
**BLG-413E – System Programming**

**150130066** Duhan Cem Karagöz

**150150701** Yunus Güngör

**Homework 1**

13.10.2017

**Objective:**

Objective of the homework is to learn how to implement a system call to a linux based system and learn how to develop and change already existing system calls and functions.

**System:**

On this homework Lubuntu 14.04 with GCC 4.7 is used.

**Program:**

At this homework we based our code and thoughts on classes and practice session slides. For this homework we are assinged to develop a way to kill a processes children if a flag is set to one and priority of that process is higher than 30. So we added a flag value to task descriptor and check priority whenever process exits.

```
1464    #if defined(CONFIG_BCACHE) || defined(CONFIG_BCACHE_MODULE)
1465        unsigned int    sequential_io;
1466        unsigned int    sequential_io_avg;
1467    #endif
1468        int myFlag;
1469    };
```

Figure 1. sched.h

To be able to create a flag and change it first of all we added a flag to end of the task descriptor. We needed to add out flag to end of the descriptor. We could not add the flag in between since it would create misallignment issues with the jump operation in machine code.

```
212         [PIDTYPE_PID]  = INIT_PID_LINK(PIDTYPE_PID),        \
213         [PIDTYPE_PGID] = INIT_PID_LINK(PIDTYPE_PGID),       \
214         [PIDTYPE_SID]  = INIT_PID_LINK(PIDTYPE_SID),        \
215     },                                  \
216     .thread_group   = LIST_HEAD_INIT(tsk.thread_group),     \
217     .thread_node    = LIST_HEAD_INIT(init_signals.thread_head), \
218     .myFlag=0,  \
219     INIT_IDS                                \
220     INIT_PERF_EVENTS(tsk)                       \
221     INIT_TRACE_IRQFLAGS                     \
222     INIT_LOCKDEP                            \
223     INIT_FTRACE_GRAPH                       \
224     INIT_TRACE_RECURSION                        \
225     INIT_TASK_RCU_PREEMPT(tsk)                  \
226     INIT_CPUSET_SEQ(tsk)                        \
227     INIT_VTIME(tsk)                         \
228 }
```

Figure 2. init_task.h

When first process created it will be allocated staticly and its tast descriptor is taken from init_task.h therefore we need to initialaze its flag as zero at the file.

```
asmlinkage long set_myFlag(pid_t pid, int flag){

  printk("set my flag starts running with: %d, %d \n",pid,flag);

  struct task_struct *task;
  task=find_task_by_vpid(pid);

  if (task>0)
  {
     printk("found task:  %d  \n", task->pid);
    if(current_cred()>0){
      if(current_cred()->uid <= 0 || current_cred()->gid<=0){ //https://www.kernel.org/doc/Documentation/security/credentials.txt
         //Process has root priv. check flag
        if(flag==1 || flag==0){
          task->myFlag=flag;
          printk("my_flag value of process %d has been set to %d \n",pid,task->myFlag);
          //https://www.ibm.com/developerworks/library/l-kernel-logging-apis/index.html
          return 0;
        }
        else{
          printk("my_flag value of process %d can not set to %d \n",pid,flag);
          return -EINVAL;
        }

        }
        else{
          //Not Root throw error
          printk("my_flag value of process %d can not set to %d \n",pid,flag);
         //https://www.ibm.com/developerworks/library/l-kernel-logging-apis/index.html
          return -EPERM; // http://man7.org/linux/man-pages/man2/syscalls.2.html
        }
    }
    else{
      printk("user error \n");
      return -EACCES;
    }
  }
  else{
    printk("no process found \n");
    return -ESRCH;
  }
}

```

Figure 3. set_myFlag.c

When a process needs to change its flag value it needs to call this funtion with a pid number but calling process must have root privilages. At the first if it cheks if pid is valid than checks if calling process have root privilages than it will assign the new flag value to process. If a check is failed than corresponding error will be returned to the calling process.

```
847   asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
848                unsigned long idx1, unsigned long idx2);
849   asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
850   asmlinkage long sys_seccomp(unsigned int op, unsigned int flags,
851                   const char __user *uargs);
852   asmlinkage long set_myFlag(pid_t pid, int flag);
853   #endif
854
```

Figure 4. sys_calls.c

We added a prototype of our call to sys_calls to be recognized and compiled.

```
350 i386    finit_module        sys_finit_module
351 i386    sched_setattr       sys_ni_syscall
352 i386    sched_getattr       sys_ni_syscall
353 i386    renameat2       sys_ni_syscall
354 i386    seccomp         sys_seccomp
355 i386    set_myFlag  set_myFlag
```

Figure 5. syscall_32.tbl

To use our new system call we added the system call to syscall_32.tbl.

```
    atomic_set(&tsk->usage, 2);
#ifdef CONFIG_BLK_DEV_IO_TRACE
    tsk->btrace_seq = 0;
#endif
    tsk->splice_pipe = NULL;
    tsk->task_frag.page = NULL;

    account_kernel_stack(ti, 1);

    tsk->myFlag = 0 ;

    return tsk;

free_ti:
    free_thread_info(ti);
free_tsk:
    free_task_struct(tsk);
    return NULL;
}
```

Figure 6. fork.c

When a new process is forked child process copies its task from its parent. So at every fork we set flag that copied to zero to prevent any mishaps from happening.

```
    struct task_struct *tsk = current;
    int group_dead;

    struct task_struct *task;
    struct list_head *list;
    int pids;
    int priory;
    int tsk_flag=tsk->myFlag;
    priory=sys_getpriority(PRIO_PROCESS, tsk->pid);

    if(tsk_flag == 1 && priory>10)
    {
            printk("Parents with pid pid:%d proi is %d.\n",tsk->pid, priory);
            printk("Parent with pid:%d has been terminated Killing offsprings.\n",tsk->pid);
            //get all childs pid and call them to exit
            list_for_each(list, &current->children)
            {
                task = list_entry(list, struct task_struct, sibling);
                pids = task->pid;
                printk("Child with pid:%d has been terminated \n",task->pid);
                sys_kill(task->pid, SIGKILL);
                // task points to a children
            }
        //Linux Kernel Development - 3rd Edition Page 30
    }
```

Figure 7. exit.c

To be able to kill child processes with parent we altered the exit.c Every time a process exits it calls exit() function that in turn calls do_exit() so we added a code linet o first part of do_exit() evertime a process exits it will go through this lines of code. If a process arrives with flag 1 and priority above 10 it will go into the if statement. Since task descriptors are connected via double link list it is easy find child processes. With the list_for_each macro it will find each child of the current process and kill it.

```
#define NR_mycall 355
#include <unistd.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <stdlib.h>

int main (void){
     int y;
     int pid;
     int which=PRIO_PROCESS;
     id_t pids=getpid();
    int secim=0;
    printf("1-> Flag With 0 And Prio 0\n");
    printf("2-> Flag With 1 And Prio 0\n");
    printf("3-> Flag With 0 And Prio 33\n");
    printf("4-> Flag With 1 And Prio 33\n");
    printf("5-> Flag With 2 And Prio 33\n");

```

Figure 8. test_set_myValue.c Part 1.

To test our changes in the kernel we wrote a piece of code to check if our implementation is worked correctly. First part is to select one of five different scenario to check our code. First you can check if code is not working when it should not with not setting anything. Second scenario is to check if our priority checking procedures are working. Third is for flag checking. Fourth one is to check if when everything is set is our program working accordingly. Meaning is our program killing childs when it suppose to kill.

```
22      switch(secim)
23      {
24      case 1:
25      //do not set both up leave them down
26      break;
27
28      case 2:
29      y=syscall(NR_mycall, pids, 1);
30      //set flag one but leave prio low
31      break;
32
33      case 3:
34      setpriority(which, pids, -15);
35      //set prio high but flag 0
36      break;
37
38      case 4:
39      y=syscall(NR_mycall, pids, 1);
40      setpriority(which, pids, -15);
41      //set both up
42      break;
43
44      case 5:
45      y=syscall(NR_mycall, pids, 2);
46      setpriority(which, pids, -15);
47      //set both up
48      break;
49
50      default:
51      return 1;
52      }
```

Figure 9. test_set_myValue.c Part 2.

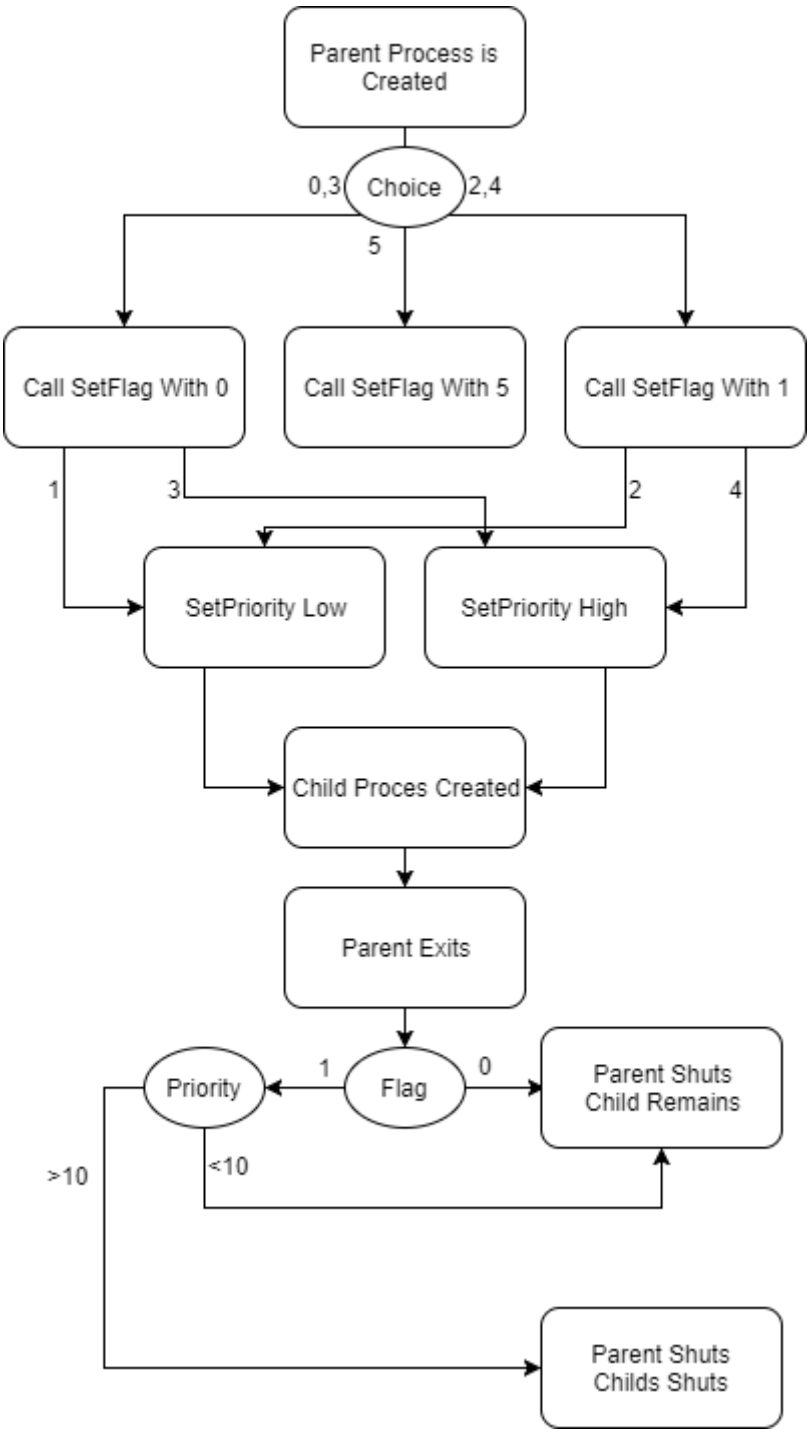
This part shows how we set flag and proirities.

```
54      pid=fork();
55
56      if(pid==0)
57      {
58          printf("I am the child\n");
59          printf("my pid=%d\n", getpid());
60              while(1)
61              {
62                  //empty must wait father to exit
63                  //or CTRL+C to stop
64              }
65
66          exit(0);
67      }
68
69      else
70      {
71          int waits;
72          printf("I am the father\n");
73          printf("my pid=%d\n", getpid());
74          printf("My Prio %d\n", getpriority(which, getpid()));
75          //sleep(1);
76           scanf("%d", &waits);
77           exit(0);
78          //return 1;
79      }
80
```

Figure 10. test_set_myValue.c Part 3.

This part is where we forked the process and put child process into infinite loop to see if it exits or not with the parent process.

**Flow Diagram:**




**Results:**

We learned how to implement a system call and change already existing ones. Also we learned how to add a value to task descriptor. And we learned how to delevop and improve existing kernels.