PRVI DOMAĆI ZADATAK - MODIFIKACIJA I REBUILD-OVANJE LINUX KERNELA

UROŠ VUKIĆ

CILJ ZADATKA

- Za proces sa zadatim PID prikazati sve signale koje je proces primio i na koje je odgovorio izvršavanjem odgovarajućeg handler-a.

ŠTA SU SIGNALI?

- Veoma kratke poruke koje se mogu poslati procesu ili grupi procesa
- Jedina informacija koja se šalje je uglavnom broj koji identifikuje signal

Brojevi prva tri signala:

#	Signal name	Default action	Comment	POSIX
1	SIGHUP	Terminate	Hang up controlling terminal or process	Yes
2	SIGINT	Terminate	Interrupt from keyboard	Yes
3	SIGQUIT	Dump	Quit from keyboard	Yes

ŠTA SU SIGNALI?

Postoje dve faze u prenošenju signala:

1. Generisanje signala – krenel update-uje (ažurira) u odredišnom procesu (proces kome se šalje signal) strukture koje predstavljaju da je poslat novi signal
2. Dostavljanje signala – kernel primorava odredišni proces da „odreaguje“ na signal tako što će promeniti svoje stanje, izvršiti odgovarajući handler ili odraditi obe stvari

STRUKTURE PODATAKA POVEZANE SA SIGNALIMA

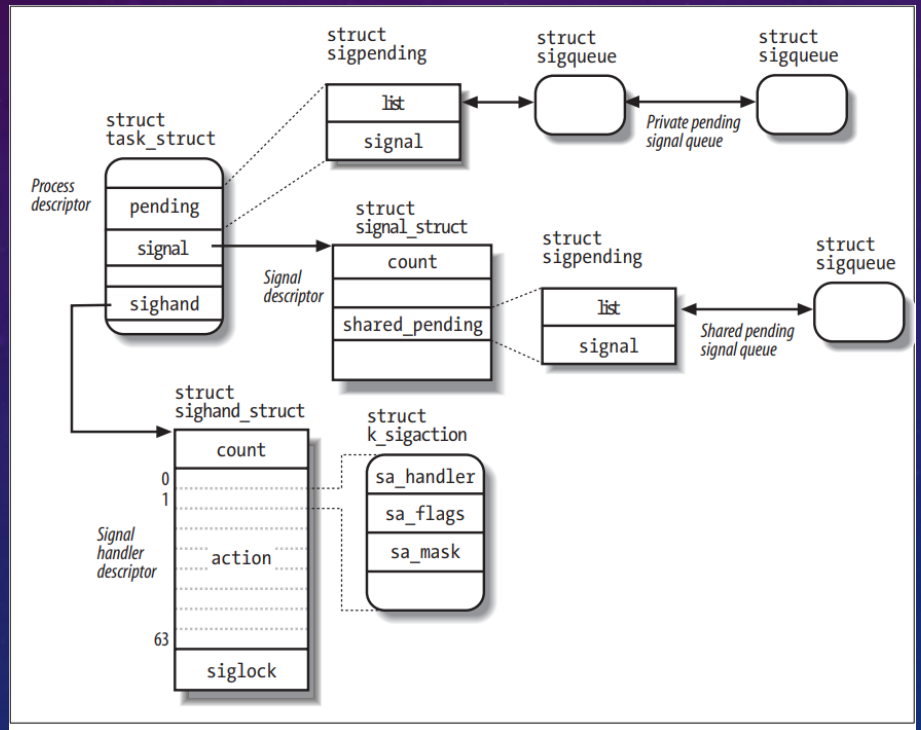


Figure 11-1. The most significant data structures related to signal handling

Preuzeto iz knjige:
Understanding Linux Kernel

MODIFIKACIJE: STRUKTURA RECEIVED_SIGNAL

Struktura koja čuva podatke o primljenom signalu:

- int sig_num – čuva tip/broj signala
- int handled – čuva da li je signal obradjen (0 – nije, 1 – jeste)
- struct list_head list – pokazivači na susedne elemente u listi

Nalazi se u file-u include/linux/signal_types.h

Unutar task_struct strukture, dodat je čvor rec_sig koji pokazuje na elemente ove strukture

```
71  #define SIGNAL_HANDLED 1
72  #define SIGNAL_NOT_HANDLED 0
73
74  struct received_signal {
75      int sig_num;
76      int handled;
77      struct list_head list;
78  };
79
```

```
1204
1205  struct list_head rec_sig;
1206
      Studio Code
1208      /*
1209       * New fields for task_struct should be added above here, so that
1210       * they are included in the randomized portion of task_struct.
1211       */
          randomized_struct_fields_end
```

MODIFIKACIJE: IZMENE U SIGNAL.C FILE-U

- U ovom fajlu se nalazi kompletna logika za implementaciju funkcionalnosti signala
- add_received_signal – funkcija koja dodaje novi čvor u listi primljenih signala unutar task_struct strukture. Poziva se u unutar __send_signal funkcije. Send signal funkcija implementira slanje signala nekom procesu tako sto dodaje novi element sigqueue strutkure u task_struct procesa.

```
1075  static void add_received_signal(int sig, struct task_struct *t)
1076  {
1077      struct received_signal *rs;
1078      rs = kmalloc(sizeof(*rs), GFP_KERNEL);
1079      if(t == NULL || rs == NULL)
1080          return;
1081      rs->sig_num = sig;
1082      rs->handled = SIGNAL_NOT_HANDLED;
1083      list_add(&rs->list, &t->rec_sig);
1084  }
1085
```

```
1132
1133      q = __sigqueue_alloc(sig, t, GFP_ATOMIC, override_rlimit);
1134      if (q) {
1135          list_add_tail(&q->list, &pending->list);
1136          add_received_signal(sig, t);
1137          switch ((unsigned long) info) {
1138          case (unsigned long) SEND_SIG_NOINFO:
1139              clear_siginfo(&q->info);
1140              q->info.si_signo = sig;
```

MODIFIKACIJE: IZMENE U SIGNAL.C FILE-U

- change_received_signal_status – funkcija koja menja status prve pojave čvora neobrađenog signal iz stanja ne obrađen u stanje obrađen. Poziva se unutar signal_delivered funkcije, koja se poziva nakon što je određeni signal uspešno obrađen.

```
2596
2597  static void change_received_signal_status(int sig)
2598  {
2599      struct received_signal *rs;
2600      list_for_each_entry(rs, &current->rec_sig, list)
2601      {
2602          if (rs->sig_num == sig && rs->handled != SIGNAL_HANDLED)
2603          {
2604              rs->handled = SIGNAL_HANDLED;
2605              return;
2606          }
2607      }
2608  }
```

```
2620  static void signal_delivered(struct ksignal *ksig, int stepping)
2621  {
2622      sigset_t blocked;
2623
2624      /* A signal was successfully delivered, and the
2625         saved sigmask was stored on the signal frame,
2626         and will be restored by sigreturn.  So we can
2627         simply clear the restore sigmask flag.  */
2628      clear_restore_sigmask();
2629
2630      sigorsets(&blocked, &current->blocked, &ksig->ka.sa.sa_mask);
2631      if (!(ksig->ka.sa.sa_flags & SA_NODEFER))
2632          sigaddset(&blocked, ksig->sig);
2633      set_current_blocked(&blocked);
2634      tracehook_signal_handler(stepping);
2635      change_received_signal_status(ksig->sig);
2636  }
```

DODATNE MODIFIKACIJE: FORK.C FILE

- U ovom file-u se nalaze funkcije koje se pozivaju prilikom kopiranja task_struct strukture kao i funkcije za brisanje task_struct.
- Unutak poziva copy_process funkcije (funkcija koja vrši kopiranje task_struct) dodat je poziv funkcije INIT_LIST_HEAD za inicijalizaciju liste koja čuva primljene signale.
- free_task_struct – funkcija koja briše received_signal čvorove iz liste prilikom brisanja task_struct

```
2127        uprobe_copy_process(p, clone_flags);
2128
2129        // Init signal list
2130        INIT_LIST_HEAD(&p->rec_sig);
2131
2132        return p;
2133
2134    bad_fork_cancel_cgroup:
```

```
171
172   static inline void free_task_struct(struct task_struct *tsk)
173   {
174       struct received_signal *p;
175       struct list_head *pos, *next;
176       list_for_each_safe(pos, next, &tsk->rec_sig)
177       {
178           p = list_entry(pos, struct received_signal, list);
179           list_del(pos);
180           kfree(p);
181       }
182
183       kmem_cache_free(task_struct_cachep, tsk);
184   }
```

SISTEMSKI POZIV ZA PRIKAZ LISTE SIGNALA

- Sistemski poziv ima samo jedan parametar – pid procesa za koji prikazuje listu. Svodi se na obilazaka liste i štampanje njenog sadržaja.

```
18
19  SYSCALL_DEFINE1(print_signals, pid_t, pid)
20  {
21      struct task_struct *p;
22      printk("SYSCALL print signals for porcess pid: %d\n", pid);
23      p = find_task_by_vpid(pid);
24      if (p != NULL)
25      {
26          print_sig(p);
27      }
28      else
29      {
30          printk("Unable to find process with pid: %d\n", pid);
31      }
32      printk("\n");
33      return 0;
34  }
```

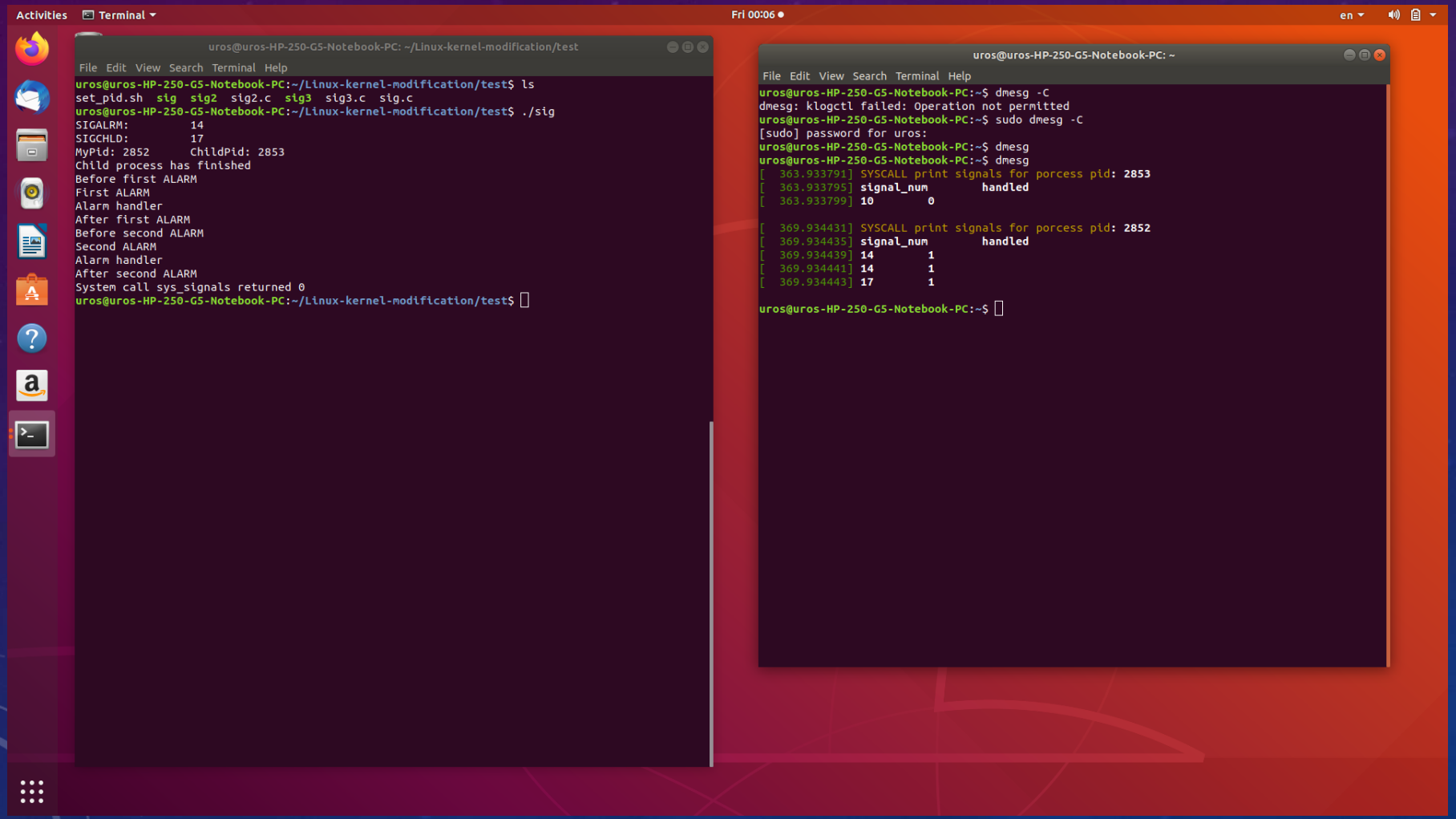
```
 8
 9  static void print_sig(struct task_struct *t)
10  {
11      struct received_signal *rs;
12      printk("signal_num\t handled\n");
13      list_for_each_entry(rs, &t->rec_sig, list)
14      {
15          printk("%d\t %d\n", rs->sig_num, rs->handled);
16      }
17  }
18
```

DEMO PROGRAM

```c
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>

#define SEC 3
// SIGALRM:      14
// SIGCHLD:      17


void echo_msg()
{
    printf("Alarm handler\n");
}

void smt()
{
    sleep(0);
    return;
}

void sleep_hand()
{
    sleep(SEC);
    exit(0);
}


void process_finished()
{
    printf("Child process has finished\n");
}
```

```c
int main()
{
    int pid;
    int mypid;
    long int amma;
    printf("SIGALRM: \t %d\n", SIGALRM);
    printf("SIGCHLD: \t %d\n", SIGCHLD);

    pid = fork();
    if (pid == 0)
    {
        signal(SIGALRM, smt);
        signal(SIGUSR1, smt);
        alarm(1);
        pause();
        amma = syscall(336, getpid());
    }
    else
    {
        int status, wpid;
        mypid = getpid();
        printf("MyPid: %d \t ChildPid: %d\n", mypid, pid);
        signal(SIGALRM, echo_msg);
        signal(SIGCHLD, process_finished);

        kill(pid, SIGUSR1);

        amma = syscall(336, pid);
        wpid = wait(&status);

        printf("Before first ALARM\n");
        alarm(SEC);
        printf("First ALARM\n");
        pause();
        printf("After first ALARM\n");


        printf("Before second ALARM\n");
        alarm(SEC);
        printf("Second ALARM\n");
        pause();
        printf("After second ALARM\n");

        amma = syscall(336, mypid);
        printf("System call sys_signals returned %ld\n", amma);

        signal(SIGALRM, SIG_DFL);
        signal(SIGCHLD, SIG_DFL);
    }
    return 0;
}
```

REZULTAT


Activities
Terminal
Fri 00:06
en
uros@uros-HP-250-G5-Notebook-PC: ~/Linux-kernel-modification/test
File Edit View Search Terminal Help
uros@uros-HP-250-G5-Notebook-PC:~/Linux-kernel-modification/test$ ls
set_pid.sh sig sig2 sig2.c sig3 sig3.c sig.c
uros@uros-HP-250-G5-Notebook-PC:~/Linux-kernel-modification/test$ ./sig
SIGALRM: 14
SIGCHLD: 17
MyPid: 2852 ChildPid: 2853
Child process has finished
Before first ALARM
First ALARM
Alarm handler
After first ALARM
Before second ALARM
Second ALARM
Alarm handler
After second ALARM
System call sys_signals returned 0
uros@uros-HP-250-G5-Notebook-PC:~/Linux-kernel-modification/test$
uros@uros-HP-250-G5-Notebook-PC: ~
File Edit View Search Terminal Help
uros@uros-HP-250-G5-Notebook-PC:~$ dmesg -C
dmesg: klogctl failed: Operation not permitted
uros@uros-HP-250-G5-Notebook-PC:~$ sudo dmesg -C
[sudo] password for uros:
uros@uros-HP-250-G5-Notebook-PC:~$ dmesg
uros@uros-HP-250-G5-Notebook-PC:~$ dmesg
[363.933791] SYSCALL print signals for porcess pid: 2853
[363.933795] signal_num handled
[363.933799] 10 0
[369.934431] SYSCALL print signals for porcess pid: 2852
[369.934435] signal_num handled
[369.934439] 14 1
[369.934441] 14 1
[369.934443] 17 1
uros@uros-HP-250-G5-Notebook-PC:~$

IMPLEMENTACIJA MODULA

Modul je implementiran kao device kome se može pristupiti preko file sistema. Pristup je isti kao i pristup proc file sistemu. Implementira se interfejs zadat file_operations strukturom

- owner – vlasnik file-a
- read – pointer na funkciju za čitanje file-a
- write – pointer na funkciju za upis u file

```
118
119  static const struct file_operations file_ops = {
120      .owner = THIS_MODULE,
121      .read = procfile_read,
122      .write = proc_write,
123  };
124
```

Prilikom inicijalizacije modula potrebno je pozvati proc_create funkciju za kreiranje proc file-a, pri čemu se prosleđuje file_operations struktura

Za uklanjanje file-a potrebno je pozvati proc_remove funkciju

IMPLEMENTACIJA MODULA

- Implementirani modul ima jedan parametar – pid kojim mu se prosleđuje pid procesa za koji treba prikazati primljene signale
- Ukoliko se pri inicijalizaciji prosledi pid parametar, onda se vrši prikaz liste unutar kernel log-a

```
136  static int __init init_signal_module(void)
137  {
138      module_proc_file = proc_create(PROCFS_NAME, 0664, NULL, &file_ops);
139      if (module_proc_file == NULL)
140      {
141          proc_remove(module_proc_file);
142          printk(KERN_ALERT "Error: Could not initialize /proc/%s\n", PROCFS_NAME);
143          return -ENOMEM;
144      }
145
146      if (pid == 0)
147      {
148          printk("Process PID not passed.\nInitialization finished\n");
149          return 0;
150      }
151      p = pid_task(find_vpid(pid), PIDTYPE_PID);
152      if (p == NULL)
153      {
154          printk("Process with PID: %d not found\nnInitialization finished\n", pid);
155          return 0;
156      }
157      print_sig(p);
158      return 0;
159  }
```

DEMO PROGRAM

```c
void print_signals(int pid)
{
    int fd;
    char buffer[BUFF_LEN];
    int read_count, write_count;
    int len;
    fd = open(MODULE_LOCATION, O_WRONLY);

    if(fd < 0)
    {
        printf("Error opening pid file descriptor\n");
        printf("error code : %d\n", fd);
        return;
    }

    sprintf(buffer, "%d", pid);
    len = strlen(buffer);

    write_count = write(fd, buffer, len);
    if (write_count > 0)
    {
        printf("Wrote to pid -> %s\n", buffer);
    }
    else
    {
        printf("Couldnt print to pid location\n");
        return;
    }
    close(fd);

    fd = open(MODULE_LOCATION, O_RDONLY);
    if(fd < 0)
    {
        printf("Error opening module file descriptor\n");
    }

    read_count = read(fd, buffer, BUFF_LEN);
    while(read_count)
    {
        printf("%.*s", read_count, buffer);
        printf("\n");
        read_count = read(fd, buffer, BUFF_LEN);
    }
    close(fd);
}
```

```c
int main()
{
    int pid;
    int mypid;
    long int amma;
    printf("SIGALRM: \t %d\n", SIGALRM);
    printf("SIGCHLD: \t %d\n", SIGCHLD);

    pid = fork();
    if (pid == 0)
    {
        signal(SIGALRM, smt);
        signal(SIGUSR1, smt);
        alarm(1);
        pause();
        // amma = syscall(336, getpid());
    }
    else
    {
        int status, wpid;
        mypid = getpid();
        printf("MyPid: %d \t ChildPid: %d\n", mypid, pid);
        signal(SIGALRM, echo_msg);
        signal(SIGCHLD, process_finished);

        kill(pid, SIGUSR1);

        // amma = syscall(336, pid);
        wpid = wait(NULL);

        printf("Before first ALARM\n");
        alarm(SEC);
        printf("First ALARM\n");
        pause();
        printf("After first ALARM\n");


        printf("Before second ALARM\n");
        alarm(SEC);
        printf("Second ALARM\n");
        pause();
        printf("After second ALARM\n");

        // amma = syscall(336, mypid);
        print_signals(mypid);

        signal(SIGALRM, SIG_DFL);
        signal(SIGCHLD, SIG_DFL);
    }

    return 0;
}
```

REZULTAT

```
uros@uros-HP-250-G5-Notebook-PC: ~/Linux-kernel-modification/test
File Edit View Search Terminal Help
uros@uros-HP-250-G5-Notebook-PC:~/Linux-kernel-modification/test$ sudo ./sig3
SIGALRM:        14
SIGCHLD:        17
MyPid: 2865     ChildPid: 2866
Child process has finished
Before first ALARM
First ALARM
Alarm handler
After first ALARM
Before second ALARM
Second ALARM
Alarm handler
After second ALARM
Wrote to pid -> 2865
SIG     HANDLED

14      1

14      1

17      1

uros@uros-HP-250-G5-Notebook-PC:~/Linux-kernel-modification/test$
```

```
uros@uros-HP-250-G5-Notebook-PC:~/Linux-kernel-modification/test$ cat set_pid.sh
sudo sh -c 'echo '$1' > /proc/sig_mod'

uros@uros-HP-250-G5-Notebook-PC:~/Linux-kernel-modification/test$ sh set_pid.sh  1
sh: echo: I/O error
uros@uros-HP-250-G5-Notebook-PC:~/Linux-kernel-modification/test$ cat /proc/sig_mod
SIG     HANDLED
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
55      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
17      0
54      0
17      0
```

HVALA NA PAŽNJI!