ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
1/12/2024

Δραστηριότητα 3: Linux και Δημιουργία Kernel Module

ΑΣΚΗΣΗ 1

Στον φάκελο */root/hello-world* αποθηκέυουμε το *hello.c*

```c
#include <linux/kernel.h>
#include <linux/module.h>

MODULE_DESCRIPTION("My kernel module");
MODULE_AUTHOR("Me");
MODULE_LICENSE("GPL");

static int my_init(void){
        printk("Message from kernel module\n");
        return 0;
}

static void my_exit(void){
        printk("Bye from kernel module\n");
}

module_init(my_init);
module_exit(my_exit);
```

Αποθηκεύουμε και το Makefile που περιέχει τις εντολές για να μεταγλωττιστεί το module που φτιάχνουμε.

```
obj-m += hello.o

PWD := $(CURDIR)

all:
        make -C/lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
        make -C/lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Μεταγλωττίζουμε το module με την εντολή make

```
root@nataliarouska:~/hello-world# make
make -C/lib/modules/6.1.0-28-amd64/build M=/root/hello-world modules
make[1]: Entering directory '/usr/src/linux-headers-6.1.0-28-amd64'
make[1]: Leaving directory '/usr/src/linux-headers-6.1.0-28-amd64'
root@nataliarouska:~/hello-world# ls
hello.c   hello.mod    hello.mod.o  Makefile       Module.symvers
hello.ko  hello.mod.c  hello.o      modules.order
```

Το *hello.ko* είναι το kernel object που θα φορτώσουμε στο kernel με την εντολή *insmod hello.ko*

```
root@nataliarouska:~/hello-world# insmod hello.ko
root@nataliarouska:~/hello-world# lsmod | grep hello
hello                  16384  0
```

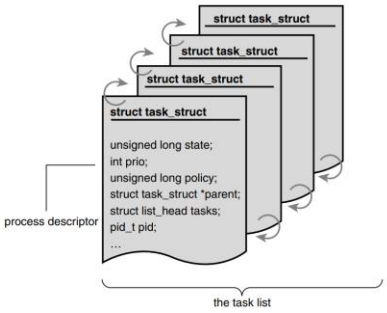
Με την εντολή *rmmod hello.ko* αφαιρούμε το module. Επίσης δεν έχει δυνατότητα εκτύπωσης των μηνυμάτων σε γραφική κονσόλα, οπότε βλέπουμε τα μηνύματα από τα logs του συστήματος με την εντολή *journalctl –since "5 minutes ago" | grep kernel*

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
1/12/2024

```
Dec 01 17:39:46 nataliarouska kernel: Message from kernel module
Dec 01 17:39:53 nataliarouska kernel: Bye from kernel module
```

ΑΣΚΗΣΗ 2

Ο kernel του linux αποθηκεύει τις διεργασίες σε μία διπλά διασυνδεδεμένη λίστα που λέγεται task list. Κάθε στοιχείο της λίστας είναι μία δομή *task_struct* που ορίζεται στο αρχείο *linux/include/linux/sched.h* και είναι ένας process descriptor. Περιέχει όλες τις πληροφορίες για μία συγκεκριμένη διεργασία (e.g. state,priority,resources). Αντίθετα με τη δομή *proc* στο xv6, η δομή *task_struct* έχει περισσότερα πεδία και καταλαμβάνει 1.7 kilobytes σε 32-bit σύστημα.




```
struct task_struct {
#ifdef CONFIG_THREAD_INFO_IN_TASK
        /*
         * For reasons of header soup (see current_thread_info()), this
         * must be the first element of task_struct.
         */
        struct thread_info              thread_info;
#endif
        unsigned int                    __state;

        /* saved state for "spinlock sleepers" */
        unsigned int                    saved_state;

        /*
         * This begins the randomizable portion of task_struct. Only
         * scheduling-critical items should be added above here.
         */
        randomized_struct_fields_start

        void                            *stack;
        refcount_t                      usage;
        /* Per task flags (PF_*), defined further below: */
        unsigned int                    flags;
        unsigned int                    ptrace;
```

...

Figure 1: Linux Kernel Development Chapter 3

Παρακάτω αναλύονται μερικά βασικά πεδία της δομής *task_struct*

- *pid_t pid*: The process ID.
- *pid_t tgid*: Thread group ID (same as pid for the main thread).
- *char comm[TASK_COMM_LEN]*: Name of the executable.
- *state*: Current state of the process (e.g., TASK_RUNNING, TASK_INTERRUPTIBLE).
- *exit_state*: State after the process has terminated (e.g., EXIT_ZOMBIE, EXIT_DEAD).

```
/* Used in tsk->__state: */
#define TASK_RUNNING                    0x00000000
#define TASK_INTERRUPTIBLE              0x00000001
#define TASK_UNINTERRUPTIBLE            0x00000002
#define __TASK_STOPPED                  0x00000004
#define __TASK_TRACED                   0x00000008
/* Used in tsk->exit_state: */
#define EXIT_DEAD                       0x00000010
#define EXIT_ZOMBIE                     0x00000020
#define EXIT_TRACE                      (EXIT_ZOMBIE | EXIT_DEAD)
/* Used in tsk->__state again: */
#define TASK_PARKED                     0x00000040
#define TASK_DEAD                       0x00000080
#define TASK_WAKEKILL                   0x00000100
#define TASK_WAKING                     0x00000200
#define TASK_NOLOAD                     0x00000400
#define TASK_NEW                        0x00000800
#define TASK_RTLOCK_WAIT                0x00001000
#define TASK_FREEZABLE                  0x00002000
#define __TASK_FREEZABLE_UNSAFE         (0x00004000 * IS_ENABLED(CONFIG_LOCKDEP))
#define TASK_FROZEN                     0x00008000
#define TASK_STATE_MAX                  0x00010000
```

- *int prio*: Process priority (used by the scheduler to determine the order for CPU occupation)
- *struct task_struct __rcu *parent*: Pointer to the parent process of the current process
- *struct mm_struct *mm*: Pointer to (mm_struct) for the virtual memory of the process

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
1/12/2024

- *struct files_struct *files:* Open files information (pointer to the file descriptor)
- *struct signal_struct *signal*: Singal handling in a thread group (threads share the signal_struct)

ΑΣΚΗΣΗ 3

Στον φάκελο */root/list-processes* αποθηκέυουμε το *list-processes.c*

```c
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/sched/signal.h>
MODULE_DESCRIPTION("List current processes");
MODULE_AUTHOR("Me");
MODULE_LICENSE("GPL");
static int my_proc_init(void)
{
struct task_struct *p; /* Needed for later **/
printk("Current process: pid = %d; name = %s\n",
current ->pid , current ->comm);
printk("\nProcess list:\n\n");
for_each_process(p){
  printk("pid = %d; name = %s\n",p->pid , p->comm); }
return 0;
}
static void my_proc_exit(void)
{
printk("Current process: pid = %d; name = %s\n",
current ->pid , current ->comm);
}
module_init(my_proc_init);
module_exit(my_proc_exit);
```

Στο αρχείο *linux/arch/x86/include/asm/current.h* υπάρχει το macro current που επιστρέφει έναν δείκτη στη δομή task_struct που εκτελείται εκείνη τη στιγμή.

```c
static __always_inline struct task_struct *get_current(void)
{
        if (IS_ENABLED(CONFIG_USE_X86_SEG_SUPPORT))
                return this_cpu_read_const(const_pcpu_hot.current_task);

        return this_cpu_read_stable(pcpu_hot.current_task);
}

#define current get_current()
```

Στο αρχείο *linux/include/linux/sched/signal.h* υπάρχει το macro *for_each_process(p)*, for loop που διατρέχει την την κυκλικά διασυνδεδεμένη λίστα task list.

```c
#define for_each_process(p) \
        for (p = &init_task ; (p = next_task(p)) != &init_task ; )
```

Αποθηκεύουμε και το Makefile που περιέχει τις εντολές για να μεταγλωττιστεί το module που φτιάχνουμε.

```
obj-m += list-processes.o

PWD := $(CURDIR)

all:
        make -C/lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
        make -C/lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
1/12/2024

Μεταγλωττίζουμε το module με την εντολή make

```
root@nataliarouska:~/list-processes# ls
list-processes.c    list-processes.mod.c  Makefile
list-processes.ko   list-processes.mod.o  modules.order
list-processes.mod  list-processes.o      Module.symvers
root@nataliarouska:~/list-processes# make
make -C/lib/modules/6.1.0-28-amd64/build M=/root/list-processes modules
make[1]: Entering directory '/usr/src/linux-headers-6.1.0-28-amd64'
make[1]: Leaving directory '/usr/src/linux-headers-6.1.0-28-amd64'
```

Φορτώνουμε το module με την εντολή insmod list-processes.ko και το αφαιρούμε με την εντολή rmmod list-processes.ko. Βλέπουμε τα μηνύματα από τα logs του συστήματος, όπου τυπώνεται η λίστα με όλες τις υπάρχουσες διεργασίες του συστήματος τη στιγμή που φορτώνεται το module.

```
root@nataliarouska:~/list-processes# insmod list-processes.ko
root@nataliarouska:~/list-processes# rmmod list-processes.ko
root@nataliarouska:~/list-processes# journalctl --since "1 minutes ago" | grep kernel
Dec 07 12:11:27 nataliarouska kernel: Current process: pid = 3855; name = insmod
Dec 07 12:11:27 nataliarouska kernel: Process list:
Dec 07 12:11:27 nataliarouska kernel: pid = 1; name = systemd
Dec 07 12:11:27 nataliarouska kernel: pid = 2; name = kthreadd
Dec 07 12:11:27 nataliarouska kernel: pid = 3; name = rcu_gp
Dec 07 12:11:27 nataliarouska kernel: pid = 4; name = rcu_par_gp
Dec 07 12:11:27 nataliarouska kernel: pid = 5; name = slub_flushwq
Dec 07 12:11:27 nataliarouska kernel: pid = 6; name = netns
Dec 07 12:11:27 nataliarouska kernel: pid = 8; name = kworker/0:0H
Dec 07 12:11:27 nataliarouska kernel: pid = 9; name = kworker/u2:0
Dec 07 12:11:27 nataliarouska kernel: pid = 10; name = mm_percpu_wq
Dec 07 12:11:27 nataliarouska kernel: pid = 11; name = rcu_tasks_kthre
Dec 07 12:11:27 nataliarouska kernel: pid = 12; name = rcu_tasks_rude_
Dec 07 12:11:27 nataliarouska kernel: pid = 13; name = rcu_tasks_trace
Dec 07 12:11:27 nataliarouska kernel: pid = 14; name = ksoftirqd/0
Dec 07 12:11:27 nataliarouska kernel: pid = 15; name = rcu_preempt
Dec 07 12:11:27 nataliarouska kernel: pid = 16; name = migration/0
Dec 07 12:11:27 nataliarouska kernel: pid = 18; name = cpuhp/0
Dec 07 12:11:27 nataliarouska kernel: pid = 20; name = kdevtmpfs
Dec 07 12:11:27 nataliarouska kernel: pid = 21; name = inet_frag_wq
Dec 07 12:11:27 nataliarouska kernel: pid = 22; name = kauditd
Dec 07 12:11:27 nataliarouska kernel: pid = 23; name = khungtaskd
Dec 07 12:11:27 nataliarouska kernel: pid = 24; name = oom_reaper
Dec 07 12:11:27 nataliarouska kernel: pid = 25; name = kworker/u2:1
Dec 07 12:11:27 nataliarouska kernel: pid = 27; name = writeback
Dec 07 12:11:27 nataliarouska kernel: pid = 28; name = kcompactd0
Dec 07 12:11:27 nataliarouska kernel: pid = 29; name = ksmd
Dec 07 12:11:27 nataliarouska kernel: pid = 30; name = khugepaged
Dec 07 12:11:27 nataliarouska kernel: pid = 31; name = kintegrityd
Dec 07 12:11:27 nataliarouska kernel: pid = 32; name = kblockd
Dec 07 12:11:27 nataliarouska kernel: pid = 33; name = blkcg_punt_bio
Dec 07 12:11:27 nataliarouska kernel: pid = 34; name = tpm_dev_wq
Dec 07 12:11:27 nataliarouska kernel: pid = 35; name = edac-poller
Dec 07 12:11:27 nataliarouska kernel: pid = 36; name = devfreq_wq
```

...

```
Dec 07 12:11:27 nataliarouska kernel: pid = 1857; name = tracker-miner-f
Dec 07 12:11:27 nataliarouska kernel: pid = 1871; name = VBoxClient
Dec 07 12:11:27 nataliarouska kernel: pid = 1873; name = VBoxClient
Dec 07 12:11:27 nataliarouska kernel: pid = 1896; name = VBoxClient
Dec 07 12:11:27 nataliarouska kernel: pid = 1897; name = VBoxClient
Dec 07 12:11:27 nataliarouska kernel: pid = 1906; name = fwupd
Dec 07 12:11:27 nataliarouska kernel: pid = 1944; name = gnome-calendar
Dec 07 12:11:27 nataliarouska kernel: pid = 1981; name = gvfsd-trash
Dec 07 12:11:27 nataliarouska kernel: pid = 2033; name = dconf-service
Dec 07 12:11:27 nataliarouska kernel: pid = 2164; name = gnome-terminal-
Dec 07 12:11:27 nataliarouska kernel: pid = 2326; name = bash
Dec 07 12:11:27 nataliarouska kernel: pid = 2398; name = sudo
Dec 07 12:11:27 nataliarouska kernel: pid = 2401; name = sudo
Dec 07 12:11:27 nataliarouska kernel: pid = 2402; name = bash
Dec 07 12:11:27 nataliarouska kernel: pid = 2412; name = gvfsd-metadata
Dec 07 12:11:27 nataliarouska kernel: pid = 3020; name = kworker/0:0
Dec 07 12:11:27 nataliarouska kernel: pid = 3030; name = kworker/0:1
Dec 07 12:11:27 nataliarouska kernel: pid = 3299; name = kworker/0:2
Dec 07 12:11:27 nataliarouska kernel: pid = 3312; name = kworker/u2:2
Dec 07 12:11:27 nataliarouska kernel: pid = 3855; name = insmod
Dec 07 12:11:31 nataliarouska kernel: Module exit
Dec 07 12:11:31 nataliarouska kernel: Current process: pid = 3857; name = rmmod
```

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
1/12/2024
ΑΣΚΗΣΗ 4

Δημιουργώ το πρόγραμμα *forking.c.* Προσθέτω μία καθυστέρηση 20sec, αφού δημιουργηθούν οι διεργασίες-παιδιά για να προλάβω να φορτώσω το module πριν τερματιστούν.

```c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#define SLEEP_TIME 1
int main(int argc , char *argv[])
{
int n = 0;
printf("PID: %ld\n", (long)getpid ());
/* Sleep for SLEEP_TIME seconds. */
sleep(SLEEP_TIME);
while (1) {
sleep(SLEEP_TIME);
/* break after a few iterations (too many processes) */

if (n++ > 3) {   //Όταν έχουμε 16 διεργασίες (με τον γονέα) το πρόγραμμα τερματίζει
   sleep(20);// καθυστέρηση για να προλάβω να φορτώσω το module
   exit(0);
}
// Η fork() επιστρέφει 0 όταν καλείται από διεργασία-παιδί,
// επιστρέφει το pid του παιδιού όταν καλείται απο διεργασία-γονέα
printf("Forked! PID: %d\n", fork());
}

return 0;
}
```

Στη δομή *task_struct* υπάρχουν τα πεδία *children και sibling*

```c
/*
 * Children/sibling form the list of natural children:
 */
struct list_head                children;
struct list_head                sibling;
```

- children: Αντιπροσωπεύει μια διπλά διασυνδεδεμένη λίστα με όλα τα παιδιά μιας διεργασίας, η οποία αποτελείται από δείκτες στις δομές task_struct που περιγράφουν κάθε παιδί.
- sibling: Χρησιμοποιείται για την σύνδεση των διεργασιών-παιδιών στη λίστα children της γονικής διεργασίας

Στο αρχείο *linux/tools/include/linux/list.h* υπάρχει το macro list_for_each_entry, με το οποίο μπορούμε να διατρέξουμε τα στοιχεία της λίστας children, παιδιά της γονικής διεργασίας.

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
1/12/2024

```c
/**
 * list_for_each_entry  -       iterate over list of given type
 * @pos:        the type * to use as a loop cursor.
 * @head:       the head for your list.
 * @member:     the name of the list_head within the struct.
 */
#define list_for_each_entry(pos, head, member)
        for (pos = list_first_entry(head, typeof(*pos), member);
             &pos->member != (head);
             pos = list_next_entry(pos, member))
```

Στο *root/list-children* δημιουργώ το αρχείο *list-children.c.* Η γονική διεργασία δημιουργεί με τη fork() μία διεργασία παιδί. Σε κάθε επανάληψη οι διεργασίες διπλασιάζονται, άρα τελικά θα έχουμε σύνολο 16 διεργασίες (με n=3). Το παιδί είναι ένα αντίγραφο του γονέα και εκτελεί τις εντολές του forking.c ακριβώς μετά την κλήση της fork(). Άρα, οι διεργασίες-παιδία δημιουργούν δικά τους παιδιά. Οπότε για να καταγράψουμε όλες τις διεργασίες πρέπει εκτός από τη λίστα children του γονέα, να διατρέξουμε και τη λίστα children των παιδιών κ.ο.κ . Αυτό γίνεται αναδρομικά στην print_children().

```c
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/sched/signal.h>
MODULE_DESCRIPTION("List forking children");
MODULE_AUTHOR("Me");
MODULE_LICENSE("GPL");

void print_children(struct task_struct *p);

static int my_proc_init(void){
  struct task_struct *p; /* Needed for later **/
  printk("Current process: pid = %d; name = %s\n", current ->pid , current ->comm);
  for_each_process(p){
      if(strcmp(p->comm,"forking")==0){/*find parent pid (first process with name forking)*/
          printk("Parent forking pid = %d\n",p->pid);
          printk("Children list: \n");
          print_children(p); /*print children of parent process*/
          break;/*stop the iteration in task list  (only the first forking process needed)*/
      }
  }
  printk("End of children \n");
  return 0;
}
void print_children(struct task_struct *p)
{       struct task_struct *child;
        struct list_head *list=&(p->children);/*pointer to the head of children list*/
        list_for_each_entry(child,list,sibling){/*iterate over siblings in children list*/
                printk("child  pid = %d\n",child->pid);
                /*recursive call to print_children(child) for the children of children to be printed*/
                print_children(child);/
        }
}
static void my_proc_exit(void){
        printk("Current process: pid = %d; name = %s\n",current ->pid , current ->comm);
}

module_init(my_proc_init);
module_exit(my_proc_exit);
```

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
1/12/2024

Τρέχω σε ένα terminal το forking.c και σε ένα άλλο φορτώνω το module list-children.ko πριν τερματιστεί το πρόγραμμα. Παρακάτω φαίνονται τα αποτελέσματα:

```
root@nataliarouska:/media/sf_shared_folder_vm# gcc forking.c -o forking
root@nataliarouska:/media/sf_shared_folder_vm# ./forking
PID: 575238
Forked! PID: 575239
Forked! PID: 0
Forked! PID: 575240
Forked! PID: 0
Forked! PID: 575241
Forked! PID: 0
Forked! PID: 575242
Forked! PID: 575243
Forked! PID: 0
Forked! PID: 0
Forked! PID: 575244
Forked! PID: 575245
Forked! PID: 0
Forked! PID: 0
Forked! PID: 575246
Forked! PID: 575249
Forked! PID: 575247
Forked! PID: 575248
Forked! PID: 0
Forked! PID: 0
Forked! PID: 0
Forked! PID: 0
Forked! PID: 575250
Forked! PID: 0
Forked! PID: 575251
Forked! PID: 575252
Forked! PID: 575253
Forked! PID: 0
Forked! PID: 0
Forked! PID: 0
```

-

```
root@nataliarouska:~/list-children# insmod list-children.ko
root@nataliarouska:~/list-children# rmmod list_children.ko
root@nataliarouska:~/list-children# journalctl --since "1 minutes ago" |grep kernel
Dec 07 17:52:24 nataliarouska kernel: Current process: pid = 575254; name = insmod
Dec 07 17:52:24 nataliarouska kernel: Parent forking pid = 575238
Dec 07 17:52:24 nataliarouska kernel: Children list:
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575239
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575241
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575245
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575252
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575250
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575244
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575253
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575251
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575240
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575243
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575248
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575246
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575242
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575249
Dec 07 17:52:24 nataliarouska kernel: child  pid = 575247
Dec 07 17:52:24 nataliarouska kernel: End of children
Dec 07 17:52:28 nataliarouska kernel: Current process: pid = 575256; name = rmmod
```

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
1/12/2024
Παρατηρούμε ότι το κάθε παιδί παίρνει το pid του προηγούμενου αυξημένο κατά 1 και το πρώτο παιδί το pid του γονέα αυξημένο κατά 1. Ωστόσο, αυτά δεν εμφανίζονται με τη σειρά στα logs του συστήματος γιατί τα παιδια του γονέα καλούν και αυτά την fork(), οπότε κάνουν δικά τους παιδιά και αφού τρέχουν ταυτόχρονα είναι τυχαία η σειρά που θα τυπωθούν τα pids.