

**BLG 413E**

**SYSTEM PROGRAMMING**

CRN: 12300

**REPORT OF PROJECT #2**

Submission Date: 25.11.2014

**Members of Group #16:**

Mustafa UÇAR 040100113

Tuğrul YATAĞAN 040100117

Emre GÖKREM 040100124

**Demo Assistant:**

Mustafa ERSEN

# **Introduction**

A system call which either only terminates all the children of a given process or terminates all children, all siblings and their children of a given process is written (The process itself is not terminated). Only processes having root privileges can successfully execute this system call.

The “exit” system call is also modified. If the “nice” value of the process which has executed the “exit” system call is greater than 10, then the “exit” system call also use the new system call **(my\_process\_terminator**) we have written with flag=0, all children of the exiting process are terminated along with the process itself. If a parent process terminated, children process of terminated process are adopted by init (pid:1) process in Linux.

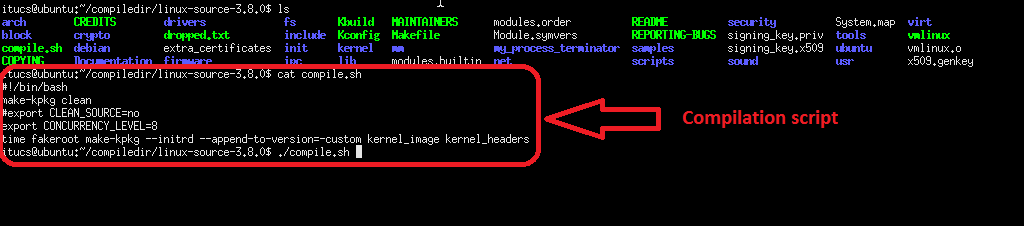
# **Compilation and Installation**

The custom kernel is compiled with a bash script with modified Makefile:

#!/bin/bash

make-kpkg clean

time fakeroot make-kpkg --initrd --append-to-version=-custom kernel\_image kernel\_headers



Compilation finished about half an hour:



The new kernel installed with a bash script:

#!/bin/bash

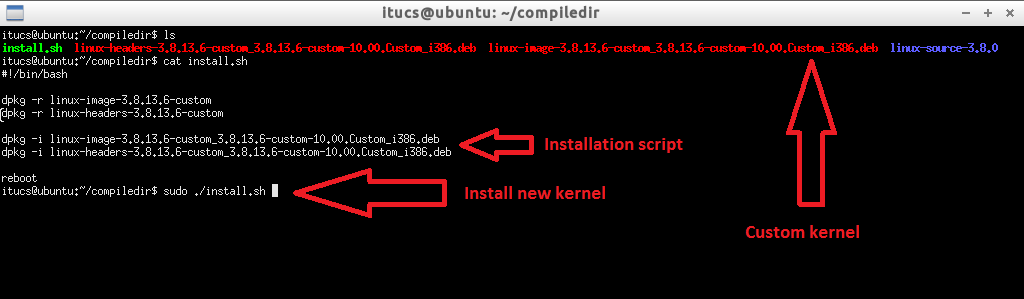
dpkg -r linux-image-3.8.13.6-custom

dpkg -r linux-headers-3.8.13.6-custom

dpkg -i linux-image-3.8.13.6-custom\_3.8.13.6-custom-10.00.Custom\_i386.deb

dpkg -i linux-headers-3.8.13.6-custom\_3.8.13.6-custom-10.00.Custom\_i386.deb

reboot



After reboot, now we are working on new kernel.

# **Running**

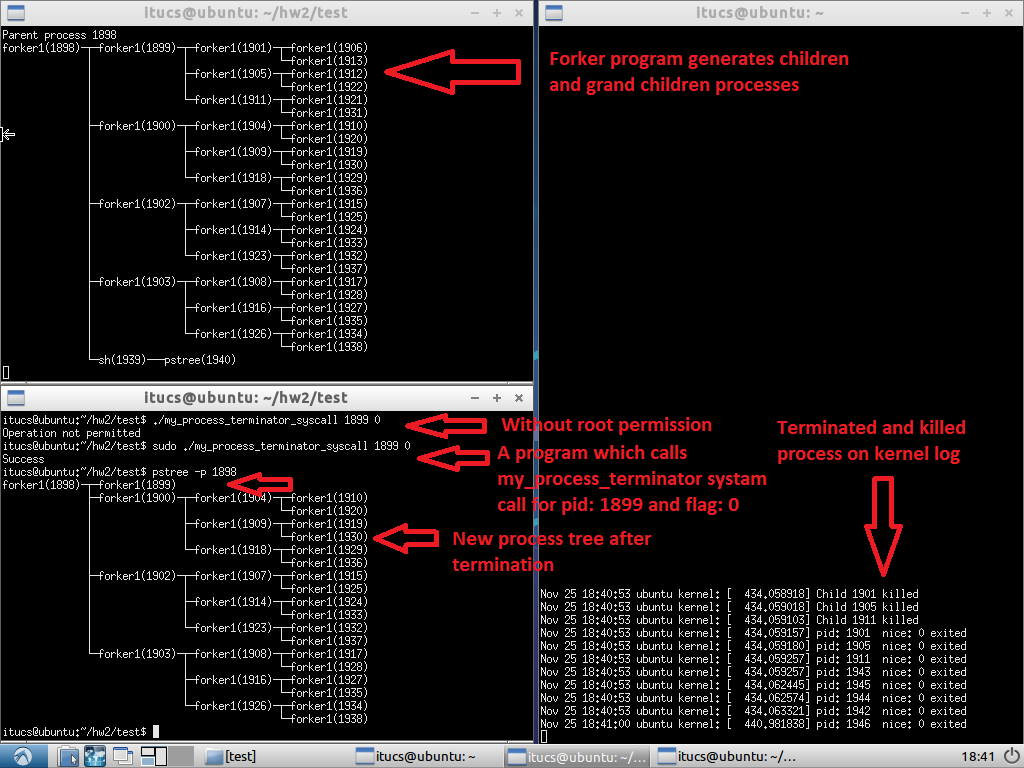
# **New Process Terminator System Call**

New system call can be tested by our test programs. First program (**forker1**) generates children and grandchildren processes with fork system call, second program (**my\_process\_terminator\_syscall**) calls new **my\_process\_terminator** system call with parameters PID and flag.

System calls tested with and without root permission. New process tree is shown after termination.

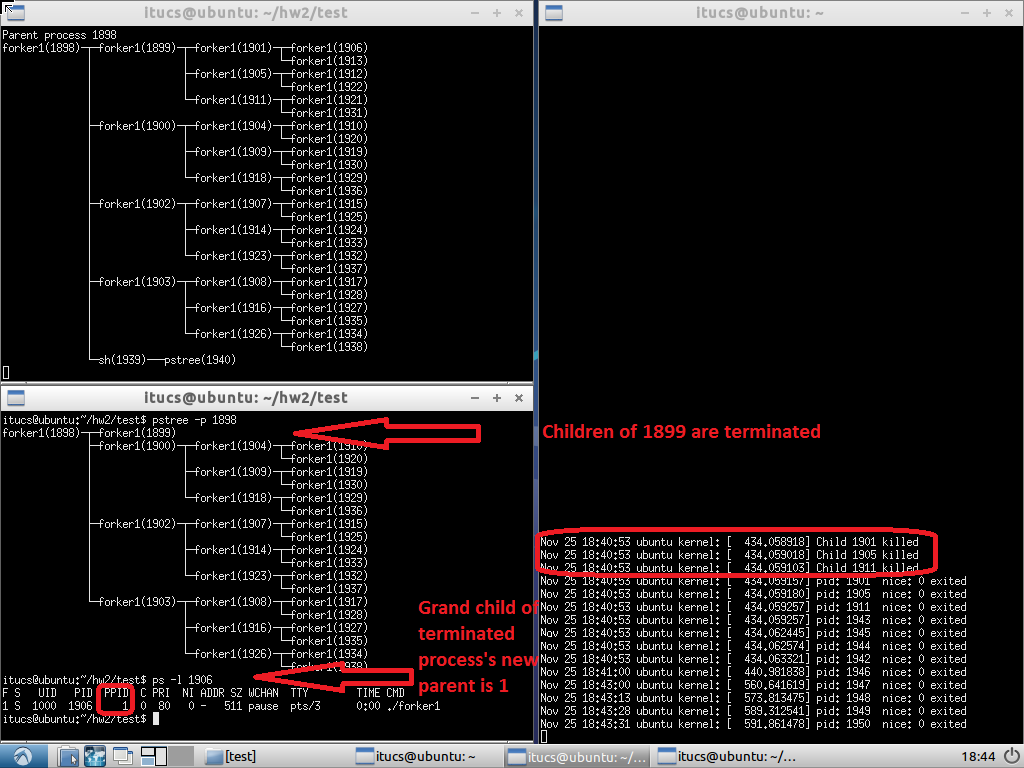
# **Flag 0**

Firstly, **my\_process\_terminator** system call called with flag 0 in test program (**my\_process\_terminator\_syscall**):



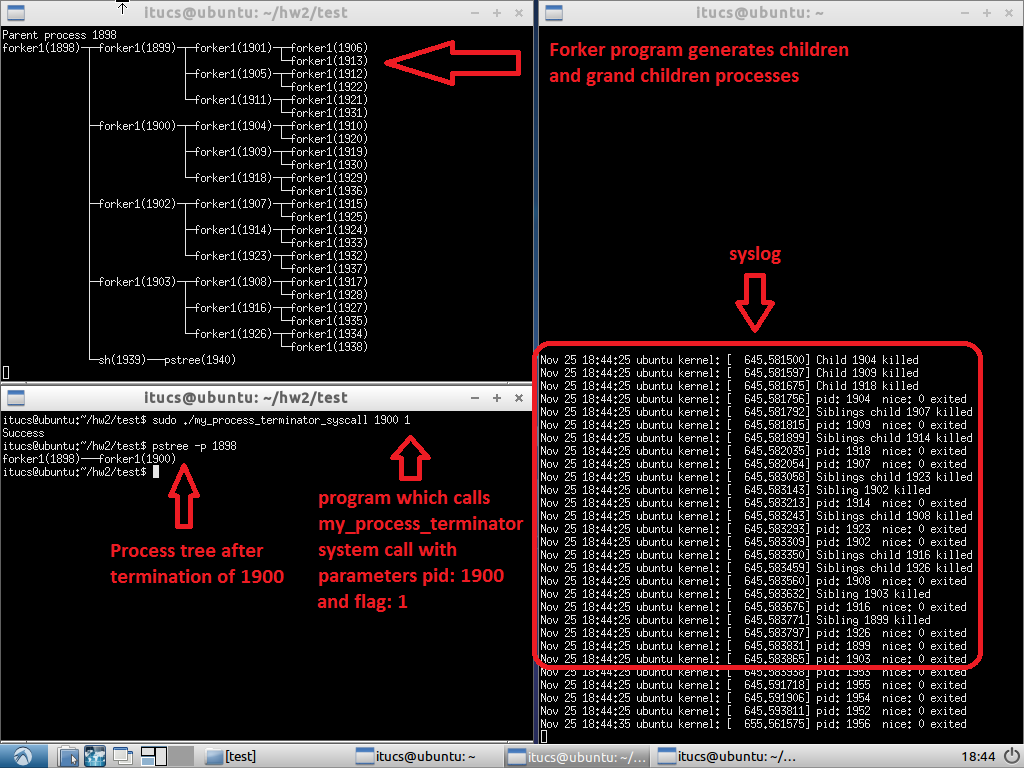
After calling system call with flag 0, children processes are terminated but process itself, siblings and grandchildren processes are alive.

After termination of process with flag 0, orphan processes are adopted by init process:



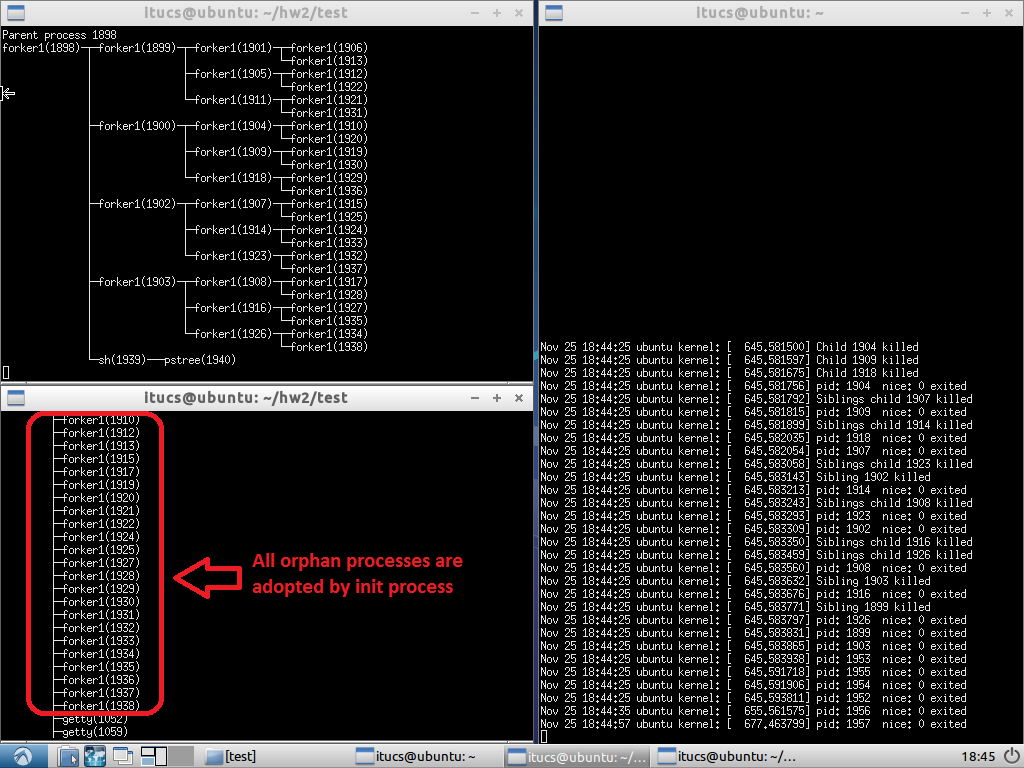
# **Flag 1**

Secondly, **my\_process\_terminator** system call called with flag 1 in test program (**my\_process\_terminator\_syscall**):



After calling system call with flag 1, all siblings and children processes are terminated but process itself and grandchildren processes are alive.

After termination of process with flag 1, orphan processes are adopted by init process:



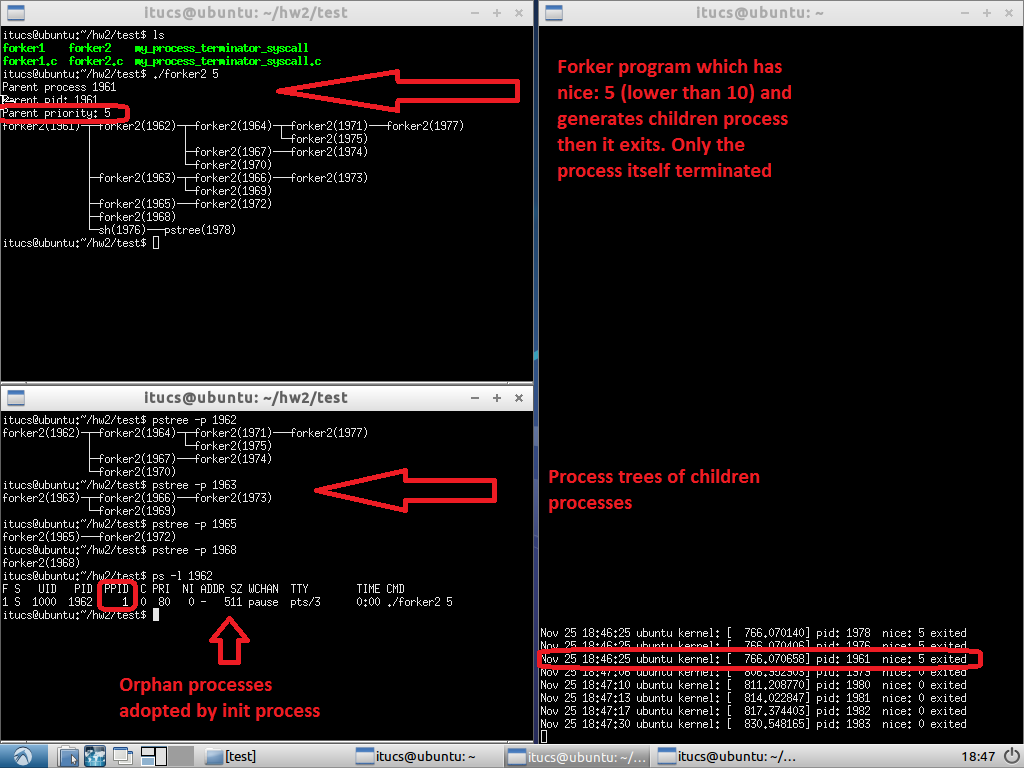
# **Modified Exit System Call**

Modified exit system call can be tested by our test program (**forker2**) which generates children and grandchildren processes with fork system call. The program takes nice value as an argument.

System call tested with and without root permission. New process tree is shown after termination.

# **Nice value lower than 10**

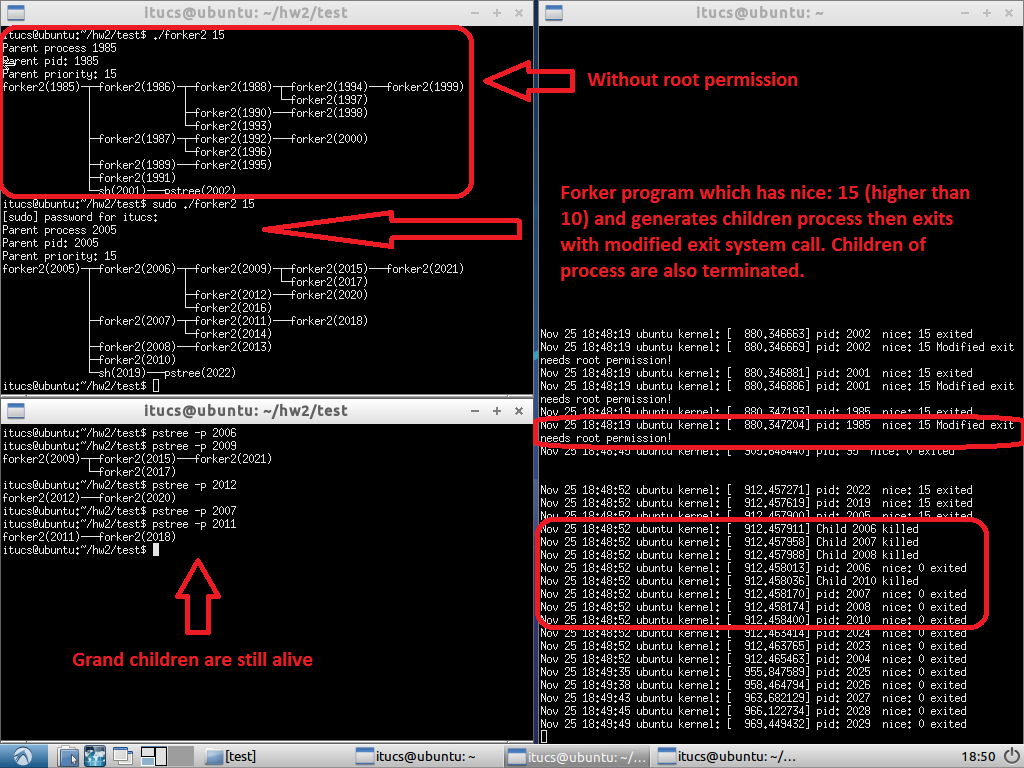
Firstly, modified **exit** system call called with nice value 5 (lower than 10) in test program (**forker2**):



Only the process itself is terminated. Exit system call works as normally.

# **Nice value greater than 10**

Secondly, modified **exit** system call called with nice value 15 (greater than 10) in test program (**forker2**):



The process and also children of the process are terminated. Exit system call works with new **my\_process\_terminator** system call.

# **Code**

# **New Process Terminator System Call**

Firstly root permission check is done:

if(! capable(CAP\_SYS\_ADMIN)){

return EPERM;

}

Finding and killing child process of given PID:

struct task\_struct \*p; // process task struct

struct task\_struct \*child\_task;

struct list\_head \*children\_list;

for\_each\_process(p){

if(p->pid == pid){

list\_for\_each(children\_list, &(p->children)){

child\_task = list\_entry(children\_list, struct task\_struct, sibling);

printk("Child %d killed\n",child\_task->pid);

sys\_kill(child\_task->pid,SIGKILL);

}

}

}

**for\_each\_process** iterates over all process in the system, **list\_for\_each** iterates over process’s children list and **sys\_kill** kills processes with **SIGKILL** flag.

# **Modified Exit System Call**

Exit system call modified with **exit.c** file under the kernel directory in Linux source code. In the **exit.c** file, only the **do\_exit** function modified. These are added to head of the function:

printk("pid: %d nice: %d exited\n", tsk->pid, task\_nice(tsk));

if (task\_nice(tsk) > 10){ // gets nice value of the process

if(capable(CAP\_SYS\_ADMIN)){

sys\_my\_process\_terminator(tsk->pid,0);

}

else{

printk("pid: %d nice: %d Modified exit needs root permission!\n", tsk->pid, task\_nice(tsk));

}

}

If nice value of process is greater than 10, modified code block runs instead of original code. **task\_nice** macro gets nice value of process and then if process has permission new **sys\_my\_process\_terminator** system call called with flag 0.