OSLab 2023-24 — 3η Εργαστηριακή Άσκηση

Κόρδας Νικόλαος - Α.Μ.: 03121032
Κριθαρίδης Κωνσταντίνος - Α.Μ.: 03121045

19 Μαΐου 2024

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

1.

Αρχικά, τυπώνουμε τον χάρτη εικονικής μνήμης της τρέχουσας διεργασίας με την εντολή show_maps().

```
/*
 * 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();

// My code starts
show_maps();
//My code ends
```

```
Step 1: Print the virtual address space map of this process [1378596].


Virtual Memory Map of process [1378596]:
55a869b9b000-55a869b9c000 r--p 00000000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9c000-55a869b9d000 r-xp 00001000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9d000-55a869b9e000 r--p 00002000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9e000-55a869b9f000 r--p 00002000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9f000-55a869ba0000 rw-p 00003000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a86b32f000-55a86b350000 rw-p 00000000 00:00 0                          [heap]
7f1dbd004000-7f1dbd026000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd026000-7f1dbd17f000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd17f000-7f1dbd1ce000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1ce000-7f1dbd1d2000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1d2000-7f1dbd1d4000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1d4000-7f1dbd1da000 rw-p 00000000 00:00 0
7f1dbd1e0000-7f1dbd1e1000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd1e1000-7f1dbd201000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd201000-7f1dbd209000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd20a000-7f1dbd20b000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd20b000-7f1dbd20c000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd20c000-7f1dbd20d000 rw-p 00000000 00:00 0
7ffed8f15000-7ffed8f36000 rw-p 00000000 00:00 0                          [stack]
7ffed8fe0000-7ffed8fe4000 r--p 00000000 00:00 0                          [vvar]
7ffed8fe4000-7ffed8fe6000 r-xp 00000000 00:00 0                          [vdso]
-------------------------------------------------------
```

Στις γραμμές φαίνονται οι περιοχές εικονικής μνήμης VMAs στον χάρτη μνήμης της διεργασίας. Στις στήλες φαίνονται με τη σειρά για κάθε περιοχή: το εύρος εικονικών διευθύνσεων που καταλαμβάνει, τα δικαιώματα πρόσβασης που έχει η διεργασίας στην κάθε περιοχή μνήμης (r, w, x, p/s) για read, write, execute και private (copy on write)/shared αντίστοιχα, το offset, το device (major:minor), το inode και το pathname που υποστηρίζει την απεικόνιση μνήμης.

2.

Χρησιμοποιώντας την εντολή mmap δημιουργούμε έναν buffer μεγέθους 1 σελίδας και ξανατυπώνουμε τον χάρτη μνήμης της διεργασίας.

```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();

//My code starts
if((heap_private_buf = (char *) mmap(NULL, buffer_size, PROT_READ|PROT_WRITE, MAP_ANONYMOUS|
  MAP_PRIVATE, -1, 0)) == MAP_FAILED){
  die("mmap error");
}
show_maps();
//My code ends
```

```
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 [1378596]:
55a869b9b000-55a869b9c000 r--p 00000000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9c000-55a869b9d000 r-xp 00001000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9d000-55a869b9e000 r--p 00002000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9e000-55a869b9f000 r--p 00002000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9f000-55a869ba0000 rw-p 00003000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a86b32f000-55a86b350000 rw-p 00000000 00:00 0                          [heap]
7f1dbd004000-7f1dbd026000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd026000-7f1dbd17f000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd17f000-7f1dbd1ce000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1ce000-7f1dbd1d2000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1d2000-7f1dbd1d4000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1d4000-7f1dbd1da000 rw-p 00000000 00:00 0
7f1dbd1e0000-7f1dbd1e1000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd1e1000-7f1dbd201000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd201000-7f1dbd209000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd209000-7f1dbd20a000 rw-p 00000000 00:00 0
7f1dbd20a000-7f1dbd20b000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd20b000-7f1dbd20c000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd20c000-7f1dbd20d000 rw-p 00000000 00:00 0
7ffed8f15000-7ffed8f36000 rw-p 00000000 00:00 0                          [stack]
7ffed8fe0000-7ffed8fe4000 r--p 00000000 00:00 0                          [vvar]
7ffed8fe4000-7ffed8fe6000 r-xp 00000000 00:00 0                          [vdso]
-------------------------------------------------------
```

Η νέα περιοχή μνήμης που δημιουργήθηκε είναι αυτή στο εύρος εικονικών διευθύνσεων 7f1dbd209000-7f1dbd20a000 (πράγματι μεγέθους 1 page = 4KB) με δικαιώματα rw-p.

3.

Με την εντολή get_physical_address τυπώνουμε το physical page frame number (pfn) στο οποίο έχει απεικονιστεί το VMA που αντιστοιχεί στην σελίδα του buffer μας.

```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();

//My code starts
pa = get_physical_address((unsigned long) heap_private_buf);
if(pa != 0) printf("The physical address of the buffer is %" PRIu64 ".\n", pa);
//My code ends
```

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

VA[0x7f1dbd209000] is not mapped; no physical memory allocated.
```

Βλέπουμε πως ακόμα δεν έχει απεικονιστεί σε φυσική διεύθυνση ο buffer μας. Αυτό είναι αναμενόμενο, αφού τα linux

χρησιμοποιούν lazy allocation και δεν δεσμεύουν όντως τη μνήμη όταν κάνουμε malloc ή mmap, αλλά μόνο όταν όντως πάμε να τη χρησιμοποιήσουμε.

4.

Αρχικοποιούμε τον buffer σε 0 με την εντολή memset και ξανατυπώνουμε την φυσική του διεύθυνση.

```
/*
 * 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();

//My code starts
memset(heap_private_buf, 0, buffer_size);
pa = get_physical_address((unsigned long) heap_private_buf);
if(pa != 0) printf("The physical address of the buffer is %" PRIu64 ".\n", pa);
//My code ends
```

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

The physical address of the buffer is 9718382592.
```

Παρατηρούμε πως πλέον που χρησιμοποιήσαμε τον buffer μας, όντως δεσμεύτηκε χώρος στη φυσική μνήμη για αυτόν, το physical pfn του οποίου φαίνεται παραπάνω.

5.

Τώρα χρησιμοποιώντας την εντολή mmap απεικονίζουμε το αρχείο file.txt στον χώρο εικονικών διευθύνσεων της διεργασίας μας. Στη συνέχεια, τυπώνουμε τα περιεχόμενά του μέσω του virtual memory mapping και ξαναεμφανίζουμε τον χάρτη μνήμης της διεργασίας μας.

```
/*
 * 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();

//My code starts
if((fd = open("file.txt", O_RDONLY)) == -1){
  die("open error");
}
struct stat st;
if(stat("file.txt", &st) < 0){
  die("stat error");
}
size = st.st_size;
if((file_shared_buf = (char *) mmap(NULL, size, PROT_READ, MAP_SHARED, fd, 0)) == MAP_FAILED){
  die("mmap error");
}
for(size_t i = 0; i < size; i++){
  putchar(file_shared_buf[i]);
}
show_maps();
//My code ends
```

```
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 [1378596]:
55a869b9b000-55a869b9c000 r--p 00000000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9c000-55a869b9d000 r-xp 00001000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9d000-55a869b9e000 r--p 00002000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9e000-55a869b9f000 r--p 00002000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9f000-55a869ba0000 rw-p 00003000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a86b32f000-55a86b350000 rw-p 00000000 00:00 0                          [heap]
7f1dbd004000-7f1dbd026000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd026000-7f1dbd17f000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd17f000-7f1dbd1ce000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1ce000-7f1dbd1d2000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1d2000-7f1dbd1d4000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1d4000-7f1dbd1da000 rw-p 00000000 00:00 0
7f1dbd1df000-7f1dbd1e0000 r--s 00000000 00:27 9580625                    /home/oslab/oslab121/exercise3/mmap/file.txt
7f1dbd1e0000-7f1dbd1e1000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd1e1000-7f1dbd201000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd201000-7f1dbd209000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd209000-7f1dbd20a000 rw-p 00000000 00:00 0
7f1dbd20a000-7f1dbd20b000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd20b000-7f1dbd20c000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd20c000-7f1dbd20d000 rw-p 00000000 00:00 0
7ffed8f15000-7ffed8f36000 rw-p 00000000 00:00 0                          [stack]
7ffed8fe0000-7ffed8fe4000 r--p 00000000 00:00 0                          [vvar]
7ffed8fe4000-7ffed8fe6000 r-xp 00000000 00:00 0                          [vdso]
-------------------------------------------------------
```

Το αρχείο, όπως είναι φανερό, έχει απεικονιστεί στον χώρο εικονικών διευθύνσεων που έχει ως pathname το path του αρχείου. Επειδή απεικονίσαμε το αρχείο με την προστασία PROT_READ και το flag MAP_SHARED, βλέπουμε πως τα δικαιώματα πρόσβασης της διεργασίας μας στο virtual memory mapping του αρχείου είναι r–s. Επιπλέον, παρατηρούμε πως το μέγεθος του VMA που λήφθηκε είναι ίσο με 1 σελίδα (4KB), το οποιό είναι αναμενόμενο, αφού το λειτουργικό σύστημα (ΛΣ) μας δίνει VMAs με κβαντισμένο πλήθος σελίδων, και το αρχείο είναι πολύ μικρό οπότε δεν χρειάζεται πάνω από 4KB.

6.

Χρησιμοποιούμε πάλι την εντολή mmap, αυτή τη φορά για τη δημιουργία ενός διαμοιρασμένου buffer μεγέθους 1 σελίδας. Η απεικόνιση γίνεται με την προστασίας PROT_READ|PROT_WRITE και τα flags MAP_ANONYMOUS|MAP_SHARED, αφού θέλουμε να έχουμε rw πρόσβαση στον buffer, ο οποίος να είναι διαμοιρασμένος και δεν αντιστοιχεί σε κάποιο file descriptor. Αρχικοποιούμε τον buffer σε 0 με την memset και τυπώνουμε εκ νέου τον χάρτη μνήμης της διεργασίας.

```
1  /*
2   * Step 6: Use mmap(2) to allocate a shared buffer of 1 page. Use
3   * heap_shared_buf.
4   */
5  printf(RED "\nStep 6: Use mmap(2) to allocate a shared buffer of size "
6    "equal to 1 page. Initialize the buffer and print the new "
7    "mapping information that has been created.\n" RESET);
8  press_enter();
9
10 //My code starts
11 if((heap_shared_buf = (char *) mmap(NULL, buffer_size, PROT_READ|PROT_WRITE, MAP_ANONYMOUS|MAP_SHARED
     , -1, 0)) == MAP_FAILED){
12   die("map error");
13 }
14 memset(heap_shared_buf, 0, buffer_size);
15 show_maps();
16 //My code ends
```

```
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 [1378596]:
55a869b9b000-55a869b9c000 r--p 00000000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9c000-55a869b9d000 r-xp 00001000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9d000-55a869b9e000 r--p 00002000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9e000-55a869b9f000 r--p 00002000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9f000-55a869ba0000 rw-p 00003000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a86b32f000-55a86b350000 rw-p 00000000 00:00 0                          [heap]
7f1dbd004000-7f1dbd026000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd026000-7f1dbd17f000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd17f000-7f1dbd1ce000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1ce000-7f1dbd1d2000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1d2000-7f1dbd1d4000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1d4000-7f1dbd1da000 rw-p 00000000 00:00 0
7f1dbd1de000-7f1dbd1df000 rw-s 00000000 00:01 11322                      /dev/zero (deleted)
7f1dbd1df000-7f1dbd1e0000 r--s 00000000 00:27 9580625                    /home/oslab/oslab121/exercise3/mmap/file.txt
7f1dbd1e0000-7f1dbd1e1000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd1e1000-7f1dbd201000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd201000-7f1dbd209000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd209000-7f1dbd20a000 rw-p 00000000 00:00 0
7f1dbd20a000-7f1dbd20b000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd20b000-7f1dbd20c000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd20c000-7f1dbd20d000 rw-p 00000000 00:00 0
7ffed8f15000-7ffed8f36000 rw-p 00000000 00:00 0                          [stack]
7ffed8fe0000-7ffed8fe4000 r--p 00000000 00:00 0                          [vvar]
7ffed8fe4000-7ffed8fe6000 r-xp 00000000 00:00 0                          [vdso]
-------------------------------------------------------
```

Το VMA που δημιουργήθηκε είναι αυτό με το pathname /dev/zero (deleted) με τα δικαιώματα πρόσβασης rw-s και μέγεθος 1 σελίδα (4KB), όπως ήταν επιθυμητό.

7.

Σε αυτό το σημείο, η main() κάνει fork, δημιουργώντας μια νέα διεργασία, και οδηγεί το παιδί στην διαδικασία child και τον γονέα στην διαδικασία parent.

Τυπώνουμε τους χάρτες εικονικής μνήμης της κάθε διεργασίας καλώντας τη συνάρτηση show_maps() τόσο από τον parent όσο και από το child.

```c
/*
 * 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();

//My code starts
show_maps();
//My code ends
```

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

//My code starts
show_maps();
//My code ends
```

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


Virtual Memory Map of process [1378596]:
55a869b9b000-55a869b9c000 r--p 00000000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9c000-55a869b9d000 r-xp 00001000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9d000-55a869b9e000 r--p 00002000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9e000-55a869b9f000 r--p 00002000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9f000-55a869ba0000 rw-p 00003000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a86b32f000-55a86b350000 rw-p 00000000 00:00 0                          [heap]
7f1dbd004000-7f1dbd026000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd026000-7f1dbd17f000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd17f000-7f1dbd1ce000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1ce000-7f1dbd1d2000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1d2000-7f1dbd1d4000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1d4000-7f1dbd1da000 rw-p 00000000 00:00 0
7f1dbd1de000-7f1dbd1df000 rw-s 00000000 00:01 11322                      /dev/zero (deleted)
7f1dbd1df000-7f1dbd1e0000 r--s 00000000 00:27 9580625                    /home/oslab/oslab121/exercise3/mmap/file.txt
7f1dbd1e0000-7f1dbd1e1000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd1e1000-7f1dbd201000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd201000-7f1dbd209000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd209000-7f1dbd20a000 rw-p 00000000 00:00 0
7f1dbd20a000-7f1dbd20b000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd20b000-7f1dbd20c000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd20c000-7f1dbd20d000 rw-p 00000000 00:00 0
7ffed8f15000-7ffed8f36000 rw-p 00000000 00:00 0                          [stack]
7ffed8fe0000-7ffed8fe4000 r--p 00000000 00:00 0                          [vvar]
7ffed8fe4000-7ffed8fe6000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------


Virtual Memory Map of process [1378810]:
55a869b9b000-55a869b9c000 r--p 00000000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9c000-55a869b9d000 r-xp 00001000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9d000-55a869b9e000 r--p 00002000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9e000-55a869b9f000 r--p 00002000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a869b9f000-55a869ba0000 rw-p 00003000 00:27 9580682                    /home/oslab/oslab121/exercise3/mmap/mmap
55a86b32f000-55a86b350000 rw-p 00000000 00:00 0                          [heap]
7f1dbd004000-7f1dbd026000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd026000-7f1dbd17f000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd17f000-7f1dbd1ce000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1ce000-7f1dbd1d2000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1d2000-7f1dbd1d4000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f1dbd1d4000-7f1dbd1da000 rw-p 00000000 00:00 0
7f1dbd1de000-7f1dbd1df000 rw-s 00000000 00:01 11322                      /dev/zero (deleted)
7f1dbd1df000-7f1dbd1e0000 r--s 00000000 00:27 9580625                    /home/oslab/oslab121/exercise3/mmap/file.txt
7f1dbd1e0000-7f1dbd1e1000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd1e1000-7f1dbd201000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd201000-7f1dbd209000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd209000-7f1dbd20a000 rw-p 00000000 00:00 0
7f1dbd20a000-7f1dbd20b000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd20b000-7f1dbd20c000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f1dbd20c000-7f1dbd20d000 rw-p 00000000 00:00 0
7ffed8f15000-7ffed8f36000 rw-p 00000000 00:00 0                          [stack]
7ffed8fe0000-7ffed8fe4000 r--p 00000000 00:00 0                          [vvar]
7ffed8fe4000-7ffed8fe6000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------
```

Παρατηρούμε πως ο χάρτης εικονικής μνήμης του παιδιού είναι ακριβές αντίγραφο του αντίστοιχου του γονέα. Αυτό είναι αναμενόμενο, αφού, ακόμα και για τα VMAs που δεν είναι διαμοιρασμένα, λόγω της τεχνικής Copy-on-Write (CoW), μόνο όταν το παιδί πάει να κάνει write πάνω σε αυτά θα γίνει η αντιγραφή σε νέα VMAs (και physical page frames) για το παιδί. Οπότε κατά το fork(), το ΛΣ αρκεί να αντιγράψει απλά τον χάρτη μνημης του γονέα στο παιδί, χωρίς να κάνει καμία πραγματική αντιγραφή εικονικής ή φυσικής μνήμης.

8.

Παρακάτω τυπώνουμε από τον γονέα και το παιδί τη φυσική διεύθυνση του heap_private_buf (που απεικονίστηκε στην εικονική μνήμη με MAP_PRIVATE οπότε δεν είναι διαμοιρασμένο στη διεργασία παιδί).

```
1  /*
2   * Step 8: Get the physical memory address for heap_private_buf.
3   * Step 8 - Parent
4   */
5  printf(RED "\nStep 8: Find the physical address of the private heap "
6    "buffer (main) for both the parent and the child.\n" RESET);
```

```
  press_enter();

  //My code starts
  pa = get_physical_address((unsigned long)heap_private_buf);
  if(pa != 0) printf("parent: The physical address of the private buffer is %" PRIu64 ".\n", pa);
  //My code ends
```

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

  //My code starts
  pa = get_physical_address((unsigned long)heap_private_buf);
  if(pa != 0) printf("child: The physical address of the private buffer is %" PRIu64 ".\n", pa);
  //My code ends
```

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

parent: The physical address of the private buffer is 9718382592.
child: The physical address of the private buffer is 9718382592.
```

Όπως βλέπουμε, η φυσική διεύθυνση του private buffer είναι η ίδια στον γονέα και στο παιδί. Αυτό δεν θα πρέπει να μας παραξενέψει, αφού, παρότι γνωρίζουμε ότι δεν είναι διαμοιρασμένος, οπότε δεν θα τον επεξεργάζονται και οι δύο διεργασίες από την ίδια θέση μνήμης, λόγω του CoW, αφού δεν έχουμε γράψει πάνω σε αυτόν από καμία από τις δύο διεργασίες από τη στιγμή που έγινε το fork(), δεν έχει χρειαστεί να δημιουργηθεί αντίγραφο του buffer σε νέο VMA (ή physical pfn όμοια).

9.

Στη συνέχεια, γράφουμε στον heap_private_buf από το παιδί και ξανατυπώνουμε τη φυσική διεύθυνσή του και από τις δύο διεργασίες.

```
  /*
   * 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();

  //My code starts
  pa = get_physical_address((unsigned long)heap_private_buf);
  if(pa != 0) printf("parent: The physical address of the private buffer is %" PRIu64 ".\n", pa);
  //My code ends
```

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

  //My code starts
  heap_private_buf[0] = 0;
  pa = get_physical_address((unsigned long)heap_private_buf);
  if(pa != 0) printf("child: The physical address of the private buffer is %" PRIu64 ".\n", pa);
  //My code ends
```

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

parent: The physical address of the private buffer is 9718382592.
child: The physical address of the private buffer is 6082355200.
```

Όπως περιμέναμε, μετά την εγγραφή από το παιδί, λόγω του CoW που αντέγραψε τα περιεχόμενα του προηγούμενου VMA σε νέο VMA για τον buffer για το παιδί, η φυσική διεύθυνση του private buffer για το παιδί άλλαξε, και είναι πλέον διαφορετική από αυτή για τον γονέα.

10.

Τώρα κάνουμε το ίδιο (εγγραφή και εκτύπωση φυσικής διεύθυνσης) για τον heap_shared_buf (shared buffer).

```
/*
 * 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();

//My code starts
pa = get_physical_address((unsigned long)heap_shared_buf);
if(pa != 0) printf("parent: The physical address of the shared buffer is %" PRIu64 ".\n", pa);
//My code ends
```

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

//My code starts
heap_shared_buf[0] = 0;
pa = get_physical_address((unsigned long)heap_shared_buf);
if(pa != 0) printf("child: The physical address of the shared buffer is %" PRIu64 ".\n", pa);
//My code ends
```

```
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: The physical address of the shared buffer is 5028982784.
child: The physical address of the shared buffer is 5028982784.
```

Παρατηρούμε ότι εδώ, παρά την εγγραφή, η φυσική διεύθυνση του shared buffer στον γονέα και στο παιδί παρέμεινε η ίδια. Αυτό είναι αναμενόμενο, αφού ο buffer είναι διαμοιρασμένος και στις δύο διεργασίες, οπότε και οι δύο διεργασίες επεξεργάζονται τον ίδιο buffer, σε κοινή θέση μνήμης (τόσο εικονικής όσο και φυσικής).

11.

Απαγορεύουμε το δικαίωμα εγγραφής στον shared buffer από τη διεργασία παιδί και τυπώνουμε τις πληροφορίες της εικονικής διεύθυνσης του shared buffer από τον γονέα και το παιδί.

```
/*
 * 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();

//My code starts
printf("parent: ");
show_va_info((uint64_t)heap_shared_buf);
//My code ends
```

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

//My code starts
if(mprotect(heap_shared_buf, buffer_size, PROT_READ) == -1){
  die("mprotect error");
}
printf("child: ");
show_va_info((uint64_t)heap_shared_buf);
//My code ends
```

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

parent: 7f1dbd1de000-7f1dbd1df000 rw-s 00000000 00:01 11322                  /dev/zero (deleted)
child: 7f1dbd1de000-7f1dbd1df000 r--s 00000000 00:01 11322                  /dev/zero (deleted)
```

Βλέπουμε πως, πράγματι, ενώ ο γονέας έχει ακόμα δικαίωμα write στον shared buffer, το παιδί δεν έχει πλέον.

12.

Τέλος, ελευθερώνουμε παρακάτω όλες τις απεικονίσεις μνήμης που φτιάξαμε τόσο από τον γονέα όσο και το παιδί.

```
  /*
   * Step 12: Free all buffers for parent and child.
   * Step 12 - Parent
   */

  //My code starts
  if(munmap(heap_private_buf, buffer_size) == -1) die("munmap");
  if(munmap(heap_shared_buf, buffer_size) == -1) die("munmap");
  if(munmap(file_shared_buf, size) == -1) die("munmap");
  //My code ends
```

```
  /*
   * Step 12 - Child
   */

  //My code starts
  if(munmap(heap_private_buf, buffer_size) == -1) die("munmap");
  if(munmap(heap_shared_buf, buffer_size) == -1) die("munmap");
  if(munmap(file_shared_buf, size) == -1) die("munmap");
  //My code ends
```

2 Παράλληλος υπολογισμός Mandelbrot με διεργασίες αντί για νήματα

Καλούμαστε να τροποποιήσουμε την υλοποίησή μας για τον παράλληλο υπολογισμό του συνόλου Mandelbrot από την 2η Εργαστηριακή Άσκηση ώστε να λειτουργεί κατανείμοντας την εργασία σε πολλές διεργασίες αντί πολλά νήματα.

2.1 Semaphores πάνω από διαμοιραζόμενη μνήμη

Παραθέτουμε παρακάτω τα κύρια αλλαγμένα σημεία του κώδικα.

Με την safe_shared_malloc και την safe_free κάνουμε map σε διαμοιρασμένη μνήμη και unmap για να ελευθερώσουμε το VMA της απεικόνισης.

```
int NPROCS;

sem_t *semaphores;

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

  if ((p = mmap(NULL, size, PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS, -1, 0)) == MAP_FAILED) {
    die("mmap error");
  }

  return p;
}

void safe_free(void *ptr, size_t size){
  if(munmap(ptr, size) == -1) die("munmap error");
}
```

Στη συνέχεια, χρησιμοποιούμε τα semaphores με τον ίδιο ακριβώς τρόπο. Οι διαφορές είναι ότι λόγω του ότι εμείς καλούμε όπως θέλουμε τη συνάρτηση, δεν περιοριζόμαστε από την pthread_create στο να περάσουμε μόνο ένα argument ως void *, αλλά περνάμε κανονικά τα fd και line. Επίσης, όταν τελειώσει η κάθε διεργασία τη δουλειά της, ελευθερώνουμε την απεικόνιση των σημαφόρων από τον χάρτη μνήμης της.

```
void compute_and_output_mandel_line(int fd, int line)
{
 /*
   * A temporary array, used to hold color values for the line being drawn
```

```
   */
  int color_val[x_chars];

  for( ; line < y_chars; line += NPROCS) {
    compute_mandel_line(line, color_val);
    sem_wait(&semaphores[(line)%NPROCS]); //lock current semaphore
    output_mandel_line(fd, color_val);
    sem_post(&semaphores[(line + 1)%NPROCS]); //unlock the next semaphore
  }
  safe_free(semaphores, NPROCS*sizeof(sem_t));
}
```

Τέλος, στη main, δημιουργούμε τις απεικονίσεις στην εικονική μνήμη, αρχικοποιούμε τους σημαφόρους στις κατάλληλες τιμές (ο μηδενικός στην τιμή 1 και οι υπόλοιποι στην τιμή 0) και αυτή τη φορά θέτουμε την παράμετρο pshared = 1 για να δηλώσουμε ότι είναι σε διαμοιρασμένη μνήμη, οπότε θέλουμε να μοιραστούν σε όλες τις διεργασίες παιδιά. Ύστερα, κάνουμε τα fork και καλούμε την συνάρτηση compute_and_output_mandel_line με κατάλληλα ορίσματα από το κάθε παιδί το οποίο αφού την εκτελέσει, τερματίζει. Ο γονέας κρατάει τα pid's των παιδιών και τα περιμένει να τερματίσουν, προτού καταστρέψει τους σημαφόρους, ελευθερώσει την απεικόνισή τους από τον χάρτη μνήμης του και επαναφέρει το αρχικό χρώμα στο τερματικό.

```
int main(int argc, char **argv){
  argument_handling(argc, argv);
  if(signal(SIGINT, sigintHandler) < 0){
    perror("Could not establish SIGINT handler");
    exit(1);
  }

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

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

    procs = (pid_t *) safe_malloc(NPROCS * sizeof(pid_t));
    semaphores = (sem_t*) safe_shared_malloc(NPROCS * sizeof(sem_t));

  if(sem_init(&semaphores[0], 1, 1)) //initialize the 0th semaphore to 1
    die("semaphore init error");

  for(int i = 1; i < NPROCS; ++i) {
        if(sem_init(&semaphores[i], 1, 0)) //and all else to 0
      die("semaphore init error");
  }

    for(int i = 0; i < NPROCS; ++i) {
        pid_t p = fork();
    if(p < 0) die("fork error");
    if(p == 0){
      compute_and_output_mandel_line(1, i);
      return 0;
    }
    if(p > 0) procs[i] = p;
    }

    for(int i = 0; i < NPROCS; ++i) { //wait for all children
        int status;
    if(waitpid(procs[i], &status, WUNTRACED) == -1) die("waitpid error");
    }

  for (int i = 0; i < NPROCS; ++i) { //destroy every semaphore
    if(sem_destroy(&semaphores[i])) die("semaphore destroy error");
  }

  reset_xterm_color(1);
  safe_free(procs, NPROCS*sizeof(pid_t));
  safe_free(semaphores, NPROCS*sizeof(sem_t));
  return 0;
}
```

1.

Αναμένουμε πως η υλοποίηση με threads θα εκτελείται γρηγορότερα από αυτή με processes και διαμοιρασμένη μνήμη, καθώς τα threads δημιουργούνται γρηγορότερα αφού δεν χρειάζεται να αντιγραφεί ο χάρτης μνήμης του γονέα. Το ότι τα semaphores

βρίσκονται σε διαμοιραζόμενη μνήμη αρχικά κάνει την υλοποίηση με διεργασίες δυνατή, αφού αλλιώς δεν θα μπορούσε η μία διεργασία να επηρεάζει τους σημαφόρους της άλλης. Σχετικά με την επίδοση, γενικά ό,τι βρίσκεται σε διαμοιραζόμενη μνήμη σημαίνει ότι δεν θα χρειαστεί να αντιγραφεί η σελίδα του σύμφωνα με τον μηχανισμό CoW, οπότε εξοικονομείται χρόνος.

2.

Θεωρούμε πως δεν μπορεί να χρησιμοποιηθεί το interface mmap για να δημιουργήσει shared memory area μεταξύ διεργασιών που δεν μοιράζονται κοινό πρόγονο, καθώς το mmap διαμοιράζει το VMA στην διεργασία και στα πιθανά παιδιά της, ενώ δεν μας δίνει την δυνατότητα να διευκρινίσουμε σε ποιό process id θέλουμε να διαμοιράσουμε την περιοχή εικονικής μνήμης.

2.2 Υλοποίηση χωρίς semaphores

Παραθέτουμε ξανά παρακάτω τα κύρια αλλαγμένα σημεία του κώδικα.

Αυτή τη φορά, αντί να χρησιμοποιούμε σημαφόρους για να συγχρονίσουμε την έξοδο της κάθε διεργασίας, βάζουμε την κάθε διεργασία να τυπώσει το color_val σε έναν διαμοιρασμένο buffer διαστάσεων y_chars × x_chars, και η γονεϊκή διαδικασία τις περιμένει και μόλις τερματίσουν όλες, τυπώνει όλα τα @ με τα χρώματα του buffer μαζί.

```
int NPROCS;

unsigned char **heap_shared_buf;
```

Στην συνάρτηση compute_and_output_mandel_line που καλούμε από κάθε παιδί, πλέον αποθηκεύουμε τα χρώματα στον buffer αντί να τυπώνουμε τα χρωματισμένα @ στην οθόνη. Μόλις σταματήσει να χρησιμοποιεί μια γραμμή του buffer ένα παιδί, την κάνει munmap από τον χάρτη μνήμης του.

```
void compute_and_output_mandel_line(int line)
{
  for( ; line < y_chars; line += NPROCS) {
    compute_mandel_line(line, heap_shared_buf[line]);
    safe_free(heap_shared_buf[line], x_chars);
  }
}
```

Στη main πλέον δημιουργούμε διαμοιρασμένη απεικόνιση για κάθε γραμμή του διδιάστατου buffer, φτιάχνουμε τα παιδιά που το καθένα εκτελεί την compute_and_output_mandel_line, συλλέγουμε ξανά τα pid's τους, τα περιμένουμε, και, αφού τερματίσουν όλα, τυπώνουμε για κάθε line με τα χρώματα του heap_shared_buf[line] τα @ μέσω της output_mandel_line. Στο τέλος, ελευθερώνουμε τις απεικονίσεις που φτιάξαμε και από τον χάρτη μνήμης της γονεϊκής διαδικασίας και επαναφέρουμε το αρχικό χρώμα στο τερματικό.

```
int main(int argc, char **argv){
  argument_handling(argc, argv);
  if(signal(SIGINT, sigintHandler) < 0){
    perror("Could not establish SIGINT handler");
    exit(1);
  }

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

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

    procs = (pid_t *) safe_malloc(NPROCS * sizeof(pid_t));
  heap_shared_buf = (unsigned char **) safe_malloc(y_chars*sizeof(unsigned char *));

  for(int line = 0; line < y_chars; line++){
    heap_shared_buf[line] = (unsigned char *) safe_shared_malloc(x_chars);
  }

    for(int i = 0; i < NPROCS; ++i) {
        pid_t p = fork();
    if(p < 0) die("fork error");
    if(p == 0){
      compute_and_output_mandel_line(i);
      return 0;
    }
    if(p > 0) procs[i] = p;
    }

    for(int i = 0; i < NPROCS; ++i) { //wait for all children
```

```
        int status;
    if(waitpid(procs[i], &status, WUNTRACED) == -1) die("waitpid error");
    }

  for(int line = 0; line < y_chars; line++){
    output_mandel_line(1, heap_shared_buf[line]);
    safe_free(heap_shared_buf[line], x_chars);
  }

  reset_xterm_color(1);
  safe_free(procs, NPROCS*sizeof(pid_t));
  safe_free(heap_shared_buf, y_chars*sizeof(char *));
  return 0;
}
```

1.

Σε αυτήν την υλοποίηση, ο συγχρονισμός επιτυγχάνεται με τα wait() που κάνει ο γονιός για τα παιδιά του πριν αρχίσει να τυπώνει όλους τους χαρακτήρες με τα χρώματα του buffer. Αν ο διαμοιρασμένος buffer μας είχε διαστάσεις NPROCS × x_chars αντί y_chars × x_chars, τότε το σχήμα συγχρονισμού θα άλλαζε. Θα έπρεπε μόλις όλα τα παιδιά γράψουν από 1 γραμμή στον buffer να στείλουν σήμα στον γονέα και να περιμένουν σήμα απάντησης από αυτόν. Ο γονέας, μόλις λάβει NPROCS σήματα, γνωρίζει ότι όλα τα παιδιά του έγραψαν στον buffer, οπότε μηδενίζει τον μετρητή σημάτων του, διαβάζει τον buffer και τυπώνει τους χαρακτήρες κάθε γραμμής με τη σειρά με τα χρώματα του buffer. Μόλις τυπώσει την γραμμή line, στέλνει σήμα στο παιδί line%NPROCS να συνεχίσει τον υπολογισμό και το τύπωμα στον buffer και η διαδικασία συνεχίζεται.

Εναλλακτικά, αν θέλαμε να αποφύγουμε την διαδιεργασιακή επικοινωνία μέσω των σημάτων, θα μπορούσαν τα παιδιά να τυπώνουν μόνο 1 γραμμή στον buffer και μετά να τερματίζουν, οπότε ο γονέας θα περιμένει τα παιδιά, μόλις τερματίσουν όλα θα τυπώνει τους χαρακτήρες με τα χρώματα του buffer και θα ξαναδημιουργεί τα παιδιά για τις επόμενες NPROCS γραμμές, μέχρι να τυπωθεί όλο το σύνολο Mandelbrot.

3 (Προαιρετική) Επέκταση Άσκησης 1

Παραθέτουμε τα κύρια αλλαγμένα σημεία του κώδικα.

Δηλώνουμε τον μετρητή που θα επεξεργάζονται όλα τα παιδιά και τον σημαφόρο που θα χρησιμοποιούν για τον συγχρονισμό τους ως pointers για να τα απεικονίσουμε σε διαμοιρασμένη μνήμη μέσω της mmap.

```
int* count;
sem_t* sema;
```

Ορίζουμε τη διαδικασία safe_free για να ελευθερώνουμε τις απεικονίσεις που φτιάξαμε από τον χάρτη μνήμης μέσω της munmap.

```
void safe_free(void *ptr, size_t size){
  if(munmap(ptr, size) == -1) die("munmap error");
}
```

Ορίζουμε signal handler για το σήμα SIGINT που αντί να τερματίζει το πρόγραμμά μας, τυπώνει το πλήθος των διεργασιών παιδιών και το πλήθος εμφανίσεων του χαρακτήρα που έχουν μετρηθεί μέχρι στιγμής. Αφού ο μετρητής βρίσκεται σε διαμοιρασμένη μνήμη και η τιμή του ενημερώνεται διαρκώς από τα παιδιά αντί μία φορά από τον γονέα στο τέλος, το μήνυμα αυτό έχει νόημα.

```
void sigintHandler(int sig_num) {
  if(signal(SIGINT, sigintHandler) < 0) die("Could not establish SIGINT handler");
  sleep(1); //To be able to test the SIGINT handler
  char buff[1024];
  snprintf(buff, sizeof(buff),"There are %d processes reading the input file in parallel.\nCounted %d
    occurences of the character so far.\n", P, *count);
  print(1, buff);
}
```

Το κάθε παιδί δέχεται ως είσοδο το file descriptor του input file, το πόσα bytes πρέπει να διαβάσει και τον χαρακτήρα που πρέπει να μετρήσει. Κάθε φορά που εντοπίζει τον χαρακτήρα αυτόν, αυξάνει κατά 1 τον μετρητή στη διαμοιρασμένη μνήμη. Για να επιλυθούν τα race conditions που δημιουργούνται στον κώδικα και να επιτευχθεί συγχρονισμός, αφού τα παιδιά μπορεί ταυτόχρονα να προσπαθήσουν να αλλάξουν την τιμή του count, βάζουμε την πρόσβαση και αλλαγή στον count (κρίσιμο τμήμα) ανάμεσα σε sem_wait και sem_post για τον ίδιο σημαφόρο, ώστε να μπορεί μόνο 1 διεργασία να επεξεργάζεται ταυτόχρονα το count κάθε φορά. Στο τέλος, ελευθερώνουμε τις απεικονίσεις του σημαφόρου και του count από τον χάρτη μνήμης του κάθε παιδιού.

```c
void child(int fdr, size_t bytes_to_read, char c2c) {
  ssize_t rcnt = 0;
  char buff[1024];
  while(bytes_to_read) {
    rcnt = read(fdr, buff, min(bytes_to_read, sizeof(buff) - 1));
      if(rcnt == 0) break; //end of file
      if(rcnt == -1) die("Failed to read file\n");
    buff[rcnt] = '\0';
    bytes_to_read -= rcnt;
    for(size_t i = 0; i < rcnt; i++){
      if(buff[i] == c2c){
        sem_wait(sema);
        (*count)++;
        sem_post(sema);
      }
    }
  }
  safe_free(sema, sizeof(sem_t));
  safe_free(count, sizeof(int));
}
```

Στη main, αφού κάνουμε έλεγχο ορισμάτων και ανοίξουμε τα αρχεία, εγκαθιστούμε τον SIGINT handler, δημιουργούμε τις απεικονίσεις σε διαμοιρασμένη μνήμη για το count και τον σημαφόρο, κάνουμε fork, καλώντας την συνάρτηση child για κάθε παιδί και ύστερα τερματίζοντας, ενώ ο γονέας περιμένει τα παιδιά, καταστρέφει τον σημαφόρο και ελευθερώνει την απεικόνισή του, κλείνει το αρχείο εισόδου, τυπώνει το αποτέλεσμα της καταμέτρησης στο αρχείο εξόδου, ελευθερώνει την απεικόνιση του μετρητή και κλείνει τα αρχείο εξόδου.

```c
int main(int argc, char **argv) {
  argument_handling(argc, argv);

  int fdr, fdw;
  int oflags, mode;
  oflags = O_CREAT | O_WRONLY | O_TRUNC;
  mode = S_IRUSR | S_IWUSR;
  if((fdr = open(argv[1], O_RDONLY)) == -1) die("Problem opening file to read\n");
  if((fdw = open(argv[2], oflags, mode)) == -1) die("Problem opening file to write\n");

  struct stat st;
  if(stat(argv[1], &st) < 0) die("Stat error\n");
  if(signal(SIGINT, sigintHandler) < 0) die("Could not establish SIGINT handler");
  off_t size = st.st_size;

  P = CPUCORES;
  off_t batch_size = size / P;
  off_t extra_left = size - P * batch_size;
  pid_t p;

  if((count = (int *) mmap(NULL, sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS, -1, 0))
    == MAP_FAILED)
    die("mmap error");
  *count = 0;
  if((sema = (sem_t *) mmap(NULL, sizeof(sem_t), PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS, -1, 0)
    ) == MAP_FAILED)
    die("mmap error");
  sem_init(sema, 1, 1);

  for(int i = 0; i < P; i++) {
    size_t bytes_to_read = batch_size + (i < extra_left);
    p = fork();
    if(p < 0) die("fork error");
    else if(p == 0) {
      child(fdr, bytes_to_read, argv[3][0]);
      exit(0);
    }
  }

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

  safe_free(sema, sizeof(sem_t));
  if(sem_destroy(sema)) die("sem_destroy error");
  close(fdr);

  char buff[1024];
  snprintf(buff, sizeof(buff), "The character '%c' appears %d times in file %s.\n", argv[3][0], *count,
     argv[1]);
  print(fdw, buff);
```

```
48
49    safe_free(count, sizeof(int));
50    close(fdw);
51  }
```