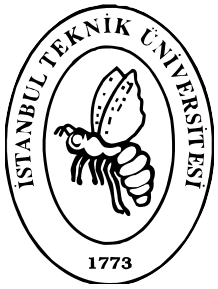
ISTANBUL TECHNICAL UNIVERSITY



BLG 413E

SYSTEM PROGRAMMING

Ayşe Şima UYAR

Project 1

Group Members

Name	Surname	Number
Emre	UYSAL	150160510
Cihat	BOSTANCI	150160542
Tugay	ACAR	150160511

INTRODUCTION

In this project, it is expected to modify kernel functions according to given constraints. For modifying the functions, it is required to implement a task descriptor named as “myFlag”. As a group, we worked on the given prototype system call and according to practice session we implemented all requirements and started out project.

FIRST STEP

In the first step, we downloaded the Linux-Source-3.13.0.tar.bz2 and extract it to make changes on it. After extracting it we created our directory “/set_myFlag/” using “mkdir” command in it. In this directory we need to create our system call. So we implemented system call function with using the given template. We returned appropriate errors using <linux/errno.h> .

```
#include <linux/syscalls.h>
#include <linux/kernel.h>
#include <linux/errno.h>

asmlinkage long sys_set_myFlag(pid_t pid, int flag)
{
    if(capable(CAP_SYS_ADMIN)) //needs root  privalage
    {
        struct task_struct *process;
        process =  find_task_by_vpid(pid); //find given pid
        if(process == NULL)
        {
            return ESRCH; //no such process
        }
        else
        {
            if(flag == 1 || flag == 0)
            {
                process->myFlag = flag; //set flag value into task_struct
                return 0; //return success after setting the flag
            }
            else
            {
                return EINVAL; //invalid argument error
            }
        }
    }
    else
    {
        return EPERM;//operation not permitted error
    }
}
```

After creating the system call we created Makefile. In the Makefile we wrote ;

```
obj-y := set_myFlag.o
```

After that we changed our directory to linux-source-3.13.0 folder and modified the Makefile in it.

```
# Objects we will link into vmlinux / subdirs we need to visit
init-y          := init/
drivers-y       := drivers/ sound/ firmware/ ubuntu/
net-y           := net/
libs-y          := lib/
core-y          := usr/ set_myFlag/
```

SECOND STEP

In the second step, we created our variable in task_struct for being capable to use it. The directory is include/linux/sched.h

```
1469    int myFlag;
```

After that we put it into init_task in the include/linux/init_task.h ;

```
218     .myFlag = 0,                                    \
```

THIRD STEP

In the third step, we put our prototype system call into include/linux/syscalls.h ;

```
852  asmlinkage long sys_set_myFlag(pid_t pid, int flag);
```

After that we put the required information into arch/x86/syscalls/syscall_32.tbl for a proper compilation.

```
355     i386    set_myFlag              sys_set_myFlag
```

FOURTH STEP

In this step, we modified the fork system call. When we call the fork() from user space do_exit() performs in the kernel space. So we modified the do_exit() function. We used current macro for detecting the current process for forking. After making the controls in an if statement we returned ECHILD error if the flag is 1.

```
1651    struct task_struct *controller = current;
1652    if(controller->myFlag == 1 && task_nice(controller)>10)
1653    {
1654        return ECHILD; //returns error no child processes
1655    }
```

After that an unexpected thing occurred in our fork. So we solved it using the copy process function which creates child processes. In the copy process function we gave child flags 0 for creating an initialized flags in child processes.

```
1290    p->myFlag = 0;  //set myFlag 0 to give childs a default flag parameter
```

FIFTH STEP

In this step, we modified the exit.c in /kernel/ directory. We controlled our constraints with an if block and after that we looked for current process's parent's children list for detecting the siblings. After that we used list_entry for making the initialization for task_struct. With the sys_kill we killed our siblings. All these steps until here in exit included if the flag is one and nice value is greater than ten. If the current process is not suitable to this constraints it makes a normal exit.

```
709     if(current->myFlag == 1 && task_nice(current)>10)
710     {
711         struct list_head *sibs_of_me;
712         struct task_struct *sib;
713         list_for_each(sibs_of_me, &current->parent->children)
714         {
715             sib = list_entry(sibs_of_me,struct task_struct, sibling);
716             sys_kill(sib->pid,SIGKILL);
717         }
718     }
```

LAST STEP BEFORE COMPILE

We learned that, it is required to modify "arch/x86/boot/compressed/head_32.S" for a successful compiling.

```
#ifdef CONFIG_EFI_STUB
        .data
efi32_config:
        .fill 11,8,0
        .long efi_call_phys
        .long 0
        .byte 0
#endif
```

After that we need to modify the grub file in our local linux in etc/default/ directory.

```
GRUB_DEFAULT=0
#GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""
```

TESTING

We used the given tested methods below. Tests are also given below.

Error Checking

1- Compile : gcc test3.c -o testError
2- Eperm Testing : ./testError <niceVal> <flag>
3- Esrch Testing: sudo ./testError <niceVal> <flag>
3.1 - Follow the guides on the test code please
4- Einval Testing : sudo ./testError <niceVal> <flag>

4.1 - Flag must be given not 1 or 0

Fork Checking

1-Compile: gcc test3.c -o forkTesting

2-Normally fork : sudo ./forkTesting <11> <0>

3-No child process created (echild) : sudo ./forkTesting <11> <1>

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include    <errno.h>
#include <string.h>

#define set_myFlag 355
int main(int argc, char** argv)
{
    if(argc<2 || argc>3)
    {
        printf("Give exact two arguments first niceValue Second Flag\n");
        return 0;
    }

    int  niceValue=atoi(argv[1]);
    int flag=atoi(argv[2]);

    //try pid manuelly for getting ESRCH error uncomment these lines
    //int pid;
    //printf("RANDOM PID: ");
    //scanf("%d",&pid);
    //int ERR=syscall(set_myFlag,pid,flag);
    //after that comment the other error part

    int niceRet =nice(niceValue);
    int ERR=syscall(set_myFlag,getpid(),flag);
    if(ERR!=0){
        printf("Error Control statement:%s\n",strerror(ERR));
        return 0;
    }
    int f=fork();

    if(f==0)
    {
        printf("CHild pid:%d ---- Parent pid:%d  niceValue:%d\n",getpid(),getppid(),niceRet);

    }
    else
    {
        printf("Error Control statement:%s \n",strerror(ERR));
        printf("Fork return %s created\n",strerror(f));

    }
    sleep(1);
    return 0;
```

Exit Checking

1-Compile: gcc exit_test2.c -o exitTest

2-Compile: gcc flag_setter.c -o controlExit

3-Run: ./exitTest

4-Normally Exit: sudo ./controlExit <0> <childpid>

5-Kill Siblings : sudo ./controlExit <1> <childpid>

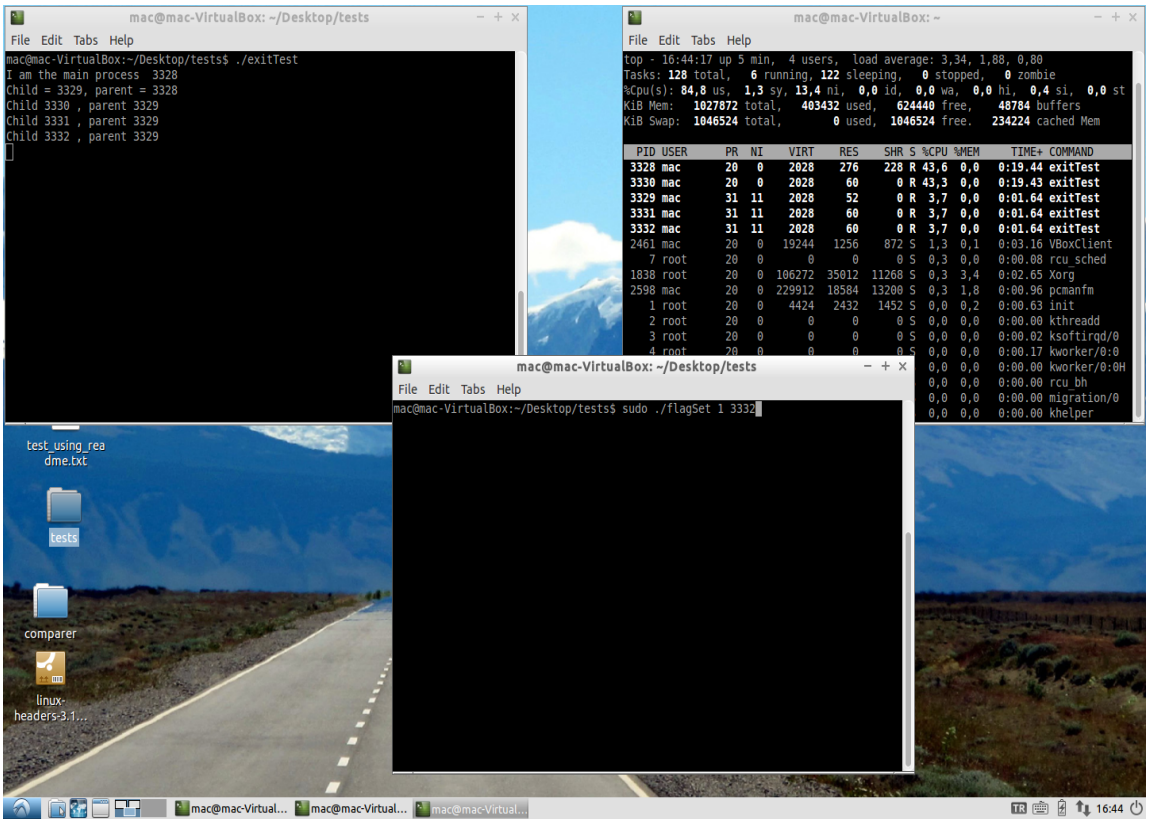
```c
int main(int argc, char ** argv)
{
    int first = fork();
    if(first == 0)
    {
        printf("Child = %d, parent = %d\n", getpid(),getppid());
        int second = fork();
        if(second == 0)
        {
            printf("Child %d , parent %d\n", getpid(),getppid());
            while(1);
        }
        else
        {
            int retval = nice(11);
            int third = fork();
            if(third == 0)
            {
                printf("Child %d , parent %d\n", getpid(),getppid());
                while(1);
            }
            else
            {
                int fifth = fork();
                if(fifth == 0)
                {
                    printf("Child %d , parent %d\n", getpid(),getppid());
                    while(1);
                }
                while(1);
            }
            while(1);
        }
    }
    else
    {
        printf("I am the main process  %d\n",getpid() );
        while(1);
    }
    sleep(1);
    return 0;
}
```

exit_test2.c

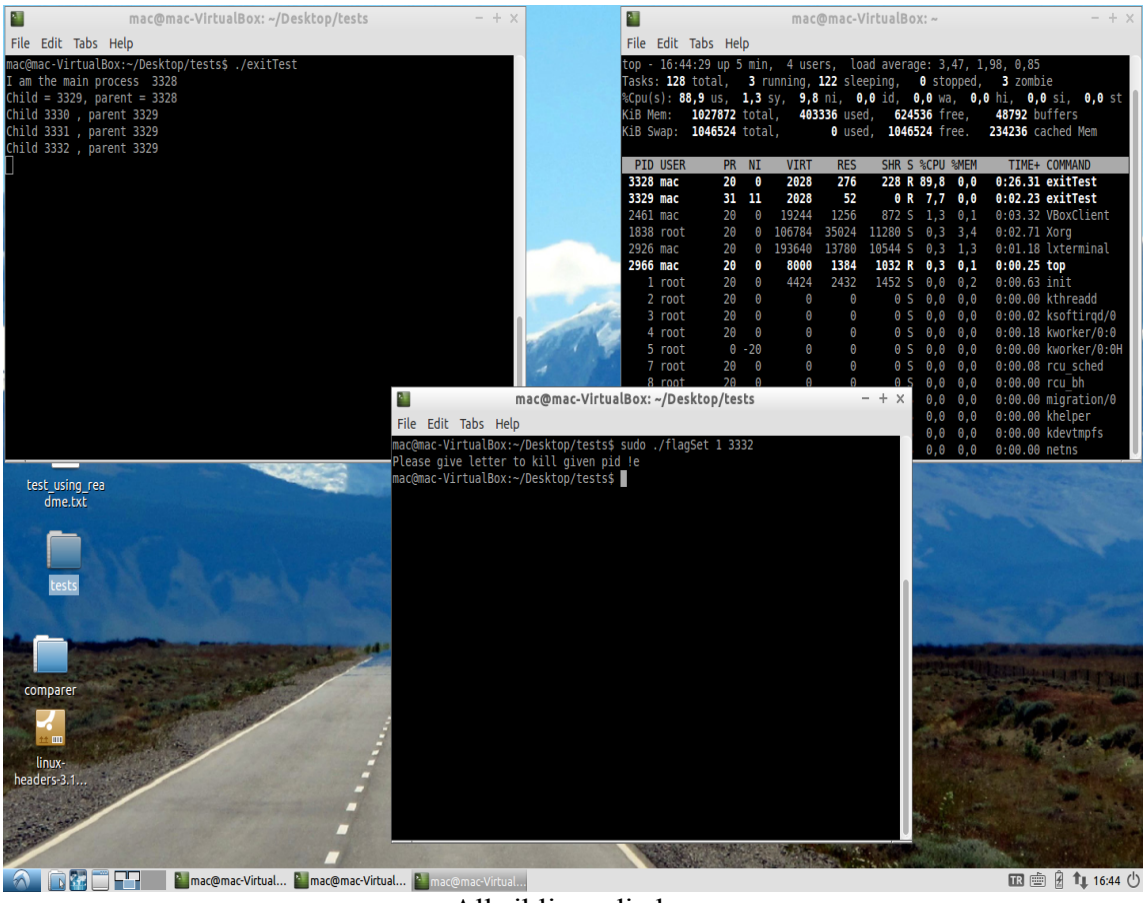
```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>



#define set_flag 355

int main(int argc, char **argv)
{


    int flag = atoi(argv[1]);
    int pid = atoi(argv[2]);
    int setter = syscall(set_flag, pid,flag);
    if(setter !=0)
    {
            printf("error code : %s ",strerror(setter));
    }

    printf("Please give letter to kill given pid !");
    getchar();
    kill(pid,9);
    return 0;

}
```

flag_setter.c

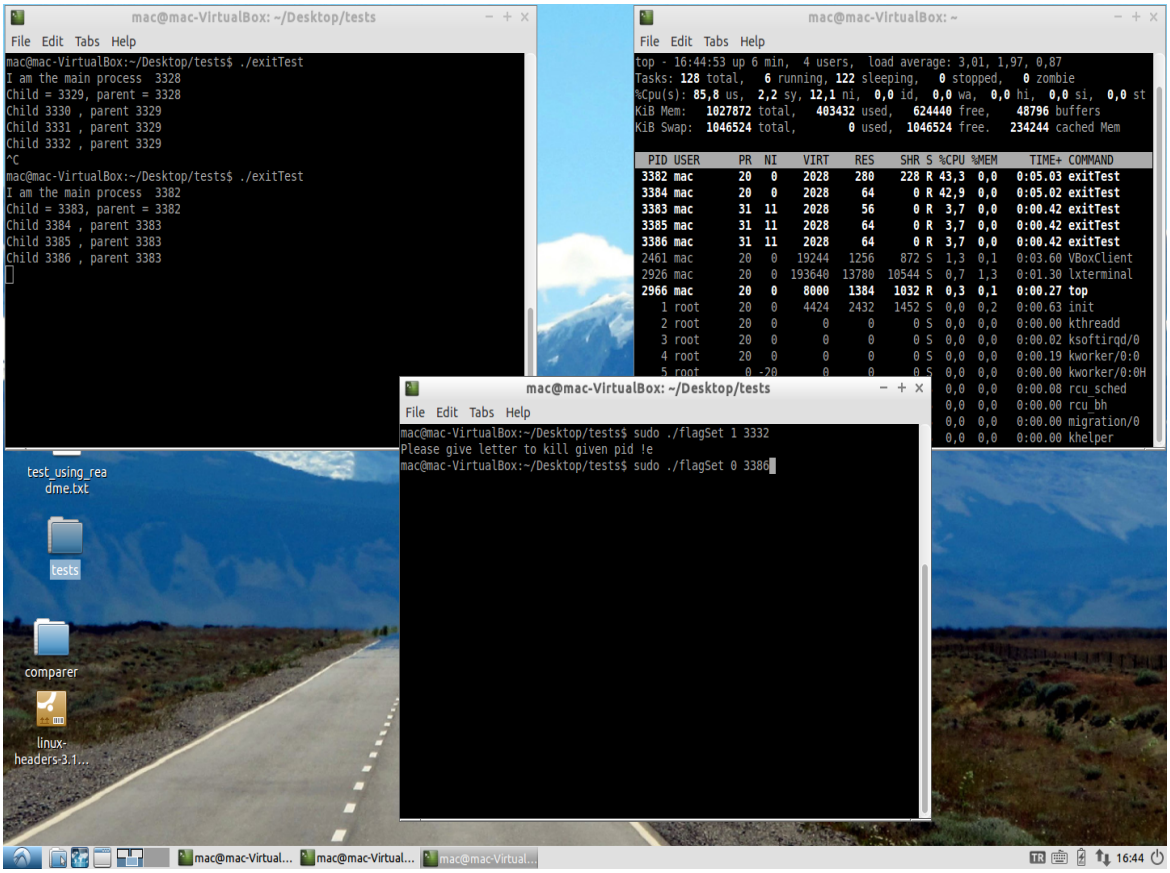
TEST RESULTS AS SCREENSHOTS



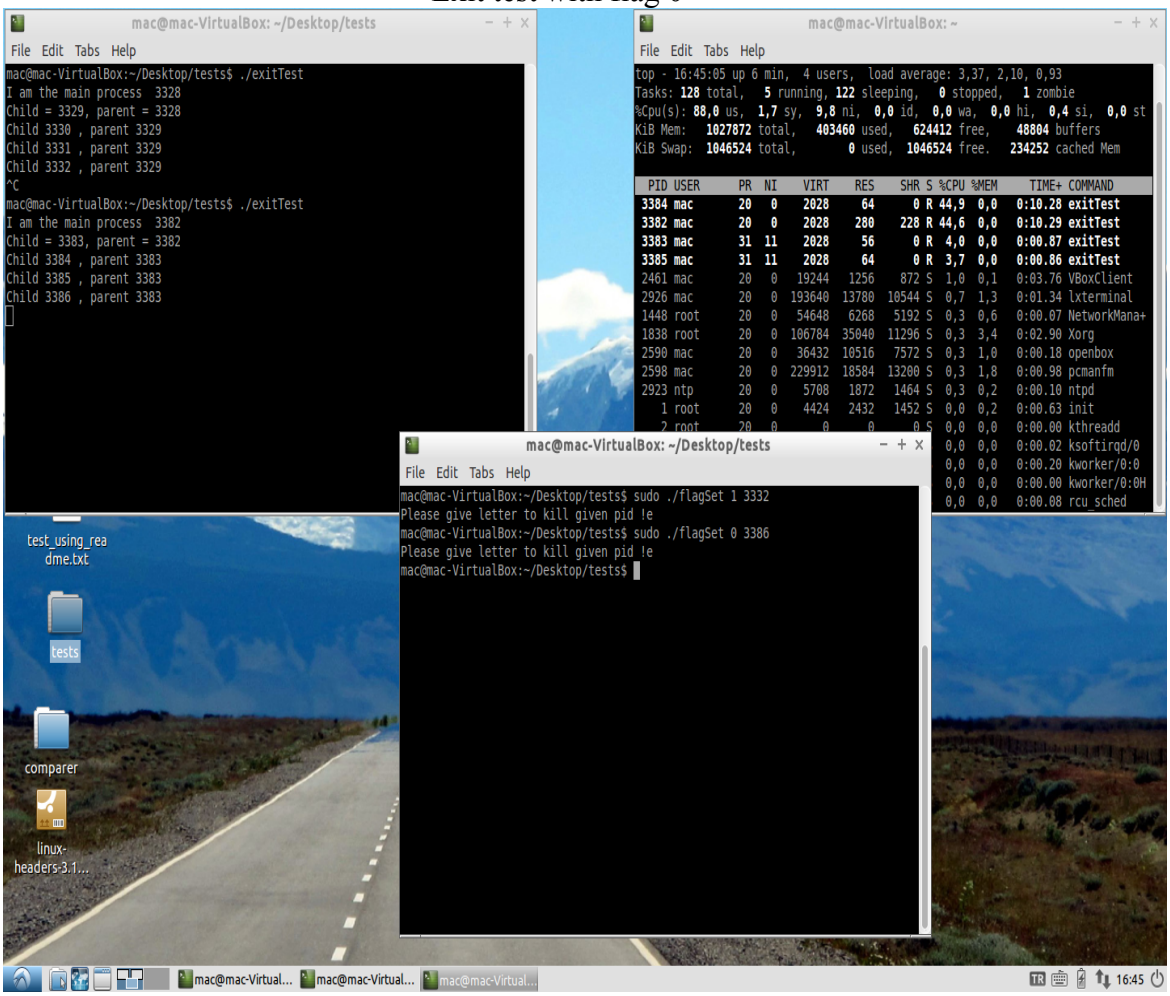
Exit test with flag 1



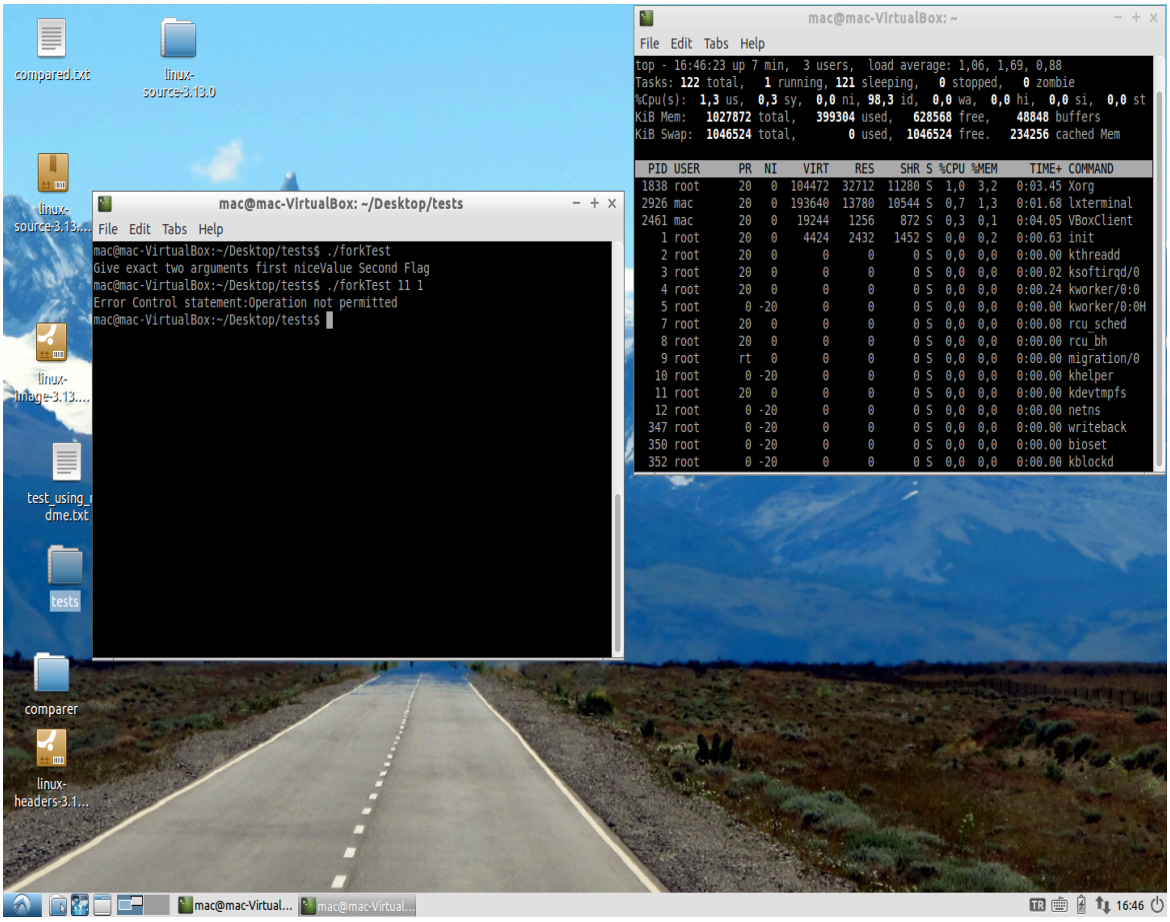
All siblings died



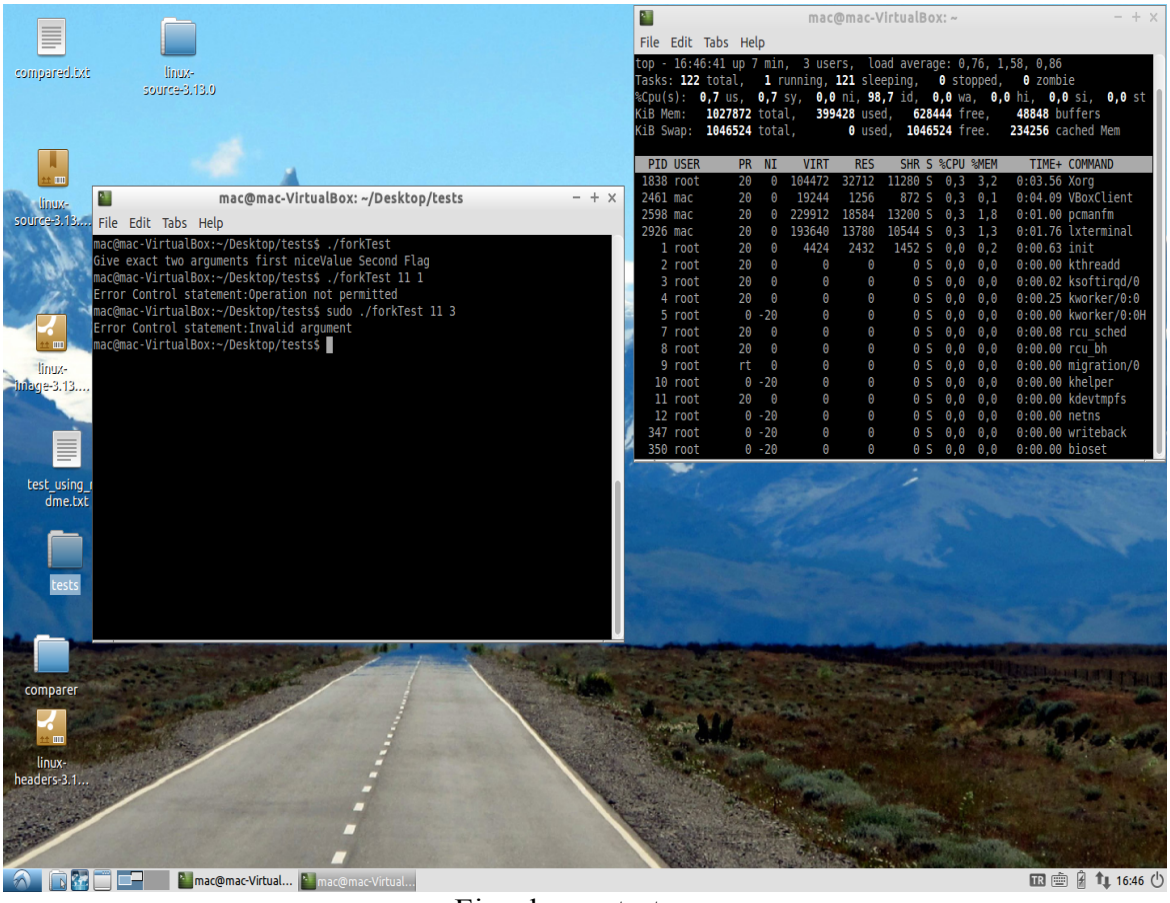
Exit test with flag 0



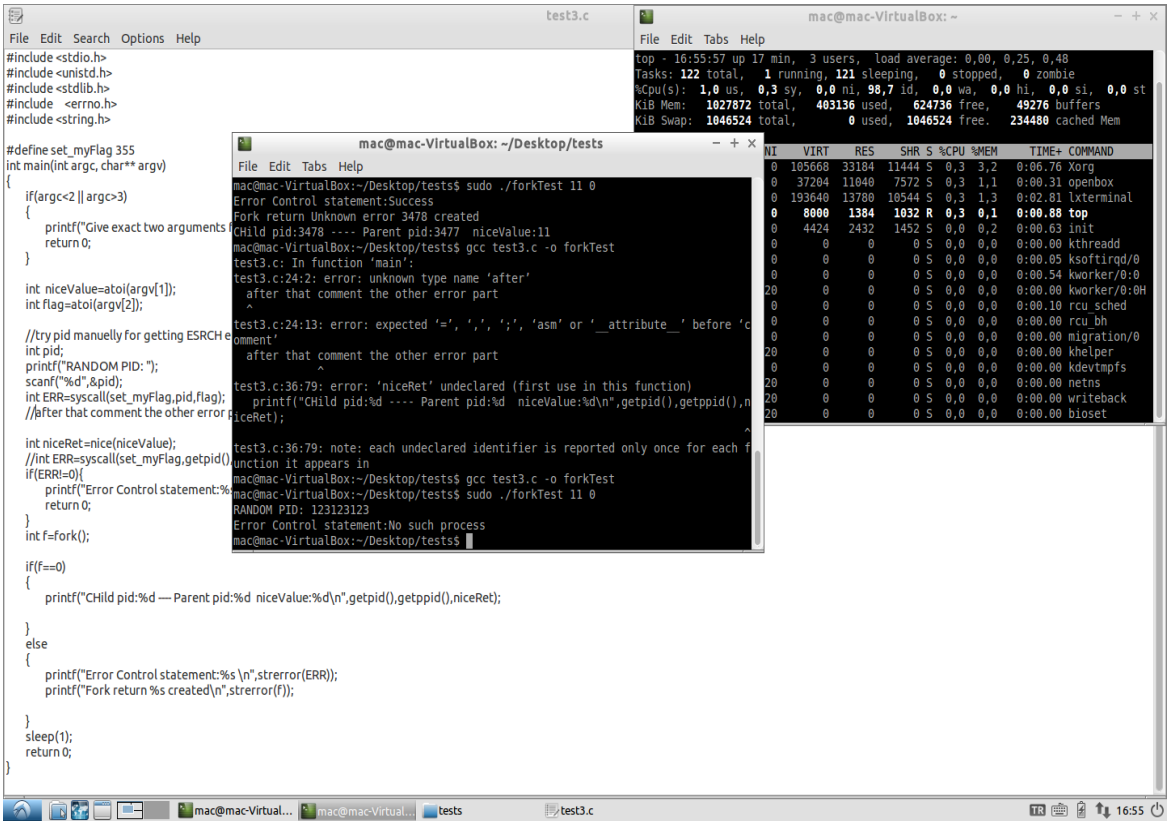
Just the given pid died



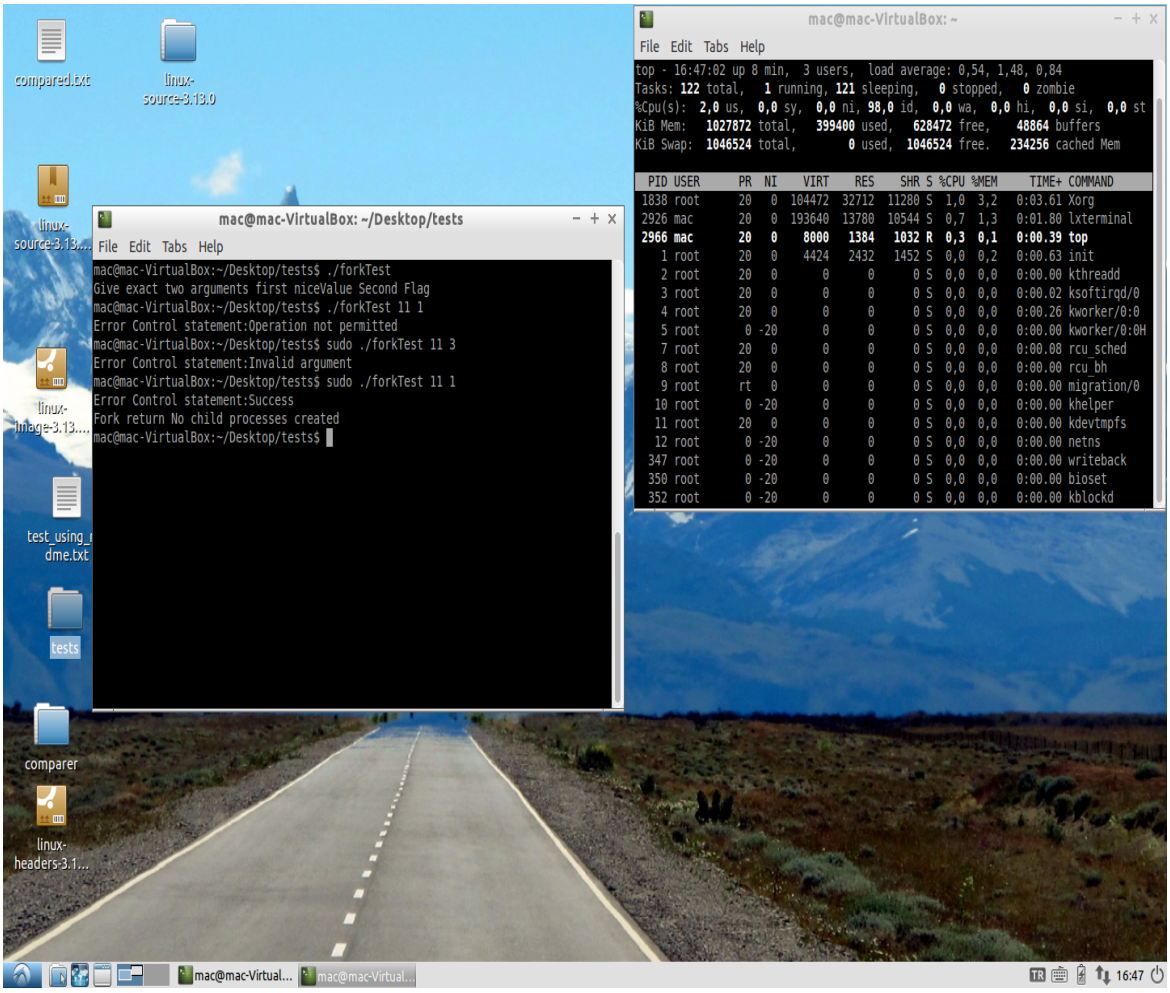
Eperm error test



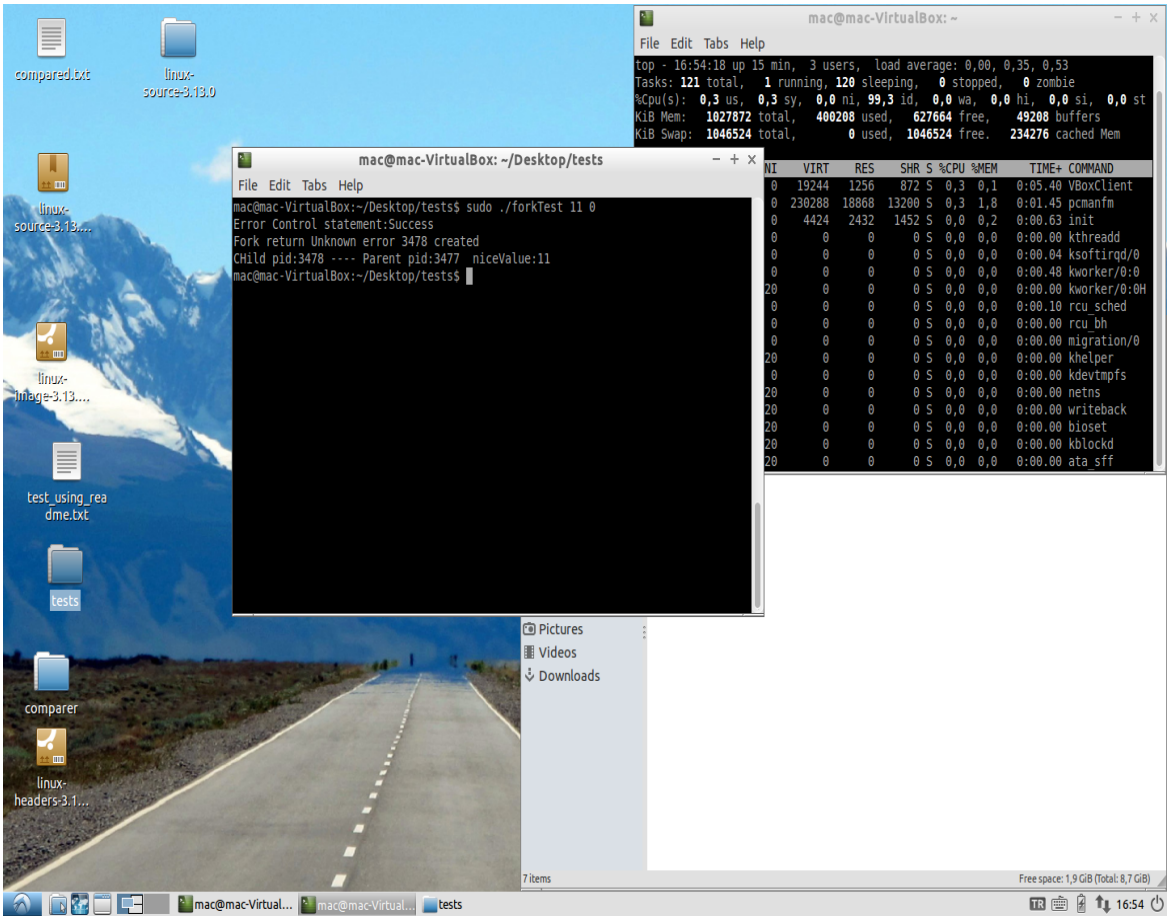
Einval error test



Esrch error test



Fork test with flag 1(echild error test)



Fork test with flag 0