Λειτουργικά Συστήματα Υπολογιστών

3η Εργαστηριακή Άσκηση

oslab064	
Νικόλαος Γιαννόπουλος	03122086
Κωνσταντίνος Καργάκος	03122166

Πίνακας περιεχομένων

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

1 - Print the virtual address space layout of this process.

```
show_maps();
```

```
Step 1: Print the virtual address space map of this process [185107].


Virtual Memory Map of process [185107]:
55a5ae801000-55a5ae802000 r--p 00000000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae802000-55a5ae803000 r-xp 00001000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae803000-55a5ae804000 r--p 00002000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae804000-55a5ae805000 r--p 00002000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae805000-55a5ae806000 rw-p 00003000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5aef00000-55a5aef21000 rw-p 00000000 00:00 0                          [heap]
7efd71577000-7efd71599000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71599000-7efd716f2000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd716f2000-7efd71741000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71741000-7efd71745000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71745000-7efd71747000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71747000-7efd7174d000 rw-p 00000000 00:00 0
7efd71752000-7efd71753000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd71753000-7efd71773000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd71773000-7efd7177b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177c000-7efd7177d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177d000-7efd7177e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177e000-7efd7177f000 rw-p 00000000 00:00 0
7ffc71850000-7ffc71871000 rw-p 00000000 00:00 0                          [stack]
7ffc718ec000-7ffc718f0000 r--p 00000000 00:00 0                          [vvar]
7ffc718f0000-7ffc718f2000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------
```

2 - Use mmap to allocate a buffer of 1 page and print the map again. Store buffer in heap_private_buf.

```
   heap_private_buf = mmap(NULL, buffer_size, PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS, -1, 0);
   if (heap_private_buf == MAP_FAILED)
       die("mmap");
   printf("heap_private_buf: %p\n", heap_private_buf);
   show_va_info((uint64_t)heap_private_buf);
   show_maps();
```

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

heap_private_buf: 0x7efd7177b000
7efd7177b000-7efd7177c000 rw-p 00000000 00:00 0

Virtual Memory Map of process [185107]:
55a5ae801000-55a5ae802000 r--p 00000000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae802000-55a5ae803000 r-xp 00001000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae803000-55a5ae804000 r--p 00002000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae804000-55a5ae805000 r--p 00002000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae805000-55a5ae806000 rw-p 00003000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5aef00000-55a5aef21000 rw-p 00000000 00:00 0                          [heap]
7efd71577000-7efd71599000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71599000-7efd716f2000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd716f2000-7efd71741000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71741000-7efd71745000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71745000-7efd71747000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71747000-7efd7174d000 rw-p 00000000 00:00 0
7efd71752000-7efd71753000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd71753000-7efd71773000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd71773000-7efd7177b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177b000-7efd7177c000 rw-p 00000000 00:00 0
7efd7177c000-7efd7177d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177d000-7efd7177e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177e000-7efd7177f000 rw-p 00000000 00:00 0
7ffc71850000-7ffc71871000 rw-p 00000000 00:00 0                          [stack]
7ffc718ec000-7ffc718f0000 r--p 00000000 00:00 0                          [vvar]
7ffc718f0000-7ffc718f2000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------
```

3 - Find the physical address of the first page of your buffer in main memory. What do you see?

```
pa = get_physical_address((uint64_t)heap_private_buf);
printf("Physical address of heap_private_buf: %lu\n", pa);
```

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

VA[0x7efd7177b000] is not mapped; no physical memory allocated.
Physical address of heap_private_buf: 0
```

Παρατηρούμε ότι δεν ότι ο buffer δεν είναι συνδεδεμένος με κάποια διεύθυνση στη φυσική μνήμη. Το λειτουργικό σύστημα δεν δεσμεύει θέση στη φυσική μνήμη εφόσον ο buffer έχει μόνο δημιουργηθεί αλλά δεν περιέχει τίποτα ακόμα.

4 - Write zeros to the buffer and repeat Step 3.

```
for (uint64_t i = 0; i < buffer_size; i++)
       heap_private_buf[i] = 0;
pa = get_physical_address((uint64_t)heap_private_buf);
printf("Physical address of heap_private_buf: %lu\n", pa);
```

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

Physical address of heap_private_buf: 5686554624
```

Βλέπουμε ότι τώρα που γεμίσαμε τον buffer το λειτουργικό σύστημα έχει δεσμεύσει για αυτόν θέση στη φυσική μνήμη. Αυτή η λειτουργία ονομάζεται page-on-demand, δηλαδή, όταν μία διεργασία δεσμεύει εικονική μνήμη, το λειτουργικό δεν την αντιστοιχίζει σε φυσική μνήμη, παρά μόνο όταν την χρειαστεί (π.χ. όταν γίνει εγγραφή)

5 - Use mmap(2) to map file.txt (memory-mapped files) and print its content. Use file_shared_buf.

```
   fd = open("file.txt", O_RDONLY);
   if (fd < 0)
       die("open");

   file_size = lseek(fd, 0, SEEK_END);
   file_shared_buf = mmap(NULL, file_size, PROT_READ, MAP_SHARED, fd, 0);
   if (file_shared_buf == MAP_FAILED)
       die("mmap");

   if (write(1, file_shared_buf, file_size) != file_size) {
       perror("write");
   }

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

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

Hello everyone!
7efd71751000-7efd71752000 r--s 00000000 00:26 8281700                    /home/oslab/oslab064/lab03/mmap/file.txt

Virtual Memory Map of process [185107]:
55a5ae801000-55a5ae802000 r--p 00000000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae802000-55a5ae803000 r-xp 00001000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae803000-55a5ae804000 r--p 00002000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae804000-55a5ae805000 r--p 00002000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae805000-55a5ae806000 rw-p 00003000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5aef00000-55a5aef21000 rw-p 00000000 00:00 0                          [heap]
7efd71577000-7efd71599000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71599000-7efd716f2000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd716f2000-7efd71741000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71741000-7efd71745000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71745000-7efd71747000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71747000-7efd7174d000 rw-p 00000000 00:00 0
7efd71751000-7efd71752000 r--s 00000000 00:26 8281700                    /home/oslab/oslab064/lab03/mmap/file.txt
7efd71752000-7efd71753000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd71753000-7efd71773000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd71773000-7efd7177b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177b000-7efd7177c000 rw-p 00000000 00:00 0
7efd7177c000-7efd7177d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177d000-7efd7177e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177e000-7efd7177f000 rw-p 00000000 00:00 0
7ffc71850000-7ffc71871000 rw-p 00000000 00:00 0                          [stack]
7ffc718ec000-7ffc718f0000 r--p 00000000 00:00 0                          [vvar]
7ffc718f0000-7ffc718f2000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------
```

6 - Use mmap(2) to allocate a shared buffer of 1 page. Use heap_shared_buf.

```
   heap_shared_buf = mmap(NULL, buffer_size, PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);
   if (heap_shared_buf == MAP_FAILED)
       die("mmap");
   printf("heap_shared_buf: %p\n", heap_shared_buf);
   show_va_info((uint64_t)heap_shared_buf);
   show_maps();

   for (uint64_t i = 0; i < buffer_size; i++)
       heap_shared_buf[i] = 0;

   pa = get_physical_address((uint64_t)heap_shared_buf);
   printf("Physical address of heap_shared_buf: %lu\n", pa);
```

```
Step 6: Use mmap(2) to allocate a shared buffer of size equal to 1 page. Initialize the buffer and print the new m

heap_shared_buf: 0x7efd71750000
7efd71750000-7efd71751000 rw-s 00000000 00:01 265                      /dev/zero (deleted)

Virtual Memory Map of process [185107]:
55a5ae801000-55a5ae802000 r--p 00000000 00:26 8281706                  /home/oslab/oslab064/lab03/mmap/mmap
55a5ae802000-55a5ae803000 r-xp 00001000 00:26 8281706                  /home/oslab/oslab064/lab03/mmap/mmap
55a5ae803000-55a5ae804000 r--p 00002000 00:26 8281706                  /home/oslab/oslab064/lab03/mmap/mmap
55a5ae804000-55a5ae805000 r--p 00002000 00:26 8281706                  /home/oslab/oslab064/lab03/mmap/mmap
55a5ae805000-55a5ae806000 rw-p 00003000 00:26 8281706                  /home/oslab/oslab064/lab03/mmap/mmap
55a5aef00000-55a5aef21000 rw-p 00000000 00:00 0                        [heap]
7efd71577000-7efd71599000 r--p 00000000 fe:01 144567                   /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71599000-7efd716f2000 r-xp 00022000 fe:01 144567                   /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd716f2000-7efd71741000 r--p 0017b000 fe:01 144567                   /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71741000-7efd71745000 r--p 001c9000 fe:01 144567                   /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71745000-7efd71747000 rw-p 001cd000 fe:01 144567                   /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71747000-7efd7174d000 rw-p 00000000 00:00 0
7efd71750000-7efd71751000 rw-s 00000000 00:01 265                      /dev/zero (deleted)
7efd71751000-7efd71752000 r--s 00000000 00:26 8281700                  /home/oslab/oslab064/lab03/mmap/file.txt
7efd71752000-7efd71753000 r--p 00000000 fe:01 144563                   /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd71753000-7efd71773000 r-xp 00001000 fe:01 144563                   /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd71773000-7efd7177b000 r--p 00021000 fe:01 144563                   /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177b000-7efd7177c000 rw-p 00000000 00:00 0
7efd7177c000-7efd7177d000 r--p 00029000 fe:01 144563                   /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177d000-7efd7177e000 rw-p 0002a000 fe:01 144563                   /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177e000-7efd7177f000 rw-p 00000000 00:00 0
7ffc71850000-7ffc71871000 rw-p 00000000 00:00 0                        [stack]
7ffc718ec000-7ffc718f0000 r--p 00000000 00:00 0                        [vvar]
7ffc718f0000-7ffc718f2000 r-xp 00000000 00:00 0                        [vdso]
------------------------------------------------------

Physical address of heap_shared_buf: 5278539776
```

7 - Print parent's and child's maps. What do you see?

```c
printf("Parent's pid: %d\n", getpid());
show_maps();
```

```c
printf("Child's pid: %d\n", getpid());
show_maps();
```

Παρατηρούμε παρακάτω ότι η εικονικές μνήμες της γονικής διεργασίας και της διεργασίας παιδιού είναι ίδιες μεταξύ τους και ίδιες με την εικονική μνήμη πριν τη δημιουργία της νέας διεργασιών. Αυτό συμβαίνει γιατί κατά τη δημιουργία της νέας διεργασίας αντιγράφεται η εικονική μνήμη του πατέρα σε αυτή.

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

Parent's pid: 185107

Virtual Memory Map of process [185107]:
55a5ae801000-55a5ae802000 r--p 00000000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae802000-55a5ae803000 r-xp 00001000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae803000-55a5ae804000 r--p 00002000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae804000-55a5ae805000 r--p 00002000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae805000-55a5ae806000 rw-p 00003000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5aef00000-55a5aef21000 rw-p 00000000 00:00 0                          [heap]
7efd71577000-7efd71599000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71599000-7efd716f2000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd716f2000-7efd71741000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71741000-7efd71745000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71745000-7efd71747000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71747000-7efd7174d000 rw-p 00000000 00:00 0
7efd71750000-7efd71751000 rw-s 00000000 00:01 265                        /dev/zero (deleted)
7efd71751000-7efd71752000 r--s 00000000 00:26 8281700                    /home/oslab/oslab064/lab03/mmap/file.txt
7efd71752000-7efd71753000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd71753000-7efd71773000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd71773000-7efd7177b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177b000-7efd7177c000 rw-p 00000000 00:00 0
7efd7177c000-7efd7177d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177d000-7efd7177e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177e000-7efd7177f000 rw-p 00000000 00:00 0
7ffc71850000-7ffc71871000 rw-p 00000000 00:00 0                          [stack]
7ffc718ec000-7ffc718f0000 r--p 00000000 00:00 0                          [vvar]
7ffc718f0000-7ffc718f2000 r-xp 00000000 00:00 0                          [vdso]
--------------------------------------------------------
```

```
Child's pid: 185167

Virtual Memory Map of process [185167]:
55a5ae801000-55a5ae802000 r--p 00000000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae802000-55a5ae803000 r-xp 00001000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae803000-55a5ae804000 r--p 00002000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae804000-55a5ae805000 r--p 00002000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5ae805000-55a5ae806000 rw-p 00003000 00:26 8281706                    /home/oslab/oslab064/lab03/mmap/mmap
55a5aef00000-55a5aef21000 rw-p 00000000 00:00 0                          [heap]
7efd71577000-7efd71599000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71599000-7efd716f2000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd716f2000-7efd71741000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71741000-7efd71745000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71745000-7efd71747000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7efd71747000-7efd7174d000 rw-p 00000000 00:00 0
7efd71750000-7efd71751000 rw-s 00000000 00:01 265                        /dev/zero (deleted)
7efd71751000-7efd71752000 r--s 00000000 00:26 8281700                    /home/oslab/oslab064/lab03/mmap/file.txt
7efd71752000-7efd71753000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd71753000-7efd71773000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd71773000-7efd7177b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177b000-7efd7177c000 rw-p 00000000 00:00 0
7efd7177c000-7efd7177d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177d000-7efd7177e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7efd7177e000-7efd7177f000 rw-p 00000000 00:00 0
7ffc71850000-7ffc71871000 rw-p 00000000 00:00 0                          [stack]
7ffc718ec000-7ffc718f0000 r--p 00000000 00:00 0                          [vvar]
7ffc718f0000-7ffc718f2000 r-xp 00000000 00:00 0                          [vdso]
```

8 - Get the physical memory address for heap_private_buf.

```
pa = get_physical_address((uint64_t)heap_private_buf);
printf("Physical address of heap_private_buf (parent): %lu\n", pa);
```

```
pa = get_physical_address((uint64_t)heap_private_buf);
printf("Physical address of heap_private_buf (child): %lu\n", pa);
```

```
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): 5686554624
Physical address of heap_private_buf (child): 5686554624
```

Αμέσως μετά το fork η εικονική μνήμη του πατέρα αντιγράφεται στην εικονική μνήμη του παιδιού. Οι διευθύνσεις των εικονικών μνημών των δυο διεργασιών αρχικά είναι συνδεδεμένες με τις ίδιες ακριβώς διευθύνσεις στη φυσική μνήμη. Γι' αυτό το λόγο, η διεύθυνση του private buffer κάθε διεργασίας στη φυσική μνήμη είναι ίδια.

9 - Write to heap_private_buf. What happened?

```
pa = get_physical_address((uint64_t)heap_private_buf);
printf("Physical address of heap_private_buf (parent): %lu\n", pa);
```

```
heap_private_buf[0] = 1;
pa = get_physical_address((uint64_t)heap_private_buf);
printf("Physical address of heap_private_buf (child): %lu\n", pa);
```

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

Physical address of heap_private_buf (parent): 5686554624
Physical address of heap_private_buf (child): 5440557056
```

Τώρα γράφουμε στον private buffer από τη διεργασία παιδί και βλέπουμε ότι πλέον στη διεργασία παιδί η διεύθυνση του buffer στη φυσική μνήμη άλλαξε. Η διαδικασία αυτή λέγεται copy-on-write. Όταν δημιουργείται, λοιπόν, μια νέα διεργασία η εικονική μνήμη αντιγράφεται και οι δύο εικονικές μνήμες δείχνουν στις ίδιες διευθύνσεις στη φυσική μνήμη. Αυτό αλλάζει όταν μια διεργασία επεξεργαστεί ένα στοιχείο, το οποίο πλέον θα αποθηκευτεί σε διαφορετική θέση στη φυσική μνήμη μιας και το περιεχόμενό του θα αλλάξει.

10 - Get the physical memory address for heap_shared_buf.

```
pa = get_physical_address((uint64_t)heap_shared_buf);
printf("Physical address of heap_shared_buf (parent): %lu\n", pa);
```

```
heap_shared_buf[0] = 1;
pa = get_physical_address((uint64_t)heap_shared_buf);
printf("Physical address of heap_shared_buf (child): %lu\n", pa);
```

```
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 (parent): 5278539776
Physical address of heap_shared_buf (child): 5278539776
```

Παρατηρούμε ότι, σε αντίθεση με τον private buffer, ο shared buffer παρόλο που επεξεργάστηκε από τη διεργασία παιδί παραμένει στην ίδια φυσική διεύθυνση και στις δύο διεργασίες. Αυτό συμβαίνει γιατί έχουμε δημιουργήσει αυτόν τον buffer με flag [**MAP_SHARED**]και έχει τη δυνατότητα να μοιράζεται μεταξύ των διεργασιών. Έτσι, μια αλλαγή στον shared buffer από μια διεργασία είναι εμφανή και στην άλλη διεργασία, εφόσον η θέση του στη φυσική μνήμη παραμένει ίδια και στις δύο διεργασίες.

11 - Disable writing on the shared buffer for the child.

```
printf("Parent's map after mprotect:\n");
show_va_info((uint64_t)heap_shared_buf);
```

```
if (mprotect(heap_shared_buf, buffer_size, PROT_READ) == -1)
        die("mprotect");
printf("Child's map after mprotect:\n");
show_va_info((uint64_t)heap_shared_buf);
```

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

Parent's map after mprotect:
7efd71750000-7efd71751000 rw-s 00000000 00:01 265                      /dev/zero (deleted)
Child's map after mprotect:
7efd71750000-7efd71751000 r--s 00000000 00:01 265                      /dev/zero (deleted)
```

Η απαγόρευση της εγγραφής στον shared buffer για τη διεργασία παιδί φαίνεται από τη διεύθυνση του buffer στην εικονική μνήμη του παιδιού όπου εμφανίζει δικαιώματα **r--s** και όχι **rw-s** όπως στην διεργασία πατέρα.

12 - Free all buffers for parent and child.

```
if (munmap(heap_private_buf, buffer_size) == -1)
        perror("munmap: heap_private_buf");
if (munmap(heap_shared_buf, buffer_size) == -1)
        perror("munmap: heap_shared_buf");
if (munmap(file_shared_buf, file_size) == -1)
        perror("munmap: file_shared_buf");
```

2 – Παράλληλος υπολογισμός Mandelbrot με διεργασίες αντί για νήματα

2.1 – Semaphores πάνω από διαμοιραζόμενη μνήμη

Συμπληρώνουμε τη συνάρτηση **create_shared_memory_area()** ως εξής:

```
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;

    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;
}
```

Πλέον η συνάρτηση καταλαμβάνει θέση στη μνήμη, η οποία είναι μοιραζόμενη μεταξύ των διεργασιών [**MAP_SHARED**].

Δημιουργούμε μια λίστα από **NPROCS** δείκτες σε σημαφόρους. Δηλάδη χρησιμοποιούμε ένα σημαφόρο για κάθε διεργασία.

```
static sem_t **sems;
static int nprocs;
```

Δεσμεύουμε μνήμη για τη λίστα των δεικτών με τη **malloc()** και στη συνέχεια αποθηκεύουμε κάθε σημαφόρο σε κοινή μνήμη με τη συνάρτηση **create_shared_memory_area()**. Επίσης, θέτουμε στον πρώτο σημαφόρο τιμή 1, ενώ στους υπόλοιπους τιμή 0. Ο τρόπος υλοποίησης του συγχρονισμού με τους σημαφόρους στη συνάρτηση **compute_and_output()** που έχουμε φτιάξει παραμένει ίδιος με αυτόν της άσκησης 2.

```
    sems = malloc(nprocs * sizeof(sem_t *));
    if (!sems) {
       perror("malloc sems");
       exit(1);
    }
```

```
    for (int i = 0; i < nprocs; i++) {
        sems[i] = create_shared_memory_area(sizeof(sem_t));
        if (sem_init(sems[i], 1, i == 0 ? 1 : 0) == -1) {
            perror("sem_init");
            exit(1);
        }
    }
```

Δημιουργούμε **NPROCS** διεργασίες με τη **fork()**, οι οποίες εκτελούν τη συνάρτηση **compute_and_output()** και υπολογίζουν και εκτυπώνουν το αποτέλεσμα με συγχρονισμό. Η διεργασία πατέρας περιμένει τις διεργασίες παιδιά να τελειώσουν και έπειτα καταστρέφει όλους τους σημαφόρους και αποδεσμεύει τη θέση κοινής μνήμης για κάθε σημαφόρο.

```
    for (int i = 0; i < nprocs; i++) {
        pid_t pid = fork();
        if (pid < 0) {
            perror("fork");
            exit(1);
        }
        if (pid == 0) {
            compute_and_output(i);
            exit(0);
        }
    }

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

    for (int i = 0; i < nprocs; i++) {
        if (sem_destroy(sems[i]) == -1) {
            perror("sem_destroy");
            exit(1);
        }
        destroy_shared_memory_area(sems[i], sizeof(sem_t));
    }
    free(sems);
    reset_xterm_color(1);
    return 0;
```

Ερωτήσεις:

1. Ανάμεσα στις δύο παραλληλοποιημένες υλοποιήσεις (threads vs processes) περιμένουμε να έχει καλύτερη επίδοση η υλοποίηση με τα threads. Αυτό γιατί τα threads μοιράζονται τον ίδιο χώρο εικονικής μνήμης και άρα η ανταλλαγή δεδομένων μεταξύ τους γίνεται άμεσα σε αντίθεση με τα processes. Επιπλέον, η δημιουργία threads είναι αποδοτική από τη δημιουργία διεργασιών, καθώς δεν απαιτείται αντιγραφή του περιβάλλοντος εκτέλεσης.

Η υλοποίηση με τα processes χρησιμοποιεί semaphores, που βρίσκονται σε διαμοιραζόμενη μνήμη, ώστε να έχουν την δυνατότητα να χρησιμοποιούνται από όλες τις διεργασίες. Αυτό προϋποθέτει επιπλέον διαχείριση της κοινής μνήμης (χρήση mmap), καθώς και δαπανηρή μετάβαση στη λειτουργία kernel σε κάθε πρόσβαση πάνω στον σημαφόρο.

2. Με την εκάστοτε υλοποίηση της συνάρτησης **create_shared_memory_area()**, η οποία χρησιμοποιεί την **mmap()** με flag [`MAP_ANONYMOUS`], δεν μπορεί να διαμοιραστεί η μνήμη μεταξύ διεργασιών που δεν έχουν κοινό ancestor. Αυτό γιατί ο δείκτης που επιστρέφει η mmap αποθηκεύεται σε μεταβλητή της διεργασίας πατέρα, στην οποία καλούμε τη **mmap()** και μπορεί να περάσει στις διεργασίες παιδιά μόνον αυτής της διεργασίας.
   Εναλλακτικά, εάν στην κλήση της **mmap()** δεν βάλουμε το flag [`MAP_ANONYMOUS`], μπορούμε να περάσουμε ένα file descriptor από ένα αρχείο στο σύστημα αρχείων και έτσι διεργασίες που δεν έχουν κοινό ancestor μπορούν να έχουν πρόσβαση στην κοινή μνήμη με το mmap() μέσω αυτού του αρχείου.

2.2 – Υλοποίηση χωρίς semaphores

Τροποποιούμε την υλοποίηση του προηγούμενου ερωτήματος αφαιρώντας τους σημαφόρους και χρησιμοποιώντας έναν δισδιάστατο buffer, διαστάσεων **y_chars * x_chars**, για τον οποίο δεσμεύουμε κοινή μνήμη με τη συνάρτηση **create_shared_memory_area()**.

```
int (*frame_buffer)[x_chars];
size_t buffer_size = y_chars * x_chars * sizeof(int);
frame_buffer = (int (*)[x_chars])create_shared_memory_area(buffer_size);
```

Δημιουργούμε πάλι **NPROCS** διεργασίες με τη **fork()**, οι οποίες εκτελούν τη συνάρτηση **child_work()**, υπολογίζουν το αποτέλεσμα κάθε γραμμής και το αποθηκεύουν στον buffer. Σε αυτή την περίπτωση την εκτύπωση του αποτελέσματος αναλαμβάνει η διεργασία πατέρας, η οποία, αφού τελειώσουν όλες οι διεργασίες παιδιά, εκτυπώνει κάθε γραμμή που είναι αποθηκευμένη στον buffer. Τέλος, αποδεσμεύεται η κοινή μνήμη του buffer.

```
for (int i = 0; i < nprocs; i++) {
    pid_t pid = fork();
    if (pid < 0) {
        perror("fork");
        exit(1);
    }
    if (pid == 0) {
      child_work(i, frame_buffer);
        exit(0);
    }
}

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

```
    }

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

    destroy_shared_memory_area(frame_buffer, buffer_size);
    reset_xterm_color(1);
    return 0;
```

Συνάρτηση **void child_work(int pid, int (*frame_buffer)[x_chars])**:

```
void child_work(int pid, int (*frame_buffer)[x_chars]) {
    int color_val[x_chars];

    for (int line = pid; line < y_chars; line += nprocs) {
       compute_mandel_line(line, color_val);

       memcpy(frame_buffer[line], color_val, sizeof(color_val));
    }
}
```

Κάθε διεργασία μπαίνει στο for loop, ώστε να υπολογίσει τις γραμμές *i, i+n, i+2*n, ...* , τις οποίες και αποθηκεύει στην αντίστοιχη γραμμή του buffer με τη συνάρτηση **memcpy()** (αντιγράφει το περιεχόμενο του **color_val** στη γραμμή **line** του **frame_buffer**).

Ερωτήσεις:

1. Σε αυτή την υλοποίηση δεν απαιτείται σχήμα συγχρονισμού, μιας και κάθε γραμμή που υπολογίζεται από τις διεργασίες αποθηκεύεται σε αντίστοιχη θέση στον μοιραζόμενο buffer και έπειτα η διεργασία πατέρας εκτυπώνει το περιεχόμενο του buffer.
   Εάν ο buffer είχε διαστάσεις **NPROCS * x_chars**, τότε κάθε φορά που όλες οι διεργασίες θα υπολόγιζαν μία γραμμή ο buffer θα γέμιζε και θα έπρεπε να τυπωθεί το περιεχόμενό του, πριν οι διεργασίες προχωρήσουν στον υπολογισμό των επόμενων γραμμών. Σε αυτή την περίπτωση θα έπρεπε να φροντίσουμε για τον συγχρονισμό, ώστε η διεργασία πατέρας να τυπώνει την κατάλληλη στιγμή αλλά και οι διεργασίες παιδιά να γράφουν εκ νέου τις γραμμές στον buffer, αφού οι προηγούμενες έχουν τυπωθεί.

3 – Επέκταση Άσκησης 1

Για την επέκταση της άσκησης αυτής δημιουργούμε έναν δείκτη **total_count**, ο οποίος θα αυξάνεται κατά 1 κάθε φορά που μια διεργασία βρίσκει τον χαρακτήρα στο αρχείο, καθώς και έναν δείκτη για τον σημαφόρο που θα χρησιμοποιήσουμε για τον συγχρονισμό.

```
static sem_t *sem;

int active_children = 0;
int *total_count;
```

Δεσμεύουμε μοιραζόμενη μνήμη για το **total_count**, έτσι ώστε να έχουν πρόσβαση στη μεταβλητή όλες οι διεργασίες Επίσης αφαιρούμε όλα τα **pipes** που είχαμε χρησιμοποιήσει, αφού πλέον κάθε διεργασία θα προσθέτει το αποτέλεσμα της στο **total_count**.

```
total_count = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
                MAP_SHARED | MAP_ANONYMOUS, -1, 0);
if (total_count == MAP_FAILED) {
    perror("Problem mapping memory");
    exit(1);
}
*total_count = 0;
```

Επίσης, δεσμεύουμε μοιραζόμενη μνήμη για το σημαφόρο, έτσι ώστε να μπορεί να χρησιμοποιηθεί για τον συγχρονισμό μεταξύ των διεργασιών. Αρχικά στο σημαφόρο θέτουμε την τιμή 1.

```
sem = mmap(NULL, sizeof(sem_t), PROT_READ | PROT_WRITE,
         MAP_SHARED | MAP_ANONYMOUS, -1, 0);
if (sem == MAP_FAILED) {
    perror("Problem mapping semaphore memory");
    exit(1);
}
sem_init(sem, 1, 1);
```

Κάθε διεργασία διαβάζει από το αρχείο το κομμάτι που της αντιστοιχεί και αν βρει χαρακτήρα ίδιο με αυτόν που αναζητούμε αυξάνει το **total_count** κατά 1. Η αύξηση γίνεται ανάμεσα στις εντολές **sem_wait()** και **sem_post()**, ώστε να εξασφαλίσουμε ότι κάθε στιγμή μόνο μία διεργασία θα αυξάνει τον κοινό μετρητή **total_count**.

```
            // Read from start_index to end_index
            for (off_t index = start_index; index < end_index; index++) {
                if (read(fdr, &cc, 1) == -1) {
                    perror("Problem reading from file");
                    close(fdr);
                    exit(1);
                }
                if (cc == c2c) {
                    sem_wait(sem);
                    *total_count += 1;
                    sem_post(sem);
                }
            }
```

Αφού οι διεργασίες παιδιά τελειώσουν, τότε η διεργασία πατέρας διαβάζει το αποτέλεσμα από τη μεταβλητή **total_count**.

Τέλος, καταστρέφεται ο σημαφόρος και αποδεσμεύεται κοινή μνήμη για τη μεταβλητή **total_count** και για τον δείκτη του σημαφόρου.

```
if (munmap(total_count, sizeof(int)) == -1) {
    perror("munmap");
    exit(1);
}
```

```c
if (sem_destroy(sem) == -1) {
    perror("sem_destroy");
    exit(1);
}
if (munmap(sem, sizeof(sem_t)) == -1) {
    perror("munmap sem");
    exit(1);
}
```

Επιπλέον, τροποποιούμε τη συνάρτηση που καλείται σε περίπτωση που το πρόγραμμα δεχτεί το σήμα **Control+C**, ώστε εκτός από τον αριθμό των ενεργών διεργασιών να εμφανίζεται και η τρέχουσα τιμή του κοινού μετρήτη **total_count**. Η μεταβλητή αυτή περιέχει κάθε στιγμή τον αριθμό εμφανίσεων του χαρακτήρα που έχουν εντοπιστεί από όλες τις διεργασίες μέχρι τη συγκεκριμένη χρονική στιγμή.

```c
// Signal handler for SIGINT
void sighandler(int signum) {
    printf("\nActive search processes: %d\n", active_children);
    printf("Total count now: %d\n", *total_count);
}
```