



System Programming

Project 1

Project Report

Group 47

Mehmet Taha Çorbacıoğlu- 150130017

Kağan Özgün - 150130055

31.10.2017

Table of Contents

1	Objective	3
2	Method	3
2.1	Kernel Codes	3
2.1.1	/include/linux/sched.h	3
2.1.2	/include/linux/init_task.h.....	3
2.1.3	/kernel/fork.c	4
2.1.4	/kernel/exit.c	4
2.1.5	Makefile	4
2.1.6	/arch/x86/syscalls/syscall_32.tbl	5
2.1.7	/include/linux/syscalls.h	5
2.1.8	/setmyFlag/setmyFlag.c.....	5
2.2	Test Codes	5
2.2.1	Process Creation, System Call and Nice Value.....	6
2.2.2	Killing Process	7
2.3	Development Environment and Running	8
2.4	Test Outputs.....	9
2.4.1	Access Error	9
2.4.2	Bad Message Error	10
2.4.3	Flag Value = 0, Nice Value = 0.....	10
2.4.4	Flag Value = 0, Nice Value = 1	11
2.4.5	Flag Value = 1, Nice Value = 0.....	12
2.4.6	Flag Value = 1, Nice Value = 1	13
3	Conclusion.....	13

1 Objective

The aim of the project is writing a system call, which sets the value of a flag in the task descriptor of a process and modifying the “exit” system call to change its behavior based on the value of the flag.

2 Method

In this project, we were requested to implement a set function for flag value of the process and exit function that acts according to flag and nice value. To do this, we need to change kernel codes and write a test program that shows the result of the change.

2.1 Kernel Codes

For this project, we must change the kernel codes and compile the whole kernel again. To add flag variable to all process we need to add to process 0 and change the fork function. To kill the children of the appropriate process we need to change exit function. Finally we need to write a system call that change the value of flag.

2.1.1 /include/linux/sched.h

```
root@kagan-VirtualBox:/home/kagan/Desktop# diff sched_custom.h sched.h
1468d1467
<     int myFlag;
root@kagan-VirtualBox:/home/kagan/Desktop#
```

In this file, we add myFlag variable to the task descriptor. By doing this, every process has myFlag variable.

2.1.2 /include/linux/init_task.h

```
root@kagan-VirtualBox:/home/kagan/Desktop# diff init_task_custom.h init_task.h
164d163
<     .myFlag = 0,
root@kagan-VirtualBox:/home/kagan/Desktop#
```

Init_task file is responsible for creating process 0, which is the first process of the system. We initialize myFlag value to 0, for process 0.

2.1.3 /kernel/fork.c

```
root@kagan-VirtualBox:/home/kagan/Desktop# diff fork_custom.c fork.c
1545d1544
<         p->myFlag=0;
root@kagan-VirtualBox:/home/kagan/Desktop#
```

This file is responsible for fork function. By adding this line, every process that forked, now has zero myFlag value.

2.1.4 /kernel/exit.c

```
root@kagan-VirtualBox:/home/kagan/Desktop# diff exit_custom.c exit.c
717,727d716
<
<     int nice_value = current->static_prio - 120;
<     if(tsk->myFlag==1 && nice_value > 10){
<         struct task_struct *task;
<         struct list_head *list;
<         list_for_each(list, &current->children) {
<             task = list_entry(list,struct task_struct, sibling);
<             sys_kill(task->pid, SIGKILL);
<         }
<     }
736d724
<
root@kagan-VirtualBox:/home/kagan/Desktop#
```

This file is responsible for exit function for process. First, we get nice value by subtracting 120 from priority. The reason of that is value of priority is between 100 and 139 while nice value is between -20 and 19. If value of myFlag is 1 and nice value is higher than 10 for process that is exiting, then we kill every children of this process by using list_for_each loop.

2.1.5 Makefile

```
<
root@kagan-VirtualBox:/home/kagan/Desktop# diff Makefile_custom Makefile
540c540
< core-y          := usr/ setmyFlag/
---
> core-y          := usr/
root@kagan-VirtualBox:/home/kagan/Desktop#
```

This file include configuration of compiling the kernel. We add setmyFlag directory to compile list.

2.1.6 /arch/x86/syscalls/syscall_32.tbl

```
root@kagan-VirtualBox:/home/kagan/Desktop# diff syscall_32_custom.tbl syscall_32.tbl
364d363
< 355    i386      setmyFlag                set myFlag
root@kagan-VirtualBox:/home/kagan/Desktop#
```

This is table of the system calls. When system calls invoked, we use their numbers in system call tables. We add new line for setmyFlag function and give 355 for this function.

2.1.7 /include/linux/syscalls.h

```
root@kagan-VirtualBox:/home/kagan/Desktop# diff syscalls_custom.h syscalls.h
852d851
< asmlinkage long set_myFlag(pid_t pid, int flag);
root@kagan-VirtualBox:/home/kagan/Desktop#
```

This is the prototype of the system call that we wrote.

2.1.8 /setmyFlag/setmyFlag.c

```
1  #include <linux/kernel.h>
2  #include <linux/sched.h>
3  #include <asm/errno.h>
4
5  asmlinkage long set_myFlag(pid_t pid, int flag){ //get pid and flag value as parameter
6      struct task_struct *target_task;
7      target_task = find_task_by_vpid(pid); //find task using pid value
8      if(current_uid() != 0){ // check is user root or not
9          return -EACCES; // if not root throw error message access denied
10     }
11     if(target_task == NULL) // check is task null or not
12         return -ESRCH; // if null throw no such task error
13     if(flag == 0 || flag == 1) // check flag value between 0-1
14         target_task->myFlag = flag; // set flag value
15     else
16         return -EBADMSG; // if flag value is not 1 or 0 throw bad message error.
17     return 0;
18 }
19
```

This is system call which we wrote for this project. It takes process id (pid) of the process that we want to change its flag and value of flag as parameters. If system call is not invoked by root privileges then it returns access error. If process id is not valid then it returns no such task error. After these checks, it set flag value to one or zero but if user try to change value something else then it returns bad message error.

2.2 Test Codes

For testing the system call that we wrote, we need to write two test program. One of them is responsible for creating process and its children, and invoke system call and set nice value. The other one is responsible for showing the relation of the process and killing the parent process.

2.2.1 Process Creation, System Call and Nice Value

The purpose of the test program 1 is creating three process that one of them is parent and the other two are children of this parent process. After creating process, parent process takes flag value for itself and set by using system call. If any error occurs, it shows error message. After that program asks that user wants to change nice value of the parent process. If user wants, nice value of the parent process become 11 to make nice value greater than 10. If user does not want to renice to be sure that nice value of parent process is below 10 it sets 0 the nice value. Finally, all three process waits input by using scanf function.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <errno.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <signal.h>
#include <string.h>

int main(){
    int Flag;
    int child1, child2, child3, parent;
    int error;
    int i;
    parent = getpid(); //Parent PID
    char str[10];
    sprintf(str, "%d", parent);    //Parent PID to string

    child1 = fork(); //Parent's first child is created.
    if(child1==0){    //First Child Process
        sleep(10);
        printf("This is a child1 process and pid is %d\nParent pid: %d\n",getpid(),getppid());
        scanf("%d",&i);
    }
    else{            //Parent process continues
        sleep(1);
        child2=fork();
        if(child2==0){ //Parent's second child is created.
            sleep(10);
            printf("This is child2 process and pid is %d\nParent pid: %d\n",getpid(),getppid());
            scanf("%d",&i);
        }
        else{        //Parent process continues
            printf("The parent process has pid %d\n",parent);
            printf("Child1 pid: %d\n", child1);
            printf("Child2 pid: %d\n", child2);
            printf("Enter Flag: "); //Entering Parent flag value
            scanf("%d", &Flag);
```

```

error = syscall(355,parent,Flag); //Setting Parent's flag value
printf("System call returns: %d\n", error); //If error occurs it returns -1 otherwise 0
fprintf(stderr,"value of errno: %d\n",errno); //Prints error codes
perror("Error printed by perror");
/*
ESRCH      3    No such process
EACCES     13   Permission denied
EBADMSG    74   Not a data message
*/

printf("Do you want to renice process.(Yes:1, No:0)"); //Set nice value of parent process 11 to make
greater than 10.
scanf("%d", &i);
if(i ==1){
    char renice[50] = "renice 11 -p "; //Terminal command
    strcat(renice, str); //Adds Parent PID to end of renice string.
    system(renice); //Call terminal command
}
else{
    char nice0[50] = "renice 0 -p "; //Terminal command to ensure that parent has 0 nice value.
    strcat(nice0, str); //Adds Parent PID to end of nice0 string.
    system(nice0); //Call terminal command.
}
wait(NULL);
scanf("%d", &i);
}
}
return 0;
}

```

2.2.2 Killing Process

The main goal of the test program 2 is killing process of the test program 1. To do that first it takes three pid by order of parent, child 1 and child 2. After that, it prints tree of the process and list them. If user want to kill process, it kills and list them again.

```

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <errno.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <signal.h>
#include <string.h>

int main(){
    int parent, child1, child2;

```

```

char Parent[10];
char Child1[10];
char Child2[10];
int i;
printf("Enter parent PID: ");
scanf("%d", &parent);

printf("Enter child1 PID: ");
scanf("%d", &child1);
printf("Enter child2 PID: ");
scanf("%d", &child2);

char command[20] = "pstree -p "; //Terminal command to show as a tree
sprintf(Parent, "%d", parent); //Parent PID to string
strcat(command, Parent); //Adds Parent PID to end of command string.
system(command); //Call terminal command

char ps[20] = "ps "; //Terminal command to show all process(parent, child1, child2)

strcat(ps, Parent); //Adds Parent PID to end of ps string.
strcat(ps, " ");
sprintf(Child1, "%d", child1); //Child1 PID to string
strcat(ps, Child1); //Adds Child1 PID to end of ps string.
strcat(ps, " ");
sprintf(Child2, "%d", child2); //Child2 PID to string
strcat(ps, Child2); //Adds Child2 PID to end of ps string.
system(ps); //Call terminal command

printf("1 : to kill\n");
printf("0 : cont\n");
scanf("%d", &i);
if(i == 1){
    char kill[50] = "kill "; //Terminal command to kill parent process
    strcat(kill, Parent); //Adds Parent PID to end of kill string.
    system(kill); //Call terminal command
}
system(ps); //Call terminal command to show process after kill.
}

```

2.3 Development Environment and Running

We use Ubuntu 14.04 operating system to develop the program. It requires some packages for editing and installing the kernel. You can install by:

sudo apt-get install manpages-dev g++ patch strace ltrace linux-headers-\$(uname -r) linux-source kernel-package fakeroot libncurses5-dev libfuse-dev

After installation is completed. You can compile the kernel by using:


```
make-kpkg clean      // to clean up all from previous kernel compiles  
make defconfig      // generates a kernel configuration with the default answer  
fakeroot make-kpkg --initrd --append-to-version=-custom kernel_image  
kernel_headers
```

If you add -j4 to fakeroot command then compiling use 4 core and can be much faster to build. You can change the value by your core number.

Test programs can be compiled by gcc.

```
gcc test.c -o test.out  
gcc test2.c -o test.out
```

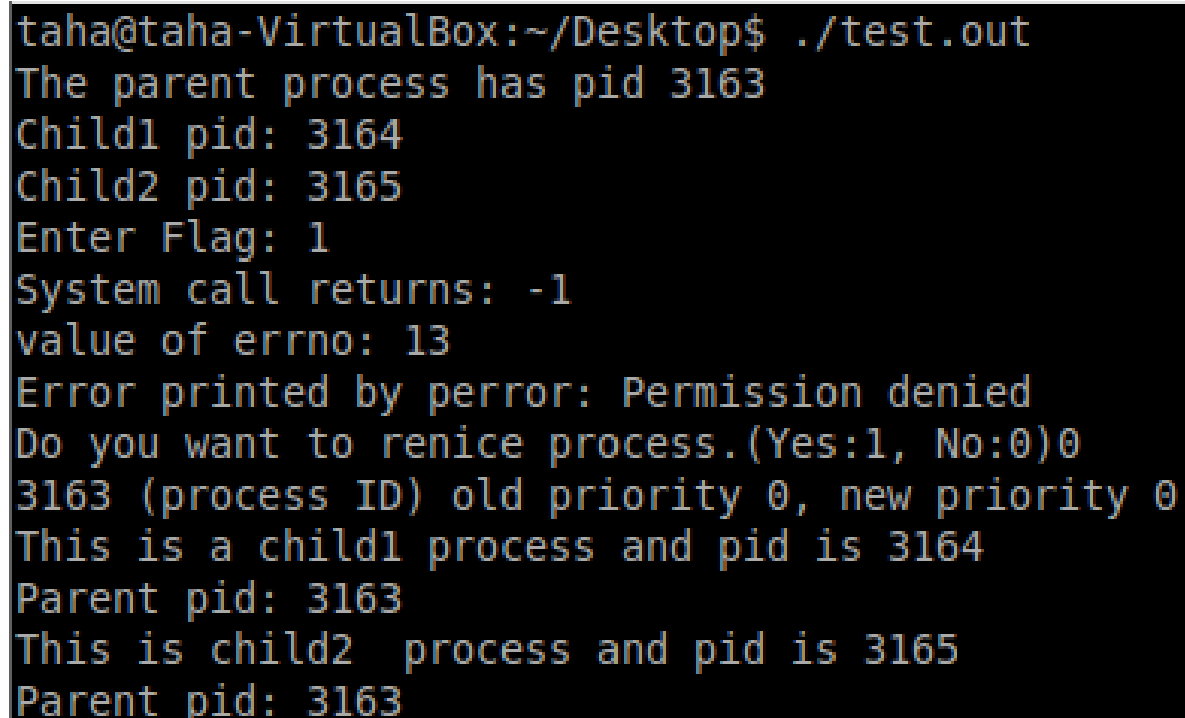
To run the test programs you should write to terminal,

```
./test.out  
./test2.out
```

2.4 Test Outputs

As we discuss in Test Codes, there are two test program one of them for creating process and setting value of flag and nice value. The second one is for killing process. The first two outputs are outputs of test program one and the rest are test program two.

2.4.1 Access Error



```
taha@taha-VirtualBox:~/Desktop$ ./test.out  
The parent process has pid 3163  
Child1 pid: 3164  
Child2 pid: 3165  
Enter Flag: 1  
System call returns: -1  
value of errno: 13  
Error printed by perror: Permission denied  
Do you want to renice process.(Yes:1, No:0)0  
3163 (process ID) old priority 0, new priority 0  
This is a child1 process and pid is 3164  
Parent pid: 3163  
This is child2 process and pid is 3165  
Parent pid: 3163
```

If test program one did not open by sudo then it gives access error and system call will not change the value of flag for parent process.

2.4.2 Bad Message Error

```
taha@taha-VirtualBox:~/Desktop$ sudo ./test.out
The parent process has pid 3261
Child1 pid: 3262
Child2 pid: 3263
Enter Flag: 10
System call returns: -1
value of errno: 74
Error printed by perror: Bad message
Do you want to renice process.(Yes:1, No:0)0
3261 (process ID) old priority 0, new priority 0
^Ctaha@taha-VirtualBox:~/Desktop$
```

If user gives value of flag other than 1 or 0 than it gives bad message error.

2.4.3 Flag Value = 0, Nice Value = 0

```
taha@taha-VirtualBox:~/Desktop$ sudo ./test2.out
Enter parent PID: 3226
Enter child1 PID: 3227
Enter child2 PID: 3228
test.out(3226)└─test.out(3227)
               └─test.out(3228)

  PID TTY          STAT       TIME COMMAND
  3226 pts/0        S+          0:00 ./test.out
  3227 pts/0        S+          0:00 ./test.out
  3228 pts/0        S+          0:00 ./test.out
1 : to kill
0 : cont
1
  PID TTY          STAT       TIME COMMAND
  3227 pts/0        S           0:00 ./test.out
  3228 pts/0        S           0:00 ./test.out
taha@taha-VirtualBox:~/Desktop$
```

If flag value is set 0 and nice value is set 0 in test program one, then when we kill parent process, only the parent process dies and children live. Children will not die because exit function kills children when both condition is provided.

2.4.4 Flag Value = 0, Nice Value = 1

```
taha@taha-VirtualBox:~/Desktop$ sudo ./test2.out
Enter parent PID: 3282
Enter child1 PID: 3283
Enter child2 PID: 3284
test.out(3282)└─test.out(3283)
               └─test.out(3284)

  PID TTY          STAT       TIME COMMAND
 3282 pts/0        SN+         0:00   ./test.out
 3283 pts/0        S+          0:00   ./test.out
 3284 pts/0        S+          0:00   ./test.out
1 : to kill
0 : cont
1
  PID TTY          STAT       TIME COMMAND
 3283 pts/0        S           0:00   ./test.out
 3284 pts/0        S           0:00   ./test.out
taha@taha-VirtualBox:~/Desktop$
```

If flag value is set 0 and nice value is set 1 in test program one, then when we kill parent process, only the parent process dies and children live. Children will not die because exit function kills children when both condition is provided.

2.4.5 Flag Value = 1, Nice Value = 0

```
taha@taha-VirtualBox:~/Desktop$ sudo ./test2.out
Enter parent PID: 3328
Enter child1 PID: 3329
Enter child2 PID: 3330
test.out(3328)└─test.out(3329)
               └─test.out(3330)

  PID TTY          STAT       TIME COMMAND
  3328 pts/0        S+          0:00 ./test.out
  3329 pts/0        S+          0:00 ./test.out
  3330 pts/0        S+          0:00 ./test.out
1 : to kill
0 : cont
1
  PID TTY          STAT       TIME COMMAND
  3329 pts/0        S           0:00 ./test.out
  3330 pts/0        S           0:00 ./test.out
taha@taha-VirtualBox:~/Desktop$
```

If flag value is set 1 and nice value is set 0 in test program one, then when we kill parent process, only the parent process dies and children live. Children will not die because exit function kills children when both condition is provided.

2.4.6 Flag Value = 1, Nice Value = 1

```
taha@taha-VirtualBox:~/Desktop$ sudo ./test2.out
Enter parent PID: 3358
Enter child1 PID: 3359
Enter child2 PID: 3360
test.out(3358)└─test.out(3359)
               └─test.out(3360)

  PID TTY          STAT       TIME COMMAND
 3358 pts/0    SN+         0:00   ./test.out
 3359 pts/0    S+          0:00   ./test.out
 3360 pts/0    S+          0:00   ./test.out
1 : to kill
0 : cont
1
  PID TTY          STAT       TIME COMMAND
taha@taha-VirtualBox:~/Desktop$
```

If flag value is set 1 and nice value is set 1 in test program one, then when we kill parent process, also children of the parent process will die.

3 Conclusion

Final product of the project is a new kernel that can set flag value of the process and kill children of the process by its flag value and nice value. To show the result of the product we also write two test programs that one of them creates process and sets the flag and nice value, while the other one shows the process tree and kills them.

We had learned to change kernel functions, add new system call and compile the kernel. In addition, in a user process, we had learned to invoke a system call and terminal command.