-7ffdef3f5000 r--p 00000000 00:00 0                          [vvar]
7ffdef3f5000-7ffdef3f7000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------------

7fd513010000-7fd513011000 r--s 00000000 00:01 61601                      /dev/zero (deleted)
```

Παρατηρούμε ότι στον χάρτης της εικονικής μνήμης του πατέρα, ο shared buffer έχει τα flag **rw-s**, άρα μπορεί να γράφει στον συγκεκριμένο buffer, σε αντίθεση με το παιδί που έχει τα flas **r–s**.

Ερώτημα 12

Τέλος, μέσω της εντολής **munmap()** αποδεσμεύουμε όλους τους buffer που χρησιμοποιήθηκαν από τη διεργασία παιδί και πατέρα με σκοπό να αποφύγουμε τη σπατάλη μνήμης, όπως φαίνεται παρακάτω:

```
munmap(heap_private_buf,buffer_size);
munmap(heap_shared_buf,buffer_size);
munmap(file_shared_buf,buffer_size);
```

Άσκηση 1.2

Σκοπός της άσκησης αυτής είναι η τροποποίηση του προγράμματος Mandelbrot set, έτσι ώστε αντί για νήματα να χρησιμοποιούνται διεργασίες για τον παραλληλισμό του υπολογισμού του.

1.2.1 Semaphores πάνω σε διαμοιραζόμενη μνήμη

Ο συγχρονισμός μεταξύ των διεργασιών θέλουμε να επιτυγχάνεται με semaphores. Παρ'όλα αυτά οι διεργασίες δεν έχουν κοινή μνήμη για να μοιράζονται τις πληροφορίες για το κλείδωμα του κρίσιμου τμήματος. Χρειάζεται λοιπόν, η δέσμευση διαμοιραζόμενης μνήμης στην οποία θα πρέπει να οριστούν τα semaphores, έτσι ώστε να έχουν πρόσβαση όλες οι διεργασίες. Για τον λόγο αυτό επεκτείνουμε τη συνάρτηση **create_shared_memory_area** με τη χρήση της κλήσης συστήματος **mmap** ώστε να επιστρέφει έναν δείκτη σε memory mapping, το οποίο θα μπορεί να διαμοιραστεί μεταξύ των διεργασιών. Η συνάρτηση παρουσιάζεται παρακάτω:

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

```
    int pages;
    void *addr;

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

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

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

        if(addr == MAP_FAILED) {
                perror("mmap");
                exit(1);
        }
    return addr;
}
```

Αφού υλοποιήσουμε την παραπάνω συνάρτηση, τη χρησιμοποιούμε για να δημιουργήσουμε έναν πίνακα από semaphores με μέγεθος ίσο με τον αριθμό των διεργασιών, με σκοπό να χρησιμοποιήσουμε το ίδιο σχήμα συγχρονισμού με την Άσκηση 2, όπου είχαμε νήματα. Η δημιουργία και η αρχικοποίηση των semaphores είναι η εξής:

```
/* Create semaphores on shared memory*/
    sem = create_shared_memory_area(sizeof(sem_t)*NPROCS);

/*Initialize first semaphore to 1*/
ret = sem_init(&sem[0], 1, 1);
    if(ret!=0) {
            perror("sem_init");
            exit(1);
    }
    /*Initialize all other semaphores to 0*/
for(int i=1; i<NPROCS; i++) {
            ret = sem_init(&sem[i], 1, 0);
            if(ret<0) {
                    perror("sem_init");
                    exit(1);
            }
    }
```

Στη συνέχεια δημιουργούμε NPROCS threads και αναθέτουμε στην κάθε διεργασία να υπολογίσει τις σειρές του Mandelbrot set με αντίστοιχο τρόπο με την περίπτωση των νημάτων. Η δημιουργία των διεργασιών και η συνάρτηση όπου γίνεται ο υπολογισμός και η εμφάνιση του Mandelbrot set φαίνονται παρακάτω:

```
/*Use fork inside a loop to create NPROCS child processes*/
for(int i=0; i<NPROCS; i++) {
    p[i] = fork();
    if(p[i]<0){
        perror("fork");
        exit(1);
    }
    if(p[i] == 0) {
        /*Each process calculates and outputs the corresponding lines*/
        for(int line = i; line<=y_chars; line+=NPROCS) {
            compute_and_output_mandel_line(1, line, i);
        }
        exit(0);
    }

}
...

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

        compute_mandel_line(line, color_val);
        /*Wait for semaphore that corresponds to specific thread*/
        ret = sem_wait(&sem[(int)procnum]);
        if(ret<0) {
                perror("sem_wait");
                exit(1);
        }

        output_mandel_line(fd, color_val);
        /*Post on semaphore of next thread (circular)*/
        ret = sem_post(&sem[((int)procnum + 1)%NPROCS]);
        if(ret<0) {
                perror("sem_post");
                exit(1);
        }

}
```

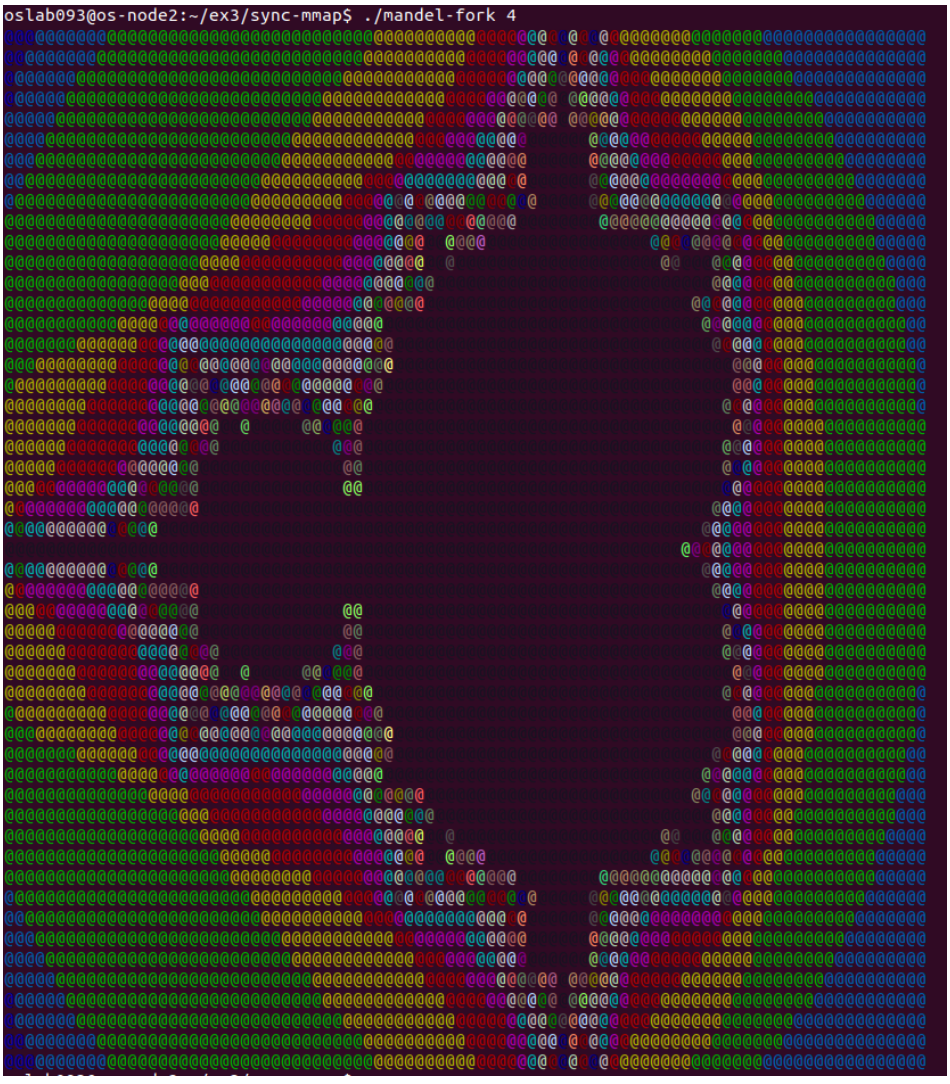
Τέλος, αφού τερματίσουν οι διεργασίες καταστρέφουμε τους σεμαφόρους και αποδεσμεύουμε τη διαμοιραζόμενη μνήμη, που είχαμε δεσμεύσει αρχικά, μέσω της συνάρτησης **destroy_shared_memory_area**

```
/*Wait for all processes to terminate*/
for(int i=0; i<NPROCS; i++) {
    p[i] = wait(&status);
//      explain_wait_status(p[i], status);
```

```
}

/*Destroy semaphores*/
for(int i=0; i<NPROCS; i++) {
    ret = sem_destroy(&sem[i]);
    if(ret!=0) {
        perror("sem_destroy");
    }
}
/*Destroy shared memory area*/
destroy_shared_memory_area(sem, sizeof(sem));
```

Εκτελώντας το πρόγραμμα με 4 διεργασίες επιβεβαιώνουμε την ορθή λειτουργία του, λαμβάνοντας το εξής αποτέλεσμα:



Ερώτημα 1

Περιμένουμε ότι η παραλληλοποίηση με τα threads θα έχει καλύτερη επίδοση σε σχέση με τα processes, καθώς τα νήματα χρειάζονται λιγότερους πόρους από τις διεργασίες, έχουν μικρότερο κόστος δημιουργίας και καταστροφής και μικρότερο κόστος επικοινωνίας. Αυτό συμβαίνει γιατί για τη δημιουργία των νημάτων είναι απαραίτητη η δημιουργία των PCB, η αφαίρεση δικαιωμάτων για το Copy-on-Write, ενώ για το context switch (εναλλαγή διεργασίων) πρέπει να γίνει και αποκατάσταση του PCB. Το γεγονός ότι τα semaphores βρίσκονται σε διαμοιραζόμενη μνήμη βελτιώνει την επίδοση του συστήματος, καθώς δε χρειάζεται να υλοποιήσουμε κάποιον πιο χρονοβόρο και πολύπλοκο τρόπο διεργασιακής επικοινωνίας.

Ερώτημα 2

Το mmap() interface θα μπορούσε ίσως να χρησιμοποιηθεί για τον διαμοιρασμό μνήμης μεταξύ διεργασιών χωρίς κοινό πρόγονο, μέσω της αποθήκευσης του δείκτη στη διαμοιραζόμενη μνήμη σε ένα αρχείο, το οποίο θα μπορούν να απεικονίσουν όλες οι διεργασίες και να έχουν πρόσβαση στη διαμοιραζόμενη μνήνη.

1.2.2 Υλοποίηση χωρίς semaphores

Για τη συγκεκριμένη υλοποίηση δεν θα γίνει χρήση των semaphores για τον συγχρονισμό των διεργασιών, αλλά οι διεργασίες παιδιά θα υπολογίζουν τις γραμμές του set και θα τις αποθηκεύουν στην αντίστοιχη γραμμή ενός διαμοιραζομένου buffer. Η δημιουργία του buffer γίνεται με τη χρήση της **create_shared_memory_area** ως εξής:

```
/*Create shared buffer for threads to write their result*/
buff = create_shared_memory_area(y_chars*sizeof(int));
```

Ο υπολογισμός των γραμμών για κάθε διεργασία και αποθήκευση στον διαμοιραζόμενο buffer φαίνεται παρακάτω:

```
/*Use fork inside a loop to creat NPROCS processes*/
for(int i=0; i<NPROCS; i++) {
    p[i] = fork();
    if(p[i]<0){
        perror("fork");
        exit(1);
    }
    if(p[i] == 0) {
        for(int line = i; line<=y_chars; line+=NPROCS) {
            /*Compute mandel line and store result to shared buff*/
            compute_mandel_line(line, buff[line]);
        }
        exit(0);
    }

}
```

Αφού τερματίσουν οι διεργασίες, η διεργασία γονέας εκτυπώνει τα δεδομένα του buffer, ώστε να εμφανιστεί η ορθή μορφή του Mandelbrot set, και στη συνέχεια γίνεται χρήση της **destroy_shared_memory_area** για την αποδέσμευση του buffer

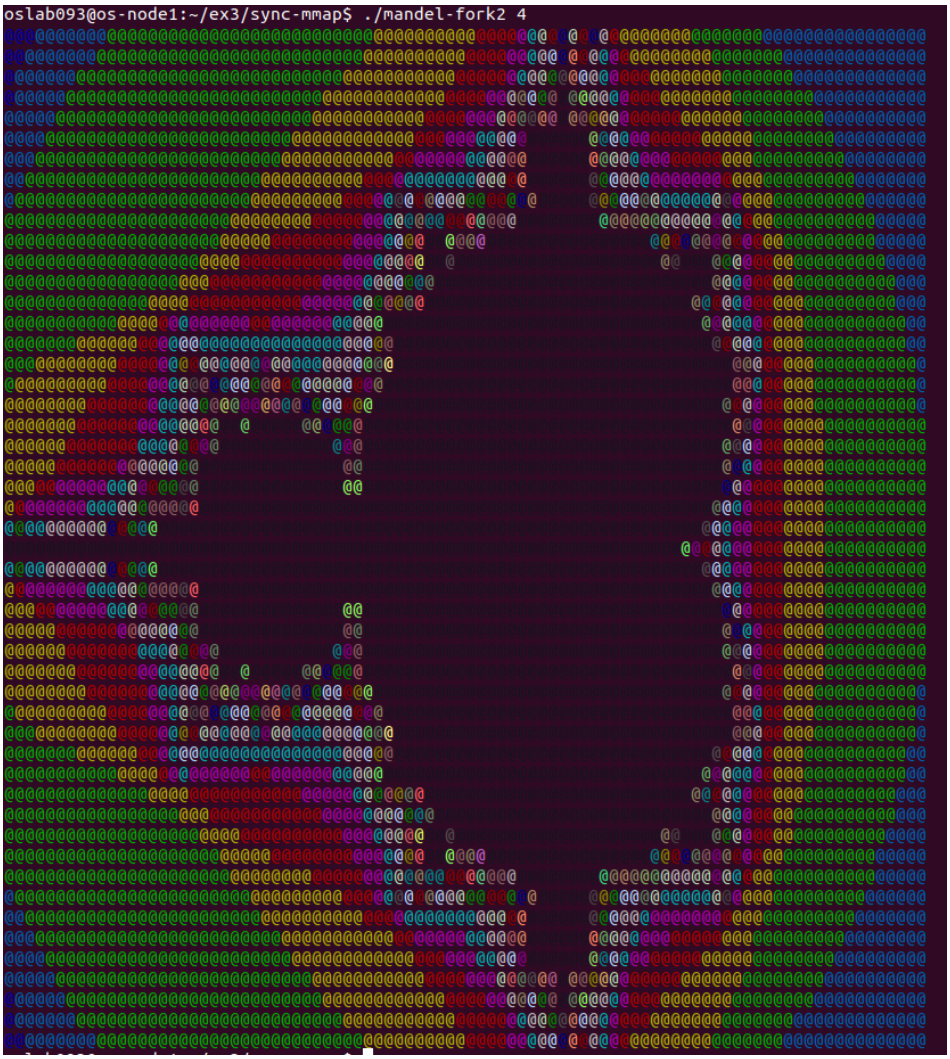
```
/*Output result from shared buff*/
for(int i=0; i<y_chars; i++) {
    output_mandel_line(1, buff[i]);
}

/*Destroy shared memory*/
```

```
for(int i=0; i<y_chars; i++) {
    destroy_shared_memory_area(buff[i], x_chars*sizeof(char));
 }

destroy_shared_memory_area(buff, sizeof(buff));
```

Εκτελώντας το πρόγραμμα με 4 διεργασίες επιβεβαιώνουμε την ορθή λειτουργία του, λαμβάνοντας το εξής αποτέλεσμα:



Ερώτημα 1

Στην περίπτωση αυτή ο συγχρονισμός των διεργασιών επιτυγχάνεται με το γεγονός ότι κάθε διεργασία αποθηκεύει στον buffer τις γραμμές που υπολογίζει στις αντίστοιχες γραμμές του buffer. Αφού λοιπόν, οι διεργασίες έχουν αποθηκεύσει τις γραμμές με τη σωστή σειρά, όπως αναφέραμε παραπάνω, η γονεϊκή διεργασία αναλαμβάνει να εκτυπώσει τα περιεχόμενα του buffer με τη σωστή σειρά, έτσι ώστε να προκύψει το σωτσό Mandelbrot set. Αν ο buffer είχε διαστάσεις NPROCS x x_chars θα έπρεπε να σπάσουμε τη διαδικασία σε τμήματα και να τυπώσουμε σταδιακά το συνολικό σχήμα. Θα έπρεπε δηλαδή, κάθε διεργασία να υπολογίζει μια γραμμή και μόλις όλες τελειώσουν να στέλνεται κάποιο σήμα για την ειδοποίηση της γονεικής διαδικασίας, ώστε να γίνεται η εκτύπωση του πρώτου τμήματος. Η διαδικασία αυτή θα επαναλαμβάνεται μέχρι να ολοκληρωθεί όλο το Mandelbrot set.