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
     */
    if (0 != raise(SIGSTOP))
        die("raise(SIGSTOP)");
    //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));


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

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


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

    /*
     * Step 12 - Child
     */

    //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");
}

/*
 * Parent process' entry point.
 */
void parent(pid_t child_pid)
{
    uint64_t pa;
    int status;

    /* Wait for the child to raise its first SIGSTOP. */
    if (-1 == waitpid(child_pid, &status, WUNTRACED))
```

```c
        die("waitpid");

    /*
     * Step 7: Print parent's and child's maps. What do you see?
     * Step 7 - Parent
     */
    printf(RED "\nStep 7: Print parent's and child's map.\n" RESET);
    press_enter();
    //TODO: Write your code here to complete parent's part of Step 7.
    printf("The memory map of the parent is:\n");
    show_maps();

    if (-1 == kill(child_pid, SIGCONT))
        die("kill");
    if (-1 == waitpid(child_pid, &status, WUNTRACED))
        die("waitpid");


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

    if (-1 == kill(child_pid, SIGCONT))
        die("kill");
    if (-1 == waitpid(child_pid, &status, WUNTRACED))
        die("waitpid");


    /*
     * Step 9: Write to heap_private_buf. What happened?
     * Step 9 - Parent
     */
    printf(RED "\nStep 9: Write to the private buffer from the child and "
        "repeat step 8. What happened?\n" RESET);
    press_enter();
    //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));

    if (-1 == kill(child_pid, SIGCONT))
        die("kill");
    if (-1 == waitpid(child_pid, &status, WUNTRACED))
        die("waitpid");


    /*
     * Step 10: Get the physical memory address for heap_shared_buf.
     * Step 10 - Parent
     */
    printf(RED "\nStep 10: Write to the shared heap buffer (main) from "
        "child and get the physical address for both the parent and "
        "the child. What happened?\n" RESET);
    press_enter();
    //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));

    if (-1 == kill(child_pid, SIGCONT))
        die("kill");
    if (-1 == waitpid(child_pid, &status, WUNTRACED))
        die("waitpid");


    /*
     * Step 11: Disable writing on the shared buffer for the child
     * (hint: mprotect(2)).
     * Step 11 - Parent
     */
    printf(RED "\nStep 11: Disable writing on the shared buffer for the "
        "child. Verify through the maps for the parent and the "
        "child.\n" RESET);
    press_enter();
    //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);

    if (-1 == kill(child_pid, SIGCONT))
        die("kill");
    if (-1 == waitpid(child_pid, &status, 0))
        die("waitpid");


    /*
     * Step 12: Free all buffers for parent and child.
     * Step 12 - Parent
     */
    //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");
}

int main(void)
{
    pid_t mypid, p;
    int fd = -1;
    uint64_t pa;

    mypid = getpid();
    buffer_size = 1 * get_page_size();

    /*
     * Step 1: Print the virtual address space layout of this process.
     */
    printf(RED "\nStep 1: Print the virtual address space map of this "
        "process [%d].\n" RESET, mypid);
    press_enter();
    // TODO: Write your code here to complete Step 1.
    show_maps();

    /*
     * Step 2: Use mmap to allocate a buffer of 1 page and print the map
     * again. Store buffer in heap_private_buf.
     */
    printf(RED "\nStep 2: Use mmap(2) to allocate a private buffer of "
        "size equal to 1 page and print the VM map again.\n" RESET);
    press_enter();
    // 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();

    /*
     * Step 3: Find the physical address of the first page of your buffer
     * in main memory. What do you see?
     */
    printf(RED "\nStep 3: Find and print the physical address of the "
        "buffer in main memory. What do you see?\n" RESET);
    press_enter();
    // 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));

    /*
     * Step 4: Write zeros to the buffer and repeat Step 3.
     */
    printf(RED "\nStep 4: Initialize your buffer with zeros and repeat "
        "Step 3. What happened?\n" RESET);
    press_enter();
    // 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));


    /*
     * Step 5: Use mmap(2) to map file.txt (memory-mapped files) and print
     * its content. Use file_shared_buf.
     */
    printf(RED "\nStep 5: Use mmap(2) to read and print file.txt. Print "
        "the new mapping information that has been created.\n" RESET);
    press_enter();
    //TODO: Write your code here to complete Step 5.
    fd=open("file.txt",O_RDONLY);
    if(fd == -1) die("open");

    struct stat st;
        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);

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

    /*
     * 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;
}


```