ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ ΥΠΟΛΟΓΙΣΤΩΝ

ΑΝΑΦΟΡΑ 3ης ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΑΣΚΗΣΗΣ

Στοιχεία Ομάδας

- Αναγνωριστικό: oslab005
- Μέλος 1ο: Πέππας Μιχαήλ – Αθανάσιος, Α.Μ: 03121026
- Μέλος 2ο: Σαουνάτσος Ανδρέας, Α.Μ: 03121197
- Ημερομηνία Παράδοσης Αναφοράς: 17.05.2024

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

Αρχικά, παραθέτουμε τον συμπληρωμένο κώδικα της άσκησης – του αρχείου mmap.c – ακολούθως, ώστε να μπορεί να γίνει εύκολη αναφορά κατά την παράθεση των απαντήσεων στα ζητούμενα ερωτήματα. Ωστόσο, για λόγους ευκολίας, τα καίρια σημεία του κώδικα θα παρατίθενται εκ νέου σε κάθε υποερώτημα, όπως και η αντίστοιχη έξοδος. Αξίζει να επισημάνουμε ότι κάθε κώδικας που συμπληρώνουμε στο αρχείο φροντίζει να ελέγχει και να προλαμβάνει πιθανά λάθη, όπως ήδη έχουμε μάθει έως τώρα, κάτι στο οποίο (θα φαίνεται) δεν θα αναφερθούμε ξανά. Συνεπώς, ο πηγαίος κώδικας του mmap.c είναι ο κάτωθι:

mmap.c

```c
/*
 * mmap.c
 *
 * Examining the virtual memory of processes.
 *
 * Operating Systems course, CSLab, ECE, NTUA
 *
 */

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <stdint.h>
#include <signal.h>
#include <sys/wait.h>

#include "help.h"

#define RED     "\033[31m"
#define RESET   "\033[0m"


char *heap_private_buf;
char *heap_shared_buf;

char *file_shared_buf;

uint64_t buffer_size;

size_t file_size;


/*
 * Child process' entry point.
 */
void child(void)
{
    uint64_t pa;

    /*
     * Step 7 - Child
     */
    if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
    //TODO: Write your code here to complete child's part of Step 7.
    printf("The memory map of the child is:\n");
    show_maps();


    /*
     * Step 8 - Child
```

```
58      */
59     if (0 != raise(SIGSTOP))
60         die("raise(SIGSTOP)");
61     //TODO: Write your code here to complete child's part of Step 8.
62     printf("The physical address of private buff for child is: %ld\n",
   get_physical_address((uint64_t)heap_private_buf));
63
64
65     /*
66      * Step 9 - Child
67      */
68     if (0 != raise(SIGSTOP))
69         die("raise(SIGSTOP)");
70     //TODO: Write your code here to complete child's part of Step 9.
71     memset(heap_private_buf,1,buffer_size);
72     printf("The new physical address of private buff for child is: %ld\n",
   get_physical_address((uint64_t)heap_private_buf));
73
74     /*
75      * Step 10 - Child
76      */
77     if (0 != raise(SIGSTOP))
78         die("raise(SIGSTOP)");
79     //TODO: Write your code here to complete child's part of Step 10.
80     memset(heap_shared_buf,1,buffer_size);
81         printf("The new physical address of the child's shared memory is: %ld\n",
   get_physical_address((uint64_t)heap_shared_buf));
82
83
84     /*
85      * Step 11 - Child
86      */
87     if (0 != raise(SIGSTOP))
88         die("raise(SIGSTOP)");
89     //TODO: Write your code here to complete child's part of Step 11.
90     if (mprotect(heap_shared_buf, buffer_size,PROT_READ) == -1)
91                 perror("mprotect");
92     printf("The memory map of the child is:\n");
93         show_maps();
94         show_va_info((uint64_t)heap_shared_buf);
95
96     /*
97      * Step 12 - Child
98      */
99
100    //TODO: Write your code here to complete child's part of Step 12.
101    if(munmap(heap_shared_buf, buffer_size)==-1) perror("munmap");
102        if(munmap(heap_private_buf, buffer_size)==-1) perror("munmap");
103    if(munmap(file_shared_buf, file_size)==-1) perror("munmap");
104}
105
106/*
107 * Parent process' entry point.
108 */
109void parent(pid_t child_pid)
110{
111    uint64_t pa;
112    int status;
113
114    /* Wait for the child to raise its first SIGSTOP. */
115    if (-1 == waitpid(child_pid, &status, WUNTRACED))
```

```
116         die("waitpid");
117
118     /*
119      * Step 7: Print parent's and child's maps. What do you see?
120      * Step 7 - Parent
121      */
122     printf(RED "\nStep 7: Print parent's and child's map.\n" RESET);
123     press_enter();
124     //TODO: Write your code here to complete parent's part of Step 7.
125     printf("The memory map of the parent is:\n");
126     show_maps();
127
128     if (-1 == kill(child_pid, SIGCONT))
129         die("kill");
130     if (-1 == waitpid(child_pid, &status, WUNTRACED))
131         die("waitpid");
132
133
134     /*
135      * Step 8: Get the physical memory address for heap_private_buf.
136      * Step 8 - Parent
137      */
138     printf(RED "\nStep 8: Find the physical address of the private heap "
139         "buffer (main) for both the parent and the child.\n" RESET);
140     press_enter();
141     //TODO: Write your code here to complete parent's part of Step 8.
142     printf("The physical address of private buff for parent is: %ld\n",
    get_physical_address((uint64_t)heap_private_buf));
143
144     if (-1 == kill(child_pid, SIGCONT))
145         die("kill");
146     if (-1 == waitpid(child_pid, &status, WUNTRACED))
147         die("waitpid");
148
149
150     /*
151      * Step 9: Write to heap_private_buf. What happened?
152      * Step 9 - Parent
153      */
154     printf(RED "\nStep 9: Write to the private buffer from the child and "
155         "repeat step 8. What happened?\n" RESET);
156     press_enter();
157     //TODO: Write your code here to complete parent's part of Step 9.
158     printf("The new physical address of private buff for parent is: %ld\n",
    get_physical_address((uint64_t)heap_private_buf));
159
160     if (-1 == kill(child_pid, SIGCONT))
161         die("kill");
162     if (-1 == waitpid(child_pid, &status, WUNTRACED))
163         die("waitpid");
164
165
166     /*
167      * Step 10: Get the physical memory address for heap_shared_buf.
168      * Step 10 - Parent
169      */
170     printf(RED "\nStep 10: Write to the shared heap buffer (main) from "
171         "child and get the physical address for both the parent and "
172         "the child. What happened?\n" RESET);
173     press_enter();
```

```
174     //TODO: Write your code here to complete parent's part of Step 10.
175     printf("The new physical address of the parent's shared memory is: %ld\n",
    get_physical_address((uint64_t)heap_shared_buf));
176
177     if (-1 == kill(child_pid, SIGCONT))
178         die("kill");
179     if (-1 == waitpid(child_pid, &status, WUNTRACED))
180         die("waitpid");
181
182
183     /*
184      * Step 11: Disable writing on the shared buffer for the child
185      * (hint: mprotect(2)).
186      * Step 11 - Parent
187      */
188     printf(RED "\nStep 11: Disable writing on the shared buffer for the "
189         "child. Verify through the maps for the parent and the "
190         "child.\n" RESET);
191     press_enter();
192     //TODO: Write your code here to complete parent's part of Step 11.
193     printf("The memory map of the parent is:\n");
194         show_maps();
195         show_va_info((uint64_t)heap_shared_buf);
196
197     if (-1 == kill(child_pid, SIGCONT))
198         die("kill");
199     if (-1 == waitpid(child_pid, &status, 0))
200         die("waitpid");
201
202
203     /*
204      * Step 12: Free all buffers for parent and child.
205      * Step 12 - Parent
206      */
207     //TODO: Write your code here to complete parent's part of Step 12.
208         if(munmap(heap_shared_buf, buffer_size)==-1) perror("munmap");
209         if(munmap(heap_private_buf, buffer_size)==-1) perror("munmap");
210         if(munmap(file_shared_buf, file_size)==-1) perror("munmap");
211 }
212
213 int main(void)
214 {
215     pid_t mypid, p;
216     int fd = -1;
217     uint64_t pa;
218
219     mypid = getpid();
220     buffer_size = 1 * get_page_size();
221
222     /*
223      * Step 1: Print the virtual address space layout of this process.
224      */
225     printf(RED "\nStep 1: Print the virtual address space map of this "
226         "process [%d].\n" RESET, mypid);
227     press_enter();
228     // TODO: Write your code here to complete Step 1.
229     show_maps();
230
231     /*
232      * Step 2: Use mmap to allocate a buffer of 1 page and print the map
```

```
233      * again. Store buffer in heap_private_buf.
234      */
235     printf(RED "\nStep 2: Use mmap(2) to allocate a private buffer of "
236         "size equal to 1 page and print the VM map again.\n" RESET);
237     press_enter();
238     // TODO: Write your code here to complete Step 2.
239     heap_private_buf = mmap(NULL, buffer_size, PROT_READ|PROT_WRITE,
MAP_ANONYMOUS|MAP_PRIVATE, fd, 0);
240     if(heap_private_buf == MAP_FAILED) {perror("mmap"); return 1;}
241     show_maps();
242
243     /*
244      * Step 3: Find the physical address of the first page of your buffer
245      * in main memory. What do you see?
246      */
247     printf(RED "\nStep 3: Find and print the physical address of the "
248         "buffer in main memory. What do you see?\n" RESET);
249     press_enter();
250     // TODO: Write your code here to complete Step 3.
251     //get_physical_address((uint64_t)heap_private_buf);
252     printf("The physical address of buff is: %ld\n",get_physical_address((uint64_t)
heap_private_buf));
253
254     /*
255      * Step 4: Write zeros to the buffer and repeat Step 3.
256      */
257     printf(RED "\nStep 4: Initialize your buffer with zeros and repeat "
258         "Step 3. What happened?\n" RESET);
259     press_enter();
260     // TODO: Write your code here to complete Step 4.
261     memset(heap_private_buf,0,buffer_size);
262     printf("The physical address of buff is: %ld\n",get_physical_address((uint64_t)
heap_private_buf));
263
264
265     /*
266      * Step 5: Use mmap(2) to map file.txt (memory-mapped files) and print
267      * its content. Use file_shared_buf.
268      */
269     printf(RED "\nStep 5: Use mmap(2) to read and print file.txt. Print "
270         "the new mapping information that has been created.\n" RESET);
271     press_enter();
272     //TODO: Write your code here to complete Step 5.
273     fd=open("file.txt",O_RDONLY);
274     if(fd == -1) die("open");
275
276     struct stat st;
277         if (stat("file.txt", &st) == 0)
278             file_size = st.st_size;
279     else {perror("stat"); exit(1);}
280
281     file_shared_buf=mmap(NULL,file_size,PROT_READ,MAP_SHARED,fd,0);
282     if(file_shared_buf == MAP_FAILED) die("mmap");
283
284     write(0,file_shared_buf,file_size);
285
286     show_maps();
287     show_va_info((uint64_t)file_shared_buf);
288
289     /*
290      * Step 6: Use mmap(2) to allocate a shared buffer of 1 page. Use
```

```
     * heap_shared_buf.
     */
    printf(RED "\nStep 6: Use mmap(2) to allocate a shared buffer of size "
        "equal to 1 page. Initialize the buffer and print the new "
        "mapping information that has been created.\n" RESET);
    press_enter();
    //TODO: Write your code here to complete Step 6.
    heap_shared_buf = mmap(NULL, buffer_size, PROT_READ|PROT_WRITE,
MAP_ANONYMOUS|MAP_SHARED, -1, 0);
    if(heap_shared_buf==MAP_FAILED) die("mmap");
    memset(heap_shared_buf,0,buffer_size);
    printf("The physical address of the new shared memory is: %ld\n",
get_physical_address((uint64_t)heap_shared_buf));
    show_maps();
    show_va_info((uint64_t)heap_shared_buf);

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

    parent(p);

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

1. Στην παρακάτω εικόνα, βλέπουμε τον συμπληρωμένο κώδικα για το ζητούμενο ερώτημα:

```
press_enter();
// TODO: Write your code here to complete Step 1.
show_maps();
```

Έτσι, μέσω της κλήσης συστήματος show_maps(), λαμβάνουμε την ακόλουθη έξοδο, δηλαδή τον χάρτη εικονικής μνήμης της τρέχουσας διεργασίας:

```
Step 1: Print the virtual address space map of this process [801788].


Virtual Memory Map of process [801788]:
562ab6011000-562ab6012000 r--p 00000000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6012000-562ab6013000 r-xp 00001000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6013000-562ab6014000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6014000-562ab6015000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6015000-562ab6016000 rw-p 00003000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6052000-562ab6073000 rw-p 00000000 00:00 0                          [heap]
7f4da6897000-7f4da68b9000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da68b9000-7f4da6a12000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a12000-7f4da6a61000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a61000-7f4da6a65000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a65000-7f4da6a67000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a67000-7f4da6a6d000 rw-p 00000000 00:00 0
7f4da6a72000-7f4da6a73000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a73000-7f4da6a93000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a93000-7f4da6a9b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9c000-7f4da6a9d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9d000-7f4da6a9e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9e000-7f4da6a9f000 rw-p 00000000 00:00 0
7ffc666f0000-7ffc66711000 rw-p 00000000 00:00 0                          [stack]
7ffc667df000-7ffc667e3000 r--p 00000000 00:00 0                          [vvar]
7ffc667e3000-7ffc667e5000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------
```

2. Στην παρακάτω εικόνα, βλέπουμε τον συμπληρωμένο κώδικα για το ζητούμενο ερώτημα:

```
// TODO: Write your code here to complete Step 2.
heap_private_buf = mmap(NULL, buffer_size, PROT_READ|PROT_WRITE, MAP_ANONYMOUS|MAP_PRIVATE, fd, 0);
if(heap_private_buf == MAP_FAILED) {perror("mmap"); return 1;}
show_maps();
```

Έτσι, μέσω της κλήσης συστήματος mmap() με τις κατάλληλες παραμέτρους, λαμβάνουμε – με επιπλέον χρήση της εντολής show_maps() – τον νέο χάρτη εικονικών διευθύνσεων μνήμης, όπως φαίνεται ακολούθως:

```
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 [801788]:
562ab6011000-562ab6012000 r--p 00000000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6012000-562ab6013000 r-xp 00001000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6013000-562ab6014000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6014000-562ab6015000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6015000-562ab6016000 rw-p 00003000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6052000-562ab6073000 rw-p 00000000 00:00 0                          [heap]
7f4da6897000-7f4da68b9000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da68b9000-7f4da6a12000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a12000-7f4da6a61000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a61000-7f4da6a65000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a65000-7f4da6a67000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a67000-7f4da6a6d000 rw-p 00000000 00:00 0
7f4da6a72000-7f4da6a73000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a73000-7f4da6a93000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a93000-7f4da6a9b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9b000-7f4da6a9c000 rw-p 00000000 00:00 0
7f4da6a9c000-7f4da6a9d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9d000-7f4da6a9e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9e000-7f4da6a9f000 rw-p 00000000 00:00 0
7ffc666f0000-7ffc66711000 rw-p 00000000 00:00 0                          [stack]
7ffc667df000-7ffc667e3000 r--p 00000000 00:00 0                          [vvar]
7ffc667e3000-7ffc667e5000 r-xp 00000000 00:00 0                          [vdso]
------------------------------------------------------
```

3. Κατόπιν, βρίσκουμε και τυπώνουμε τη φυσική μνήμη που δεσμεύσαμε προηγουμένως, μέσω της εντολής get_physical_address(), όπως φαίνεται ακολούθως στον κώδικα:

```
// TODO: Write your code here to complete Step 3.
//get_physical_address((uint64_t)heap_private_buf);
printf("The physical address of buff is: %ld\n",get_physical_address((uint64_t)heap_private_buf));
```

και λαμβάνουμε το ακόλουθο μήνυμα:

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

VA[0x7f4da6a9b000] is not mapped; no physical memory allocated.
The physical address of buff is: 0
```

Παρατηρούμε ότι το ΛΣ δεν δέσμευσε φυσική μνήμη από το μηχάνημά μας που να αντιστοιχεί στην παραπάνω εικονική! Η εξήγηση για αυτό είναι ότι μπορεί μεν να εκφράσαμε την ανάγκη μας (δυναμικά) για παραχώρηση επιπλέον μνήμης, ωστόσο αυτή δεν τη χρειαστήκαμε ποτέ και το ΛΣ μας την παραχωρεί μόνο κατά τον χρόνο που τη χρησιμοποιούμε, on demand.

4. Έπειτα, χρησιμοποιούμε την εντολή memset() προκειμένου να γεμίσουμε με μηδενικά τον buffer μας, όπως φαίνεται ακολούθως:

```
// TODO: Write your code here to complete Step 4.
memset(heap_private_buf,0,buffer_size);
printf("The physical address of buff is: %ld\n",get_physical_address((uint64_t)heap_private_buf));
```

και, ομοίως με τη διαδικασία που ακολουθήσαμε προηγουμένως, λαμβάνουμε την ακόλουθη έξοδο:

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

The physical address of buff is: 9147281408
```

Παρατηρούμε ότι τώρα – που πλέον χρειαστήκαμε και αξιοποιήσαμε ενεργά τη μνήμη που ζητήσαμε από το ΛΣ – αυτό μας την παραχώρησε και έτσι η εικονική διεύθυνση πλέον αντιστοιχεί και σε φυσική: αυτή που εμφανίζεται παραπάνω.

5. Έπειτα, όπως φαίνεται και στο ακόλουθο κομμάτι κώδικα, ανοίγουμε το αρχείο file.txt και βρίσκουμε το μέγεθός του και δεσμεύουμε αντίστοιχο μέγεθος μνήμης, μέσω της mmap():

```
//TODO: Write your code here to complete Step 5.
fd=open("file.txt",O_RDONLY);
if(fd == -1) die("open");

struct stat st;
size_t file_size;
if (stat("file.txt", &st) == 0)
        file_size = st.st_size;
else {perror("stat"); exit(1);}

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

write(0,file_shared_buf,file_size);
close(fd);

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

Παρατηρούμε ότι στην έξοδό μας βρίσκεται τόσο το περιεχόμενο του αρχείου (“Hello everyone”) όσο και ο νέος πίνακα εικονικής μνήμης της διεργασίας, όπως και η θέση του αρχείου file.txt σε αυτή. Για να την απομονώσουμε, χρησιμοποιήσαμε την εντολή show_va_info() της μνήμης που δεσμεύσαμε:

```
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 [801788]:
562ab6011000-562ab6012000 r--p 00000000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6012000-562ab6013000 r-xp 00001000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6013000-562ab6014000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6014000-562ab6015000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6015000-562ab6016000 rw-p 00003000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6052000-562ab6073000 rw-p 00000000 00:00 0                          [heap]
7f4da6897000-7f4da68b9000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da68b9000-7f4da6a12000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a12000-7f4da6a61000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a61000-7f4da6a65000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a65000-7f4da6a67000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a67000-7f4da6a6d000 rw-p 00000000 00:00 0
7f4da6a71000-7f4da6a72000 r--s 00000000 00:26 2246818                    /home/oslab/oslab005/askhsh3/file.txt
7f4da6a72000-7f4da6a73000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a73000-7f4da6a93000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a93000-7f4da6a9b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9b000-7f4da6a9c000 rw-p 00000000 00:00 0
7f4da6a9c000-7f4da6a9d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9d000-7f4da6a9e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9e000-7f4da6a9f000 rw-p 00000000 00:00 0
7ffc666f0000-7ffc66711000 rw-p 00000000 00:00 0                          [stack]
7ffc667df000-7ffc667e3000 r--p 00000000 00:00 0                          [vvar]
7ffc667e3000-7ffc667e5000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------------

7f4da6a71000-7f4da6a72000 r--s 00000000 00:26 2246818                    /home/oslab/oslab005/askhsh3/file.txt
```

6. Τέλος, πριν εκτελέσουμε τη fork(), συντάσσουμε τον ακόλουθο κώδικα, στον οποίο χρησιμοποιούμε τη memset() για να αρχικοποιήσουμε την εικονική μνήμη που δεσμεύσαμε μέσω της mmap(), ως εξής:

```
//TODO: Write your code here to complete Step 6.
fd=-1;
heap_shared_buf = mmap(NULL, buffer_size, PROT_READ|PROT_WRITE, MAP_ANONYMOUS|MAP_SHARED, fd, 0);
if(heap_shared_buf==MAP_FAILED) die("mmap");
memset(heap_shared_buf,0,buffer_size);
printf("The physical address of the new shared memory is: %ld\n",get_physical_address((uint64_t)heap_shared_buf));
show_maps();
show_va_info((uint64_t)heap_shared_buf);
```

και κατόπιν απεικονίζουμε τις νέες πληροφορίες, όπως και στα προηγούμενα ερωτήματα:

```
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.

The physical address of the new shared memory is: 7833841664

Virtual Memory Map of process [801788]:
562ab6011000-562ab6012000 r--p 00000000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6012000-562ab6013000 r-xp 00001000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6013000-562ab6014000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6014000-562ab6015000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6015000-562ab6016000 rw-p 00003000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6052000-562ab6073000 rw-p 00000000 00:00 0                          [heap]
7f4da6897000-7f4da68b9000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da68b9000-7f4da6a12000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a12000-7f4da6a61000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a61000-7f4da6a65000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a65000-7f4da6a67000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a67000-7f4da6a6d000 rw-p 00000000 00:00 0
7f4da6a70000-7f4da6a71000 rw-s 00000000 00:01 554                        /dev/zero (deleted)
7f4da6a71000-7f4da6a72000 r--s 00000000 00:26 2246818                    /home/oslab/oslab005/askhsh3/file.txt
7f4da6a72000-7f4da6a73000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a73000-7f4da6a93000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a93000-7f4da6a9b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9b000-7f4da6a9c000 rw-p 00000000 00:00 0
7f4da6a9c000-7f4da6a9d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9d000-7f4da6a9e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9e000-7f4da6a9f000 rw-p 00000000 00:00 0
7ffc666f0000-7ffc66711000 rw-p 00000000 00:00 0                          [stack]
7ffc667df000-7ffc667e3000 r--p 00000000 00:00 0                          [vvar]
7ffc667e3000-7ffc667e5000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------

7f4da6a70000-7f4da6a71000 rw-s 00000000 00:01 554                        /dev/zero (deleted)
```

Στο σημείο αυτό, εκτελούμε fork() και δημιουργούμε μια νέα διεργασία.

7. Αρχικά, συμπληρώνουμε τον κώδικα του ερωτήματος για τον πατέρα και το παιδί, ώστε να τυπώνουν τον χάρτη μνήμης ως εξής:

```
//TODO: Write your code here to complete child's part of Step 7.
printf("The memory map of the child is:\n");
show_maps();
```

```
//TODO: Write your code here to complete parent's part of Step 7.
printf("The memory map of the parent is:\n");
show_maps();
```

και το αποτέλεσμα φαίνεται ακολούθως:

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

The memory map of the parent is:

Virtual Memory Map of process [801788]:
562ab6011000-562ab6012000 r--p 00000000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6012000-562ab6013000 r-xp 00001000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6013000-562ab6014000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6014000-562ab6015000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6015000-562ab6016000 rw-p 00003000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6052000-562ab6073000 rw-p 00000000 00:00 0                          [heap]
7f4da6897000-7f4da68b9000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da68b9000-7f4da6a12000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a12000-7f4da6a61000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a61000-7f4da6a65000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a65000-7f4da6a67000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a67000-7f4da6a6d000 rw-p 00000000 00:00 0
7f4da6a70000-7f4da6a71000 rw-s 00000000 00:01 554                        /dev/zero (deleted)
7f4da6a71000-7f4da6a72000 r--s 00000000 00:26 2246818                    /home/oslab/oslab005/askhsh3/file.txt
7f4da6a72000-7f4da6a73000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a73000-7f4da6a93000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a93000-7f4da6a9b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9b000-7f4da6a9c000 rw-p 00000000 00:00 0
7f4da6a9c000-7f4da6a9d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9d000-7f4da6a9e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9e000-7f4da6a9f000 rw-p 00000000 00:00 0
7ffc666f0000-7ffc66711000 rw-p 00000000 00:00 0                          [stack]
7ffc667df000-7ffc667e3000 r--p 00000000 00:00 0                          [vvar]
7ffc667e3000-7ffc667e5000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------

The memory map of the child is:

Virtual Memory Map of process [801789]:
562ab6011000-562ab6012000 r--p 00000000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6012000-562ab6013000 r-xp 00001000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6013000-562ab6014000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6014000-562ab6015000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6015000-562ab6016000 rw-p 00003000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6052000-562ab6073000 rw-p 00000000 00:00 0                          [heap]
7f4da6897000-7f4da68b9000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da68b9000-7f4da6a12000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a12000-7f4da6a61000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a61000-7f4da6a65000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a65000-7f4da6a67000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a67000-7f4da6a6d000 rw-p 00000000 00:00 0
7f4da6a70000-7f4da6a71000 rw-s 00000000 00:01 554                        /dev/zero (deleted)
7f4da6a71000-7f4da6a72000 r--s 00000000 00:26 2246818                    /home/oslab/oslab005/askhsh3/file.txt
7f4da6a72000-7f4da6a73000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a73000-7f4da6a93000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a93000-7f4da6a9b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9b000-7f4da6a9c000 rw-p 00000000 00:00 0
7f4da6a9c000-7f4da6a9d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9d000-7f4da6a9e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9e000-7f4da6a9f000 rw-p 00000000 00:00 0
7ffc666f0000-7ffc66711000 rw-p 00000000 00:00 0                          [stack]
7ffc667df000-7ffc667e3000 r--p 00000000 00:00 0                          [vvar]
7ffc667e3000-7ffc667e5000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------
```

Παρατηρούμε ότι ο χάρτης μνήμης που τυπώθηκε είναι ο ίδιος, τόσο για τη διεργασία – γονέα όσο και για τη διεργασία – παιδί, κάτι το οποίο είναι αναμενόμενο, καθώς η fork() δημιουργεί ένα αντίγραφο του γονέα στο παιδί, τη στιγμή που αυτό γεννιέται, με όλα τα στοιχεία του. Το δικαίωμα εγγραφής αφαιρείται τόσο από τον γονέα όσο και από το παιδί, αφού έχουμε Copy-On-Write (COW).

8. Ομοίως, συμπληρώνουμε τον ζητούμενο κώδικα για τις δύο διεργασίες, ώστε να τυπώνει τη φυσική διεύθυνση της εικονικής που δεσμεύσαμε, ως εξής:

```
//TODO: Write your code here to complete child's part of Step 8.
 printf("The physical address of private buff for child is: %ld\n",get_physical_address((uint64_t)heap_private_buf));
```

```
//TODO: Write your code here to complete parent's part of Step 8.
printf("The physical address of private buff for parent is: %ld\n",get_physical_address((uint64_t)heap_private_buf));
```

και το αποτέλεσμα φαίνεται ακολούθως:

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

 The physical address of private buff for parent is: 9147281408
 The physical address of private buff for child is: 9147281408
```

Παρατηρούμε ότι η διεύθυνση φυσικής μνήμης που τυπώθηκε είναι η ίδια, τόσο για τη διεργασία – γονέα όσο και για τη διεργασία – παιδί, κάτι το οποίο είναι αναμενόμενο, καθώς η fork() δημιουργεί ένα αντίγραφο του γονέα στο παιδί, τη στιγμή που αυτό γεννιέται, με όλα τα στοιχεία του.

9. Τώρα, γράφουμε στον private buffer από τη διεργασία παιδί και επαναλαμβάνουμε τη διαδικασία, όπως στο προηγούμενο ερώτημα:

```
//TODO: Write your code here to complete parent's part of Step 9.
printf("The new physical address of private buff for parent is: %ld\n",get_physical_address((uint64_t)heap_private_buf));
```

```
 //TODO: Write your code here to complete child's part of Step 9.
 memset(heap_private_buf,1,buffer_size);
 printf("The new physical address of private buff for child is: %ld\n",get_physical_address((uint64_t)heap_private_buf));
```

και λαμβάνουμε την ακόλουθη έξοδο:

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

 The new physical address of private buff for parent is: 9147281408
 The new physical address of private buff for child is: 5582807040
```

Παρατηρούμε ότι η διεύθυνση φυσικής μνήμης, στην οποία αντιστοιχεί ο heap_private_buf, έχει παραμείνει η ίδια για τη διεργασία – γονέα, ενώ έχει αλλάξει για τη διεργασία παιδί. Αυτό ήταν αναμενόμενο, αφού το παιδί πλέον ζήτησε να γράψει ενεργά στη μνήμη και έτσι το αντίγραφό του έγινε ξεχωριστό από αυτό του πατέρα του, του αποδόθηκε δικαίωμα εγγραφής από το ΛΣ, καθώς και μία φυσική θέση μνήμης, προκειμένου να αντιγραφεί το αρχικό περιεχόμενο και να γίνουν οι απαραίτητες αλλαγές. Δηλαδή, αρχικά το περιεχόμενο του buf δεν είχε αντιγραφεί για τη διεργασία παιδί από το ΛΣ όταν έγινε το fork() (σε φυσική μνήμη, αφού δεν χρειαζόταν να το εγγράψει και παρέμενε σε μια εικονική μνήμη, ενώ τώρα αντιγράφηκε σε φυσική μνήμη, με δικαίωμα εγγραφής. Αυτό συνέβη επειδή η mmap() κλήθηκε με MAP_PRIVATE και έχουμε COW.

10. Τώρα, γράφουμε στον shared buffer από τη διεργασία παιδί και επαναλαμβάνουμε τη διαδικασία, όπως στο προηγούμενο ερώτημα:

```
//TODO: Write your code here to complete parent's part of Step 10.
printf("The new physical address of the parent's shared memory is: %ld\n",get_physical_address((uint64_t)heap_shared_buf));
```

```
//TODO: Write your code here to complete child's part of Step 10.
memset(heap_shared_buf,1,buffer_size);
printf("The new physical address of the child's shared memory is: %ld\n",get_physical_address((uint64_t)heap_shared_buf));
```

και λαμβάνουμε την ακόλουθη έξοδο:

```
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?

The new physical address of the parent's shared memory is: 7833841664
The new physical address of the child's shared memory is: 7833841664
```

Παρατηρούμε ότι η διεύθυνση φυσικής μνήμης, στην οποία αντιστοιχεί ο heap_shared_buf, έχει παραμείνει η ίδια για τη διεργασία – γονέα και για τη διεργασία παιδί, ενώ είναι ίδια και με την αρχική! Αυτό ήταν αναμενόμενο, αφού η mmap() κλήθηκε με MAP_SHARED και έχουμε COW, επιτρέποντας τη διεργασιακή επικοινωνία. Δηλαδή, αρχικά το περιεχόμενο του buf δεν είχε αντιγραφεί για τη διεργασία παιδί από το ΛΣ όταν έγινε το fork() (σε φυσική μνήμη), αφού δεν χρειαζόταν να το εγγράψει και παρέμενε σε μια εικονική μνήμη, ενώ τώρα αντιγράφηκε σε φυσική μνήμη, με δικαίωμα εγγραφής.

11. Έπειτα, απαγορεύουμε τις εγγραφές στον shared buffer για τη διεργασία – παιδί, κάνοντας χρήση της εντολής mprotect(), ως εξής:

```
//TODO: Write your code here to complete parent's part of Step 11.
printf("The memory map of the parent is:\n");
show_maps();
show_va_info((uint64_t)heap_shared_buf);
```

```
//TODO: Write your code here to complete child's part of Step 11.
if (mprotect(heap_shared_buf, buffer_size,PROT_READ) == -1)
        perror("mprotect");
printf("The memory map of the child is:\n");
show_maps();
show_va_info((uint64_t)heap_shared_buf);
```

και λαμβάνουμε την ακόλουθη έξοδο:

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

The memory map of the parent is:

Virtual Memory Map of process [801788]:
562ab6011000-562ab6012000 r--p 00000000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6012000-562ab6013000 r-xp 00001000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6013000-562ab6014000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6014000-562ab6015000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6015000-562ab6016000 rw-p 00003000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6052000-562ab6073000 rw-p 00000000 00:00 0                          [heap]
7f4da6897000-7f4da68b9000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da68b9000-7f4da6a12000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a12000-7f4da6a61000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a61000-7f4da6a65000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a65000-7f4da6a67000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a67000-7f4da6a6d000 rw-p 00000000 00:00 0
7f4da6a70000-7f4da6a71000 rw-s 00000000 00:01 554                        /dev/zero (deleted)
7f4da6a71000-7f4da6a72000 r--s 00000000 00:26 2246818                    /home/oslab/oslab005/askhsh3/file.txt
7f4da6a72000-7f4da6a73000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a73000-7f4da6a93000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a93000-7f4da6a9b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9b000-7f4da6a9c000 rw-p 00000000 00:00 0
7f4da6a9c000-7f4da6a9d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9d000-7f4da6a9e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9e000-7f4da6a9f000 rw-p 00000000 00:00 0
7ffc666f0000-7ffc66711000 rw-p 00000000 00:00 0                          [stack]
7ffc667df000-7ffc667e3000 r--p 00000000 00:00 0                          [vvar]
7ffc667e3000-7ffc667e5000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------

7f4da6a70000-7f4da6a71000 rw-s 00000000 00:01 554                        /dev/zero (deleted)
The memory map of the child is:

Virtual Memory Map of process [801789]:
562ab6011000-562ab6012000 r--p 00000000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6012000-562ab6013000 r-xp 00001000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6013000-562ab6014000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6014000-562ab6015000 r--p 00002000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6015000-562ab6016000 rw-p 00003000 00:26 2246822                    /home/oslab/oslab005/askhsh3/mmap
562ab6052000-562ab6073000 rw-p 00000000 00:00 0                          [heap]
7f4da6897000-7f4da68b9000 r--p 00000000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da68b9000-7f4da6a12000 r-xp 00022000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a12000-7f4da6a61000 r--p 0017b000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a61000-7f4da6a65000 r--p 001c9000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a65000-7f4da6a67000 rw-p 001cd000 fe:01 144567                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f4da6a67000-7f4da6a6d000 rw-p 00000000 00:00 0
7f4da6a70000-7f4da6a71000 r--s 00000000 00:01 554                        /dev/zero (deleted)
7f4da6a71000-7f4da6a72000 r--s 00000000 00:26 2246818                    /home/oslab/oslab005/askhsh3/file.txt
7f4da6a72000-7f4da6a73000 r--p 00000000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a73000-7f4da6a93000 r-xp 00001000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a93000-7f4da6a9b000 r--p 00021000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9b000-7f4da6a9c000 rw-p 00000000 00:00 0
7f4da6a9c000-7f4da6a9d000 r--p 00029000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9d000-7f4da6a9e000 rw-p 0002a000 fe:01 144563                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f4da6a9e000-7f4da6a9f000 rw-p 00000000 00:00 0
7ffc666f0000-7ffc66711000 rw-p 00000000 00:00 0                          [stack]
7ffc667df000-7ffc667e3000 r--p 00000000 00:00 0                          [vvar]
7ffc667e3000-7ffc667e5000 r-xp 00000000 00:00 0                          [vdso]
-----------------------------------------------------

7f4da6a70000-7f4da6a71000 r--s 00000000 00:01 554                        /dev/zero (deleted)
```

Παρατηρούμε ότι η απαγόρευση εγγραφής έχει ολοκληρωθεί για το παιδί και πως αν αυτό πάει να γράψει θα έχουμε segmentation fault.

12. Τέλος, αποδεσμεύουμε όλους τους buffers, κάνοντας χρήση της εντολής munmap(), ως εξής:

```
//TODO: Write your code here to complete parent's part of Step 12.
if(munmap(heap_shared_buf, buffer_size)==-1) perror("munmap");
if(munmap(heap_private_buf, buffer_size)==-1) perror("munmap");
if(munmap(file_shared_buf, file_size)==-1) perror("munmap");
```

```
//TODO: Write your code here to complete child's part of Step 12.
if(munmap(heap_shared_buf, buffer_size)==-1) perror("munmap");
if(munmap(heap_private_buf, buffer_size)==-1) perror("munmap");
if(munmap(file_shared_buf, file_size)==-1) perror("munmap");
```

- **Ενότητα 2 – Παράλληλος υπολογισμός Mandelbrot με διεργασίες αντί για νήματα**

2.1 – Semaphores πάνω από διαμοιραζόμενη μνήμη

Αρχικά, παραθέτουμε τον συμπληρωμένο κώδικα της άσκησης – του αρχείου mandel-fork1.c – ακολούθως, στον οποίο έχουμε υλοποιήσει το mandel με σημαφόρους και διεργασίες, αντί για σημαφόρους και threads. Αξίζει να επισημάνουμε ότι κάθε κώδικας που συμπληρώνουμε στο αρχείο φροντίζει να ελέγχει και να προλαμβάνει πιθανά λάθη, όπως ήδη έχουμε μάθει έως τώρα, κάτι στο οποίο (θα φαίνεται) δεν θα αναφερθούμε ξανά. Συνεπώς, ο πηγαίος κώδικας του mandel-fork1.c είναι ο κάτωθι:

mandel-fork1.c

```c
/*
 * mandel.c
 *
 * A program to draw the Mandelbrot Set on a 256-color xterm.
 *
 */

#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#include <semaphore.h>
#include <sys/mman.h>
#include <sys/wait.h>

/*TODO header file for m(un)map*/

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

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

/*
 * Output at the terminal is is x_chars wide by y_chars long
*/
int y_chars = 50;
int x_chars = 90;

/*
 * The part of the complex plane to be drawn:
 * upper left corner is (xmin, ymax), lower right corner is (xmax, ymin)
*/
double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;

/*
 * Every character in the final output is
 * xstep x ystep units wide on the complex plane.
 */
double xstep;
double ystep;

sem_t *sem;

int safe_atoi(char *s, int *val)
{
        long l;
        char *endp;

        l = strtol(s, &endp, 10);
        if (s != endp && *endp == '\0') {
```

```c
                *val = l;
                return 0;
        } else
                return -1;
}

void usage(char *argv0)
{
        fprintf(stderr, "Usage: %s procedure_count\n"
                "Exactly one argument required:\n"
                "    procedure_count: The number of procedures to create.\n",
                argv0);
        exit(1);
}

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

    int n;
    int val;

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

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

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

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

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

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

    for (i = 0; i < x_chars; i++) {
        /* Set the current color, then output the point */
        set_xterm_color(fd, color_val[i]);
```

```
118         if (write(fd, &point, 1) != 1) {
119             perror("compute_and_output_mandel_line: write point");
120             exit(1);
121         }
122     }
123
124     /* Now that the line is done, output a newline character */
125     if (write(fd, &newline, 1) != 1) {
126         perror("compute_and_output_mandel_line: write newline");
127         exit(1);
128     }
129 }
130
131 void compute_and_output_mandel_line(int fd, int init_line, int procs)
132 {
133     /*
134      * A temporary array, used to hold color values for the line being drawn
135      */
136     int color_val[x_chars];
137
138     for(int line = init_line; line < y_chars; line += procs) {
139         compute_mandel_line(line, color_val);
140         if(sem_wait(&sem[line % procs]) < 0) {perror("sem_wait"); exit(1);}
141             output_mandel_line(fd, color_val);
142         if(sem_post(&sem[(line+1) % procs]) < 0) {perror("sem_post"); exit(1);}
143     }
144 }
145
146 /*
147  * Create a shared memory area, usable by all descendants of the calling
148  * process.
149  */
150 void *create_shared_memory_area(unsigned int numbytes)
151 {
152     int pages;
153     void *addr;
154
155     if (numbytes == 0) {
156         fprintf(stderr, "%s: internal error: called for numbytes == 0\n", __func__);
157         exit(1);
158     }
159
160     /*
161      * Determine the number of pages needed, round up the requested number of
162      * pages
163      */
164     pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;
165
166     /* Create a shared, anonymous mapping for this number of pages */
167     /* TODO:*/
168     addr = mmap(NULL, pages * sysconf(_SC_PAGE_SIZE), PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);
169     if(addr == MAP_FAILED) {perror("mmap"); exit(1);}
170
171     return addr;
172 }
173
174 void destroy_shared_memory_area(void *addr, unsigned int numbytes) {
175     int pages;
176
```

```
177     if (numbytes == 0) {
178         fprintf(stderr, "%s: internal error: called for numbytes == 0\n", __func__);
179         exit(1);
180     }
181 
182     /*
183      * Determine the number of pages needed, round up the requested number of
184      * pages
185      */
186     pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;
187 
188     if (munmap(addr, pages * sysconf(_SC_PAGE_SIZE)) == -1) {
189         perror("destroy_shared_memory_area: munmap failed");
190         exit(1);
191     }
192 }
193 
194 int main(int argc, char *argv[])
195 {
196     pid_t p;
197     int procs;
198 
199     /*
200          * Parse the command line
201          */
202         if (argc != 2)
203                 usage(argv[0]);
204         if (safe_atoi(argv[1], &procs) < 0 || procs <= 0) {
205                 fprintf(stderr, "`%s' is not valid for `procedures_count'\n", argv[1]);
206                 exit(1);
207         }
208 
209     xstep = (xmax - xmin) / x_chars;
210     ystep = (ymax - ymin) / y_chars;
211 
212     sem = create_shared_memory_area(procs * sizeof(sem_t));
213     if(sem_init(&sem[0], 1, 1) < 0) {perror("sem_init"); exit(1);}
214     for(int i = 1; i < procs; i++)
215         if(sem_init(&sem[i], 1, 0) < 0) {perror("sem_init"); exit(1);}
216 
217 
218     /*
219      * draw the Mandelbrot Set, one line at a time.
220      * Output is sent to file descriptor '1', i.e., standard output.
221      */
222     for (int init_line = 0; init_line < procs; init_line++) {
223         p = fork();
224         if(p < 0) {perror("fork"); exit(1);}
225         if(p == 0) {
226                     compute_and_output_mandel_line(1, init_line, procs);
227                     exit(0);
228                 }
229     }
230 
231     //Finish program
232     for(int i=0; i<procs; i++)
233         wait(NULL);
234     destroy_shared_memory_area(sem, procs*sizeof(sem_t));
235 
236     reset_xterm_color(1);
```

```
237     return 0;
238 }
239
```

Ερωτήσεις:

1. Ανάμεσα στις υλοποιήσεις με threads και με forks, αναμένουμε καλύτερη επίδοση να έχουμε στην περίπτωση των threads. Αυτό συμβαίνει διότι κατά τη διαδικασία του fork αντιγράφεται το περιεχόμενο της γονικής διεργασίας, μια διαδικασία πολύ χρονοβόρα, αφού πρέπει να σωθεί η κατάσταση της διεργασίας από το ΛΣ στο Process Control Block. Ακόμα, η επικοινωνία μέσω διαμοιραζόμενης μνήμης είναι από μόνη της αργή, καθώς αρχικά αφαιρείται το δικαίωμα εγγραφής από τα παιδιά (έχουμε Copy-On-Write) και από τον πατέρα για τη μνήμη που αυτός έχει δεσμεύσει, ενώ όταν αυτά ζητήσουν να γράψουν τους παραχωρείται εκ νέου. Τέλος, το ίδιο το ΛΣ δημιουργεί και καταστρέφει threads πιο γρήγορα από ό,τι δημιουργεί και καταστρέφει διεργασίες, αφού τα threads έχουν εξαρχής κοινή μνήμη (τις global μεταβλητές) και έτσι δεν χρειάζεται να δεσμεύσουμε εμείς κοινή μνήμη δυναμικά, μέσω της mmap() και της create_shared_memory_area(). Ο παραπάνω ισχυρισμός επιβεβαιώθηκε και με κατάλληλη χρήση της εντολής time για τη χρονομέτρηση του προγράμματός μας, όπου η υλοποίηση με forks ήταν πολύ πιο αργή. Το αποτέλεσμα (χρόνος) με forks (ας γίνει σύγκριση με την προηγούμενη αναφορά) φαίνεται παρακάτω.

Εκτέλεση με 2 διεργασίες:

```
real    0m0.312s
user    0m0.543s
sys     0m0.059s
```

Εκτέλεση με 6 διεργασίες:

```
real    0m0.146s
user    0m0.693s
sys     0m0.018s
```

2. Δυστυχώς, το mmap interface δεν μπορεί να χρησιμοποιηθεί για τον διαμοιρασμό μνήμης μεταξύ διεργασιών που δεν έχουν κοινό ancestor. Αυτό συμβαίνει διότι σε κάθε fork() όλος ο χώρος του πατέρα – επομένως μαζί και ο χώρος εικονικών διευθύνσεων – αντιγράφεται στο παιδί. Συνεπώς, αν ο πατέρας κάνει mmap (shared), τότε η εικονική μνήμη που θα δεσμεύσει θα είναι προσβάσιμη και από το παιδί του, καθώς και από το παιδί του παιδιού του (με την ίδια λογική, αφού το αντίγραφό του θα περιέχει και το αντίγραφο του πατέρα του πατέρα του) και άρα και από οποιονδήποτε απόγονό του, αφού η κατάσταση του αρχικού πατέρα (που περιέχει την κοινή mmap) θα κληροδοτείται σε κάθε επίπεδο. Αυτό, όμως, δεν μπορεί να συμβεί αν δεν έχουμε κοινό πρόγονο, αφού τότε θα κληροδοτηθεί το αντίστοιχο αντίγραφο, το οποίο δεν θα περιέχει την κοινή mmap και άρα ο χώρος αυτός των εικονικών διευθύνσεων δεν θα μπορεί να αξιοποιηθεί.

➢ 2.2 – Υλοποίηση χωρίς semaphores

Αρχικά, παραθέτουμε τον συμπληρωμένο κώδικα της άσκησης – του αρχείου mandel-fork2.c – ακολούθως, στον οποίο έχουμε υλοποιήσει το mandel μόνο με διεργασίες, αντί για σημαφόρους και διεργασίες όπως πριν. Αξίζει να επισημάνουμε ότι κάθε κώδικας που συμπληρώνουμε στο αρχείο φροντίζει να ελέγχει και να προλαμβάνει πιθανά λάθη, όπως ήδη έχουμε μάθει έως τώρα, κάτι στο οποίο (θα φαίνεται) δεν θα αναφερθούμε ξανά. Συνεπώς, ο πηγαίος κώδικας του mandel-fork2.c είναι ο κάτωθι:

mandel-fork2.c

```c
/*
 * mandel.c
 *
 * A program to draw the Mandelbrot Set on a 256-color xterm.
 *
 */

#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#include <sys/mman.h>
#include <sys/wait.h>

/*TODO header file for m(un)map*/

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

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

/*
 * Output at the terminal is is x_chars wide by y_chars long
*/
int y_chars = 50;
int x_chars = 90;

/*
 * The part of the complex plane to be drawn:
 * upper left corner is (xmin, ymax), lower right corner is (xmax, ymin)
*/
double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;

/*
 * Every character in the final output is
 * xstep x ystep units wide on the complex plane.
 */
double xstep;
double ystep;

int **color_val;

int safe_atoi(char *s, int *val)
{
        long l;
        char *endp;

        l = strtol(s, &endp, 10);
        if (s != endp && *endp == '\0') {
                *val = l;
```

```
58                  return 0;
59          } else
60                  return -1;
61  }
62
63  void usage(char *argv0)
64  {
65          fprintf(stderr, "Usage: %s procedure_count\n"
66                  "Exactly one argument required:\n"
67                  "    procedure_count: The number of procedures to create.\n",
68                  argv0);
69          exit(1);
70  }
71
72  /*
73   * This function computes a line of output
74   * as an array of x_char color values.
75   */
76  void compute_mandel_line(int line)
77  {
78      /*
79       * x and y traverse the complex plane.
80       */
81      double x, y;
82
83      int n;
84      int val;
85
86      /* Find out the y value corresponding to this line */
87      y = ymax - ystep * line;
88
89      /* and iterate for all points on this line */
90      for (x = xmin, n = 0; n < x_chars; x+= xstep, n++) {
91
92          /* Compute the point's color value */
93          val = mandel_iterations_at_point(x, y, MANDEL_MAX_ITERATION);
94          if (val > 255)
95              val = 255;
96
97          /* And store it in the color_val[] array */
98          val = xterm_color(val);
99          color_val[n][line] = val;
100     }
101 }
102
103 /*
104  * This function outputs an array of x_char color values
105  * to a 256-color xterm.
106  */
107 void output_mandel_line(int fd, int line)
108 {
109     int i;
110
111     char point ='@';
112     char newline='\n';
113
114     for (i = 0; i < x_chars; i++) {
115         /* Set the current color, then output the point */
116         set_xterm_color(fd, color_val[i][line]);
117         if (write(fd, &point, 1) != 1) {
```

```
118             perror("compute_and_output_mandel_line: write point");
119             exit(1);
120         }
121     }
122 
123     /* Now that the line is done, output a newline character */
124     if (write(fd, &newline, 1) != 1) {
125         perror("compute_and_output_mandel_line: write newline");
126         exit(1);
127     }
128 }
129 
130 /*
131  * Create a shared memory area, usable by all descendants of the calling
132  * process.
133  */
134 void *create_shared_memory_area(unsigned int numbytes)
135 {
136     int pages;
137     void *addr;
138 
139     if (numbytes == 0) {
140         fprintf(stderr, "%s: internal error: called for numbytes == 0\n", __func__);
141         exit(1);
142     }
143 
144     /*
145      * Determine the number of pages needed, round up the requested number of
146      * pages
147      */
148     pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;
149 
150     /* Create a shared, anonymous mapping for this number of pages */
151     /* TODO:*/
152     addr = mmap(NULL, pages * sysconf(_SC_PAGE_SIZE), PROT_READ | PROT_WRITE, MAP_SHARED |
    MAP_ANONYMOUS, -1, 0);
153     if(addr == MAP_FAILED) {perror("mmap"); exit(1);}
154 
155     return addr;
156 }
157 
158 void destroy_shared_memory_area(void *addr, unsigned int numbytes) {
159     int pages;
160 
161     if (numbytes == 0) {
162         fprintf(stderr, "%s: internal error: called for numbytes == 0\n", __func__);
163         exit(1);
164     }
165 
166     /*
167      * Determine the number of pages needed, round up the requested number of
168      * pages
169      */
170     pages = (numbytes - 1) / sysconf(_SC_PAGE_SIZE) + 1;
171 
172     if (munmap(addr, pages * sysconf(_SC_PAGE_SIZE)) == -1) {
173         perror("destroy_shared_memory_area: munmap failed");
174         exit(1);
175     }
176 }
```

```c

int main(int argc, char *argv[])
{
    pid_t p;
    int procs;

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

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

    // Initialize 2-D array
    color_val = create_shared_memory_area(x_chars * sizeof(int));

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


    /*
     * draw the Mandelbrot Set, one line at a time.
     * Output is sent to file descriptor '1', i.e., standard output.
     */
    for (int init_line = 0; init_line < procs; init_line++) {
        p = fork();
        if(p < 0) {perror("fork"); exit(1);}
        if(p == 0) {
            for(int line = init_line; line < y_chars; line += procs)
                        compute_mandel_line(line);
                    exit(0);
        }
    }

    // Parent waits
    for(int i=0; i<procs; i++)
        wait(NULL);

    // Print output
    for(int line = 0; line < y_chars; line++)
        output_mandel_line(1, line);

    // Destroy memory
    for(int i = 0; i < y_chars; i++)
        destroy_shared_memory_area(color_val, y_chars * sizeof(int));
    destroy_shared_memory_area(color_val, x_chars * sizeof(int));

    reset_xterm_color(1);
    return 0;
}

```

Ερωτήσεις

1. Στη συγκεκριμένη υλοποίηση, οι διεργασίες που γεννιούνται και κάνουν το compute της κάθε γραμμής δεν έχουν την ανάγκη για συγχρονισμό, ωστόσο ο γενικότερος συγχρονισμός επιτυγχάνεται μεταξύ όλων των παιδιών και του πατέρα. Αυτό συμβαίνει, διότι αρχικά δεσμεύουμε μία μεγάλη περιοχή μνήμης, μεγέθους (x_chars*y_chars*sizeof(int)), ωστόσο ποτέ καμία διεργασία – παιδί δεν θα πάει στην ίδια «υποπεριοχή» μνήμης της αρχικής μνήμης – μεγέθους x_chars*sizeof(int), που χρειάζεται για κάθε γραμμή. Ειδικότερα, κάθε διεργασία – παιδί έχει τη δική της, αποκλειστική περιοχή – τμήμα μνήμης της αρχικής (διαμοιραζόμενης) που δεσμεύσαμε, με το παραπάνω μέγεθος, στην οποία ποτέ δεν γίνεται πρόσβαση από άλλη διεργασία και όπου αποθηκεύει το compute μιας γραμμής. Έτσι, δεν πρόκειται να «μπλεχτούν» ποτέ τα δεδομένα των computes των γραμμών του mandel και αυτά μπορούν να γίνουν ταυτόχρονα από όλες τις διεργασίες. Όπως και παλιά, το κρίσιμο τμήμα παραμένει αυτό του output της κάθε γραμμής, ωστόσο αυτό το εκτελεί στο τέλος – συνολικά ο πατέρας. Έτσι, ουσιαστικά δεν έχουμε κρίσιμο τμήμα πλέον και ο συγχρονισμός έγκειται στο γεγονός ότι το παλιό κρίσιμο τμήμα έχει απομονωθεί και έχει αποδοθεί πλήρως στον πατέρα, μετά τον τερματισμό όλων των παιδιών του.

   Σε περίπτωση που είχαμε έναν μικρότερο buffer – διαστάσεων nprocs*x_chars – τότε θα ακολουθούσαμε την εξής λογική: καθεμία από τις n διεργασίες θα υπολόγιζε (κατά σειρά) μία από τις n πρώτες γραμμές του buffer και όταν τελείωνε, θα ενημέρωνε τη διεργασία – πατέρα. Όταν όλες τερμάτιζαν (άρα ο buffer έχει γεμίσει), τότε ο πατέρας θα τύπωνε το τμηματικό αποτέλεσμα των n πρώτων γραμμών και έπειτα θα τις ειδοποιούσε, ώστε να προβούν στον υπολογισμό των επόμενων n γραμμών, με την ίδια λογική. Η διαδικασία αυτή θα επαναλαμβανόταν, μέχρι να τελειώσουν όλες οι γραμμές του Mandelbrot, οπότε και το συνολικό πρόγραμμα θα τερματίσει.