Istanbul Technical University
Faculty of Computer and Informatics

# BLG413E System Programming
# Project 2 Report
# Group 28

Cem Yusuf Aydoğdu
150120251

## 1. Introduction

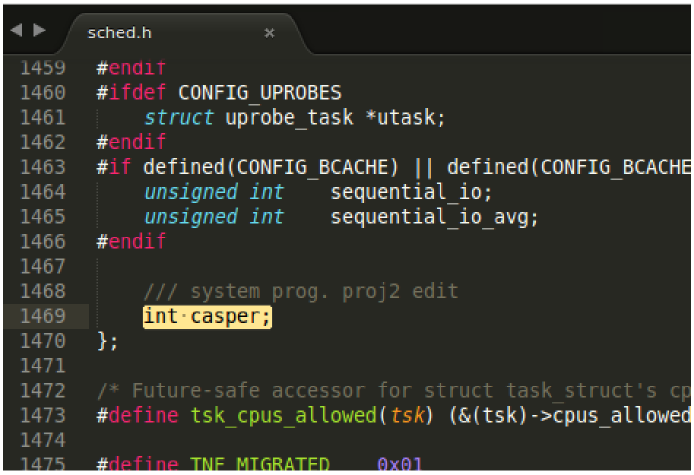
In this project, a new integer variable(**int casper**) which is used to control visibility of a process in **/proc** filesystem was added to the task descriptor and a set_casper system call was implemented in order to change the control value of a given process.

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 |

## 2. 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.

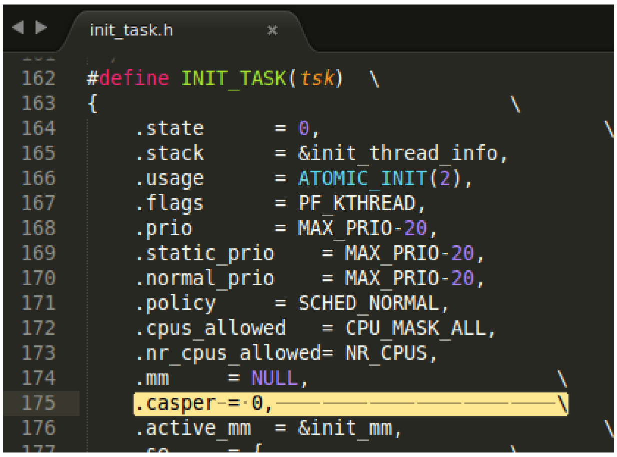


```
#endif
#ifdef CONFIG_UPROBES
	struct uprobe_task *utask;
#endif
#if defined(CONFIG_BCACHE) || defined(CONFIG_BCACHE
	unsigned int	sequential_io;
	unsigned int	sequential_io_avg;
#endif

	/// system prog. proj2 edit
	int casper;
};

/* Future-safe accessor for struct task_struct's cp
#define tsk_cpus_allowed(tsk) (&(tsk)->cpus_allowed

#define TNF_MIGRATED	0x01
```

***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.

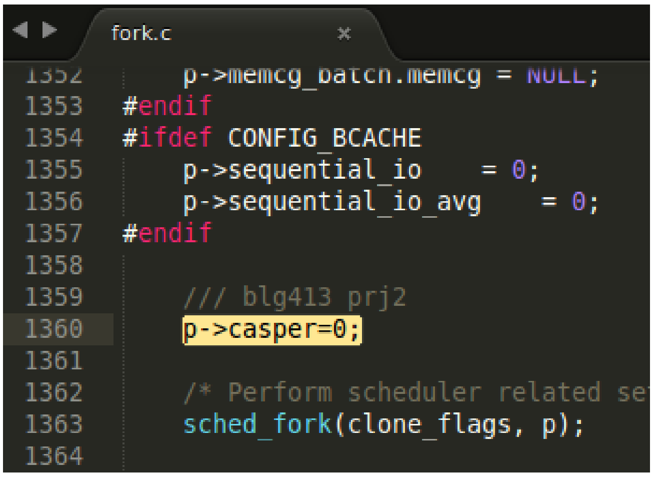
Process 0 is the first process created in kernel startup stage which is responsible for initialization of some core system functionalities. **INIT_TASK** macro only initializes this process.



```
#define INIT_TASK(tsk)  \
{                                       \
    .state      = 0,                            \
    .stack      = &init_thread_info,            \
    .usage      = ATOMIC_INIT(2),               \
    .flags      = PF_KTHREAD,                   \
    .prio       = MAX_PRIO-20,                  \
    .static_prio    = MAX_PRIO-20,              \
    .normal_prio    = MAX_PRIO-20,              \
    .policy     = SCHED_NORMAL,                 \
    .cpus_allowed   = CPU_MASK_ALL,             \
    .nr_cpus_allowed= NR_CPUS,                  \
    .mm     = NULL,                             \
    .casper = 0,                                \
    .active_mm  = &init_mm,                     \
```

***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.

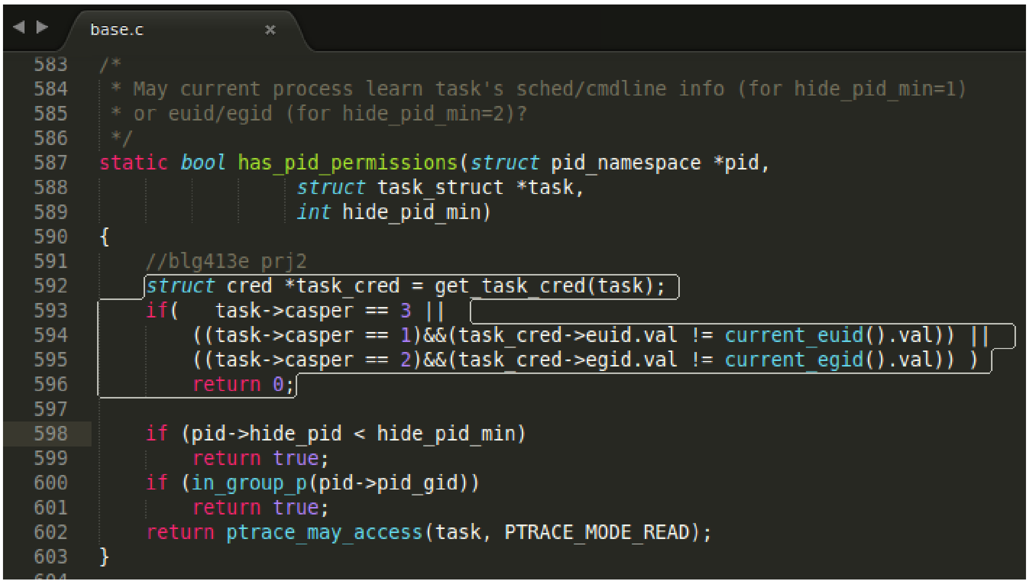


```
    p->memcg_batch.memcg = NULL;
#endif
#ifdef CONFIG_BCACHE
    p->sequential_io     = 0;
    p->sequential_io_avg     = 0;
#endif

    /// blg413 prj2
    p->casper=0;

    /* Perform scheduler related se
    sched_fork(clone_flags, p);

```

***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.



```
/*
 * May current process learn task's sched/cmdline info (for hide_pid_min=1)
 * or euid/egid (for hide_pid_min=2)?
 */
static bool has_pid_permissions(struct pid_namespace *pid,
                                struct task_struct *task,
                                int hide_pid_min)
{
    //blg413e prj2
    struct cred *task_cred = get_task_cred(task);
    if(    task->casper == 3 ||
        ((task->casper == 1)&&(task_cred->euid.val != current_euid().val)) ||
        ((task->casper == 2)&&(task_cred->egid.val != current_egid().val)) )
        return 0;

    if (pid->hide_pid < hide_pid_min)
        return true;
    if (in_group_p(pid->pid_gid))
        return true;
    return ptrace_may_access(task, PTRACE_MODE_READ);
}
```

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

## 3. System Call

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

asmlinkage long set_casper(pid_t pid, int value)
{
  if(value != 0 && value != 1 && value != 2 && value != 3)
  {
    return -EINVAL;
  }

  struct task_struct *p;
  read_lock(&tasklist_lock);
  p=find_task_by_vpid(pid);
  if(p == NULL)
  {
    read_unlock(&tasklist_lock);
    return -ESRCH;
  }
  read_unlock(&tasklist_lock);
  write_lock_irq(&tasklist_lock);
  p->casper = value;
  write_unlock_irq(&tasklist_lock);

  return 0;
}
```

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 **(arch/x86/syscalls/syscall_32.tbl)** and system call header file which is in **include/linux/syscalls.h**.

## 4. 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.

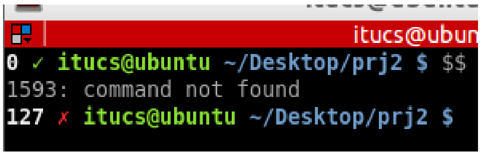
```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#define NR_set_casper 355

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.

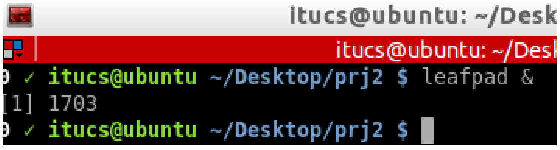
## 5. Results

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.




```
0 ✓ itucs@ubuntu ~/Desktop/prj2 $ $$
1593: command not found
127 ✗ itucs@ubuntu ~/Desktop/prj2 $
```

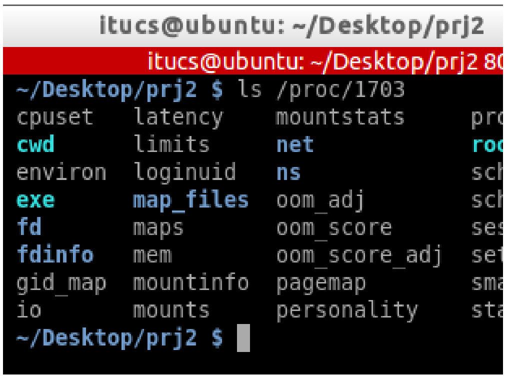
***Screenshot 5:*** *Process ID of the bash*




```
0 ✓ itucs@ubuntu ~/Desktop/prj2 $ leafpad &
[1] 1703
0 ✓ itucs@ubuntu ~/Desktop/prj2 $ 
```

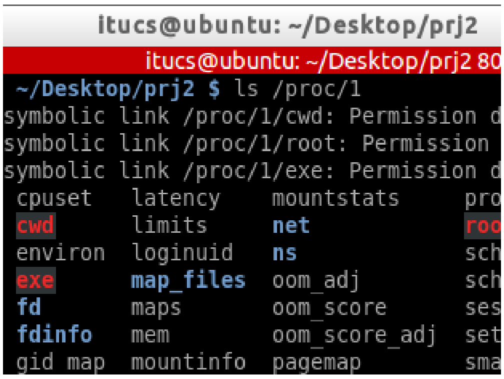
***Screenshot 6:*** *A new process with pid=1703*

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




```
~/Desktop/prj2 $ ls /proc/1703
cpuset    latency    mountstats     pro
cwd       limits     net            roo
environ   loginuid   ns             sch
exe       map_files  oom_adj        sch
fd        maps       oom_score      ses
fdinfo    mem        oom_score_adj  set
gid_map   mountinfo  pagemap        sma
io        mounts     personality    sta
~/Desktop/prj2 $ 
```

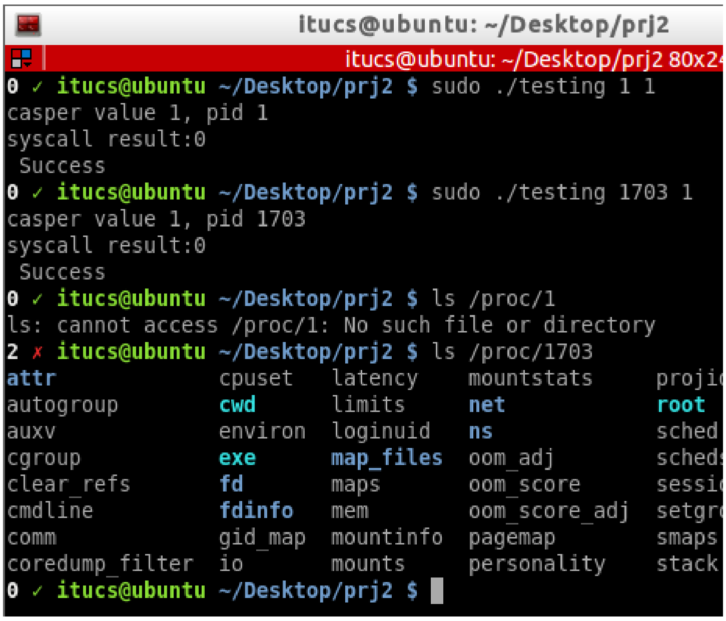
***Screenshot 7:*** *Process with same user and group id, when casper=0*




```
 ~/Desktop/prj2 $ ls /proc/1
symbolic link /proc/1/cwd: Permission d
symbolic link /proc/1/root: Permission
symbolic link /proc/1/exe: Permission d
 cpuset    latency    mountstats     pro
 cwd       limits     net            roo
 environ   loginuid   ns             sch
 exe       map_files  oom_adj        sch
 fd        maps       oom_score      ses
 fdinfo    mem        oom_score_adj  set
 gid_map   mountinfo  pagemap        sma
```

***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:

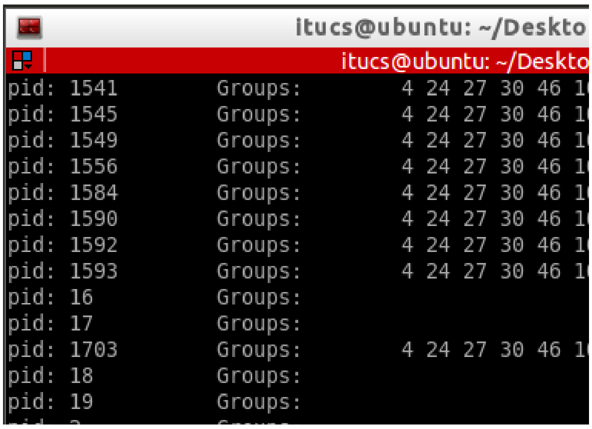

itucs@ubuntu: ~/Desktop/prj2
itucs@ubuntu: ~/Desktop/prj2 80x24

```
0 ✓ itucs@ubuntu ~/Desktop/prj2 $ sudo ./testing 1 1
casper value 1, pid 1
syscall result:0
 Success
0 ✓ itucs@ubuntu ~/Desktop/prj2 $ sudo ./testing 1703 1
casper value 1, pid 1703
syscall result:0
 Success
0 ✓ itucs@ubuntu ~/Desktop/prj2 $ ls /proc/1
ls: cannot access /proc/1: No such file or directory
2 ✗ itucs@ubuntu ~/Desktop/prj2 $ ls /proc/1703
attr             cpuset   latency    mountstats     proji
autogroup        cwd      limits     net            root
auxv             environ  loginuid   ns             sched
cgroup           exe      map_files  oom_adj        sched
clear_refs       fd       maps       oom_score      sessi
cmdline          fdinfo   mem        oom_score_adj  setgr
comm             gid_map  mountinfo  pagemap        smaps
coredump_filter  io       mounts     personality    stack
0 ✓ itucs@ubuntu ~/Desktop/prj2 $ █
```

***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**.

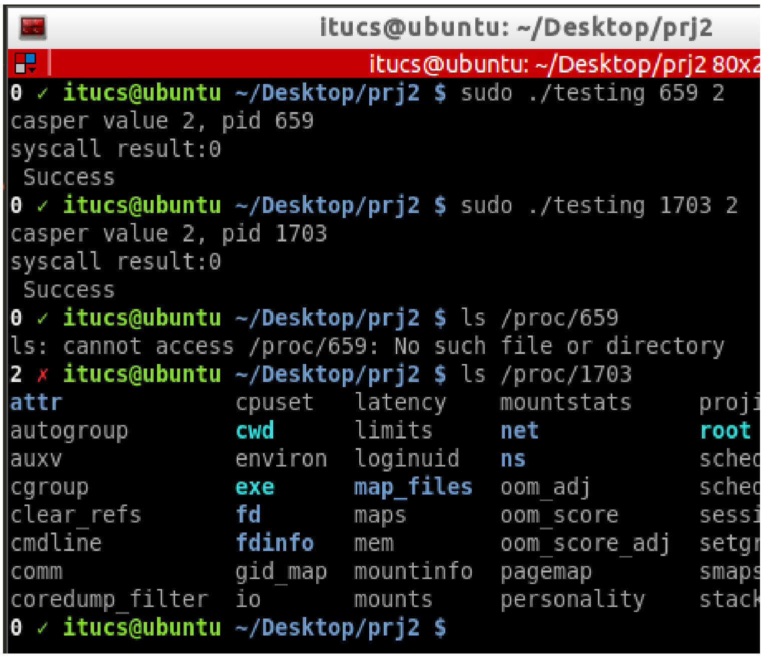
```
## Script to print groups of processes in /proc
#! /bin/bash
PID_LIST=$(ls /proc/ | awk '/[0-9]/ {print}')          ## awk finds numbers in /proc

for PID in $PID_LIST
do
        echo -e "pid: $PID  \t $(cat /proc/$PID/status | grep Groups)"
done
```



```
pid: 1541        Groups:        4 24 27 30 46 1
pid: 1545        Groups:        4 24 27 30 46 1
pid: 1549        Groups:        4 24 27 30 46 1
pid: 1556        Groups:        4 24 27 30 46 1
pid: 1584        Groups:        4 24 27 30 46 1
pid: 1590        Groups:        4 24 27 30 46 1
pid: 1592        Groups:        4 24 27 30 46 1
pid: 1593        Groups:        4 24 27 30 46 1
pid: 16          Groups:
pid: 17          Groups:
pid: 1703        Groups:        4 24 27 30 46 1
pid: 18          Groups:
pid: 19          Groups:
```

***Screenshot 10:*** *Output of print_groups.sh script*

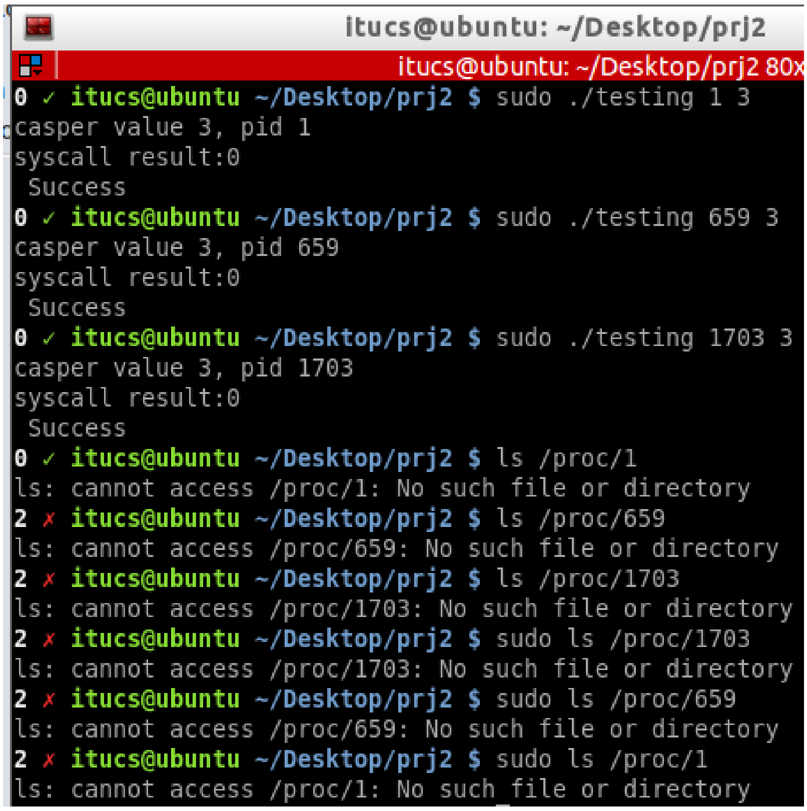
Two processes with different groups were selected to test **casper=2** condition. *659* pid does not have any common group with the bash, but all groups of *1703* pid are same with bash:


itucs@ubuntu: ~/Desktop/prj2
itucs@ubuntu: ~/Desktop/prj2 80x2

```
0 ✓ itucs@ubuntu ~/Desktop/prj2 $ sudo ./testing 659 2
casper value 2, pid 659
syscall result:0
 Success
0 ✓ itucs@ubuntu ~/Desktop/prj2 $ sudo ./testing 1703 2
casper value 2, pid 1703
syscall result:0
 Success
0 ✓ itucs@ubuntu ~/Desktop/prj2 $ ls /proc/659
ls: cannot access /proc/659: No such file or directory
2 ✗ itucs@ubuntu ~/Desktop/prj2 $ ls /proc/1703
attr                 cpuset   latency    mountstats     proji
autogroup            cwd      limits     net            root
auxv                 environ  loginuid   ns             schec
cgroup               exe      map_files  oom_adj        schec
clear_refs           fd       maps       oom_score      sessi
cmdline              fdinfo   mem        oom_score_adj  setgr
comm                 gid_map  mountinfo  pagemap        smaps
coredump_filter      io       mounts     personality    stack
0 ✓ itucs@ubuntu ~/Desktop/prj2 $
```

***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.


itucs@ubuntu: ~/Desktop/prj2
itucs@ubuntu: ~/Desktop/prj2 80x

```
0 ✓ itucs@ubuntu ~/Desktop/prj2 $ sudo ./testing 1 3
casper value 3, pid 1
syscall result:0
 Success
0 ✓ itucs@ubuntu ~/Desktop/prj2 $ sudo ./testing 659 3
casper value 3, pid 659
syscall result:0
 Success
0 ✓ itucs@ubuntu ~/Desktop/prj2 $ sudo ./testing 1703 3
casper value 3, pid 1703
syscall result:0
 Success
0 ✓ itucs@ubuntu ~/Desktop/prj2 $ ls /proc/1
ls: cannot access /proc/1: No such file or directory
2 ✗ itucs@ubuntu ~/Desktop/prj2 $ ls /proc/659
ls: cannot access /proc/659: No such file or directory
2 ✗ itucs@ubuntu ~/Desktop/prj2 $ ls /proc/1703
ls: cannot access /proc/1703: No such file or directory
2 ✗ itucs@ubuntu ~/Desktop/prj2 $ sudo ls /proc/1703
ls: cannot access /proc/1703: No such file or directory
2 ✗ itucs@ubuntu ~/Desktop/prj2 $ sudo ls /proc/659
ls: cannot access /proc/659: No such file or directory
2 ✗ itucs@ubuntu ~/Desktop/prj2 $ sudo ls /proc/1
ls: cannot access /proc/1: No such file or directory
```

***Screenshot 12:*** *Testing for casper=3*