**Istanbul Technical University**

**Faculty of Computer and Informatics**



**BLG413E System Programming**

**Project 2 Report**

**Group 28**

**Cem Yusuf Aydoğdu**

**150120251**

1. Introduction

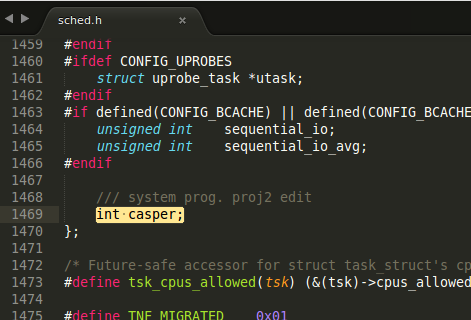
In this project, a new integer variable(**int casper**) which is used to control visibility of a process in **/proc** filesystem was added to the task descriptor and a set\_casper system call was implemented in order to change the control value of a given process.

Required visibility behaviours for each **casper** value are shown below:

|  |  |
| --- | --- |
| Casper value | Required behaviour |
| 0 | Process is visible to all |
| 1 | Process is visible to only processes  with same used id |
| 2 | Process is visible to only processes  with same group id |
| 3 | Process is not visible for any process |

1. Changes in kernel

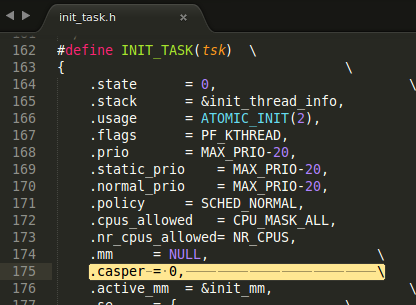
Initially, **int casper** variable was added to the end of **struct task\_struct** in **/include/linux/sched.h** file. **struct task\_struct** is task descriptor which holds information(pid, state, flags...) about a process. For every process, there exists a task descriptor in the system.



*Screenshot 1: /include/linux/sched.h*

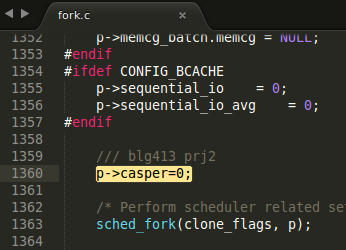
After that**, INIT\_TASK** macro in **/include/linux/init\_task.h** file was changed in order to initialize **casper** value to zero for Process 0.

Process 0 is the first process created in kernel startup stage which is responsible for initialization of some core system functionalities. **INIT\_TASK** macro only initializes this process.



*Screenshot 2: /include/linux/init\_task.h*

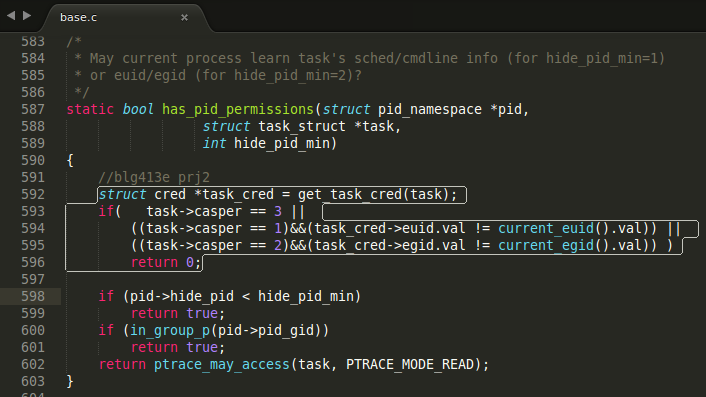
Then, casper was assigned to zero in **copy\_process** function in the **kernel/fork.c** file. This file contains functions about fork system call which is used to create processes. **copy\_process** function is responsible for the core functionality of the new process creation. This function creates a new **task\_struct** with information from a given old process, but it does not start the created process.



*Screenshot 3: /kernel/fork.c*

Finally, **has\_pid\_permissions** function in **/fs/proc/base.c** file was modified to achieve different visibility requirements . This function checks whether the current process(given as **\*pid** in the first argument) can or can not see the task’s(given as **\*task** in second argument) information. If the current process is not authorized, this function returns zero, vice versa.

In order to implement different visibility features, casper value of the task was obtained from **get\_task\_cred function** and compared with required values. If the **casper** value is 3, or **casper** value is 1 but user ids of task and current process does not match, or **casper** value is 2 but group ids of task and current process does not match; it returns 0 as false.



*Screenshot 4: /fs/proc/base.c*

1. System Call



The system call assigns ***casper*** value to a process with given pid. Initially, the system call returns “Invalid argument” error (**EINVAL**) if the given casper value is invalid.

Then, it uses a **read\_lock** in order to lock access to the critical section. After that, it finds the process using **find\_task\_by\_vpid** function. If there are no process with that given **pid**, it first unlocks the read spin lock and returns “No such process” error (**ESRCH**).

If it finds the process, it again unlocks the read lock and uses a write lock with **write\_lock\_irq** function which disables interrupts, in order to change **casper** value.

In order to add the system call, it was added to the end of the system call table **(**a**rch/x86/syscalls/syscall\_32.tbl)** and system call header file which is in **include/linux/syscalls.h**.

1. Test Code

Finally, after all implementations the kernel was compiled and installed to the system. Then, a basic test program was written to test functionalities.

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#define NR\_set\_casper 355

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

{

**if**(argc!=3 || geteuid() != 0)

{

printf("usage: sudo ./executable pid value\n");

**return** -1;

}

**int** pid = atoi(argv[1]), val=atoi(argv[2]);

printf("casper value %d, pid %d\n",val, pid);

**long** result = syscall(NR\_set\_casper, pid, val);

printf("syscall result:%ld \n ",result);

**if**(result == 0)

printf("Success\n");

**else**

{

printf("Error: ");

**if**(errno== EINVAL)

printf("Invalid value\n");

**if**(errno==ESRCH)

printf("Process not found\n");

}

**return** 0;

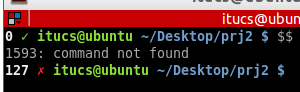
}

Number of the system call was denoted with **NR\_set\_casper** macro. Program initially checks for argument number and super user privilege conditions. Then, it parses first argument as **pid** and second argument as **casper** value.

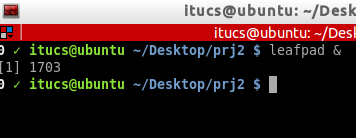
After, it uses **syscall** function to call **set\_casper** system call and stores the result in a long integer. It informs about success and if an error occures, it prints the required error message before exiting.

1. Results

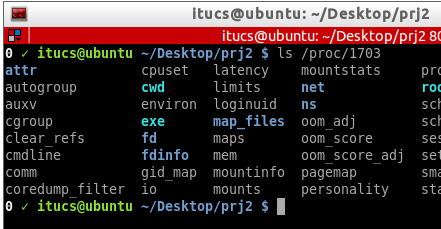
In order to test the required functionalities, process id of bash was found as *1593* with $$ command. Also, an arbitrary program with pid number *1703* is called from bash. That process has the same user and group id with bash.



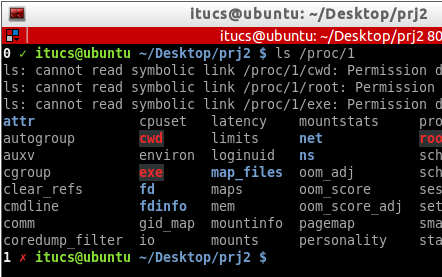
*Screenshot 5: Process ID of the bash*



*Screenshot 6: A new process with pid=1703*

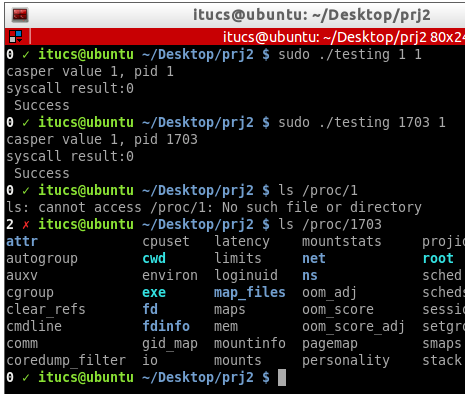
Since casper=0 initially, any process should be visible on the /proc filesystem. Testing with different processes from different users proves that.

*Screenshot 7: Process with same user and group id, when casper=0*



*Screenshot 8: Process with different user and group id, when casper=0*

When **casper=1**, only processes with same user id should be visible. Testing with *1* and *1703* as process ids:



*Screenshot 9: Testing for casper=1*

In order to test the condition where **casper=2**, a bash script was written which shows process groups for each process under the **/proc**.

## Script to print groups of processes in /proc

#! /bin/bash

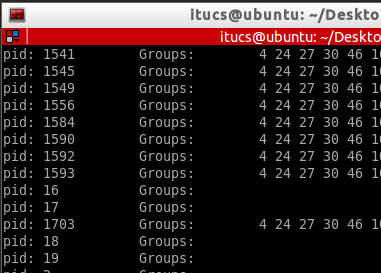
**PID\_LIST**=$(ls /proc/ | awk '/[0-9]/ {print}') ## awk finds numbers in /proc

for **PID** in $**PID\_LIST**

do

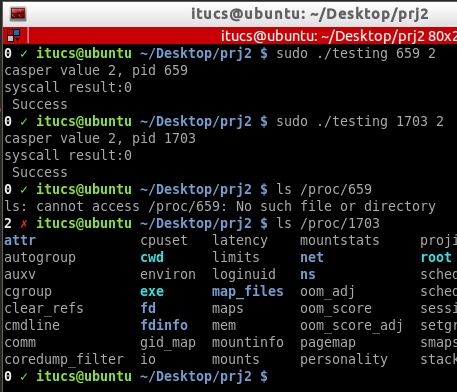
echo -e "pid: $PID \t $(cat /proc/$PID/status | grep Groups)"

done



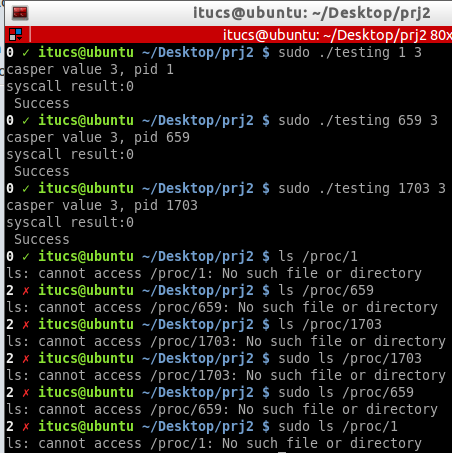
*Screenshot 10: Output of print\_groups.sh script*

Two processes with different groups were selected to test **casper=2** condition. *659* pid does not have any common group with the bash, but all groups of *1703* pid are same with bash:



*Screenshot 11: Testing for casper=2, with processes 659 and 1703*

Testing for **casper=3**, for different processes with different user and group ids. All these processes are invisible, even with super user privileges.



*Screenshot 12: Testing for casper=3*