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;
}
```