Εργασία Ανάπτυξης Αρθρώματος Πυρήνα (Kernel Module) Λειτουργικά Συστήματα (ECE ΓΚ702)

Πανεπιστήμιο Πατρών — 2022

1 Εισαγωγή

Η εργασία αυτή κάνει μία εισαγωγή στη συγγραφή απλών αρθρωμάτων πυρήνα (kernel modules) για το λειτουργικό σύστημα Linux.

Μία λεπτομερή εισαγωγή για το Linux μπορείτε να βρείτε εδώ.

1.1 Ο πηγαίος κώδικας του Linux kernel.

Ο πηγαίος κώδικας του Linux kernel είναι διαθέσιμος ελεύθερα. Κάποιος μπορεί να κατεβάσει τον κώδικα, να τον επεξεργαστεί και να δημιουργήσει δικές του εκδόσεις του Linux kernel. Στα πλαίσια των εργασιών θα ζητηθεί να αναζητήσετε ορισμούς μέσα από τον πηγαίο κώδικα.

Τρόποι εξευρεύνησης του πηγαίου κώδικα είναι:

- Λήψη του κώδικα από το kernel.org και χρήση του αγαπημένου σας editor (Visual Studio Code, Vim, Emacs ...) για αναζήτηση σε αυτόν.
- Χρήση της ιστοσελίδας elixir.bootlin.com (προτείνεται)

1.2 Τί είναι ένα Kernel Module;

Ο πυρήνας (kernel) των Linux είναι μονολιθικός, πράγμα που σημαίνει πως αν κάποιος ήθελε να προσθέσει νέα λειτουργικότητα σε αυτόν θα έπρεπε να επεξεργαστεί τον πηγαίο κώδικα, να τον επαναμεταγλωτίσει (recompile) και να κάνει επανεκκίνηση το σύστημά του ώστε να δει τις αλλαγές του.

Αυτή η διαδικασία προφανώς είναι κουραστική και περίπλοκη. Έτσι, ο πυρήνας Linux υποστηρίζει modules (αρθρώματα) τα οποία φορτώνονται στον ήδη υπάρχοντα πυρήνα κατά τη διάρκεια εκτέλεσής του και μπορούν να είναι οδηγοί συσκευής (device drivers), κάποιο σύστημα αρχείων (file system), πρωτόκολλο δικτύωσης (network protocol) κ.α.

1.3 Προετοιμασία περιβάλλοντος ανάπτυξης.

Ένα kernel module έχει άμεση πρόσβαση στο υλικό του συστήματος. Αυτό σημαίνει πως υπάρχει η δυνατότητα να φέρει το σύστημα σε κατάσταση που χρειάζεται η επανεκκίνησή του. Για αυτό το λόγο προτείνεται να δουλέψετε σε κάποιο εικονικό μηχάνημα (virtual machine) ώστε να αποφύγετε το ρίσκο βλάβης στο σύστημά σας.

Για να έχετε ένα εικονονικό μηχάνημα μπορείτε να χρησιμοποιήσετε το δωρεάν λογισμικό Virtual Box Από εκεί, μπορείτε να δημιουργήσετε ένα εικονικό μηχάνημα χρησιμοποιώντας οποιαδήποτε διανομή των Linux επιθυμείτε. Διαφοροποιήσεις μεταξύ διανομών θα υπάρχουν μόνο στη διαδικασία απόκτησης των απαραίτητων πακέτων για την ανάπτυξη του κώδικα του module.

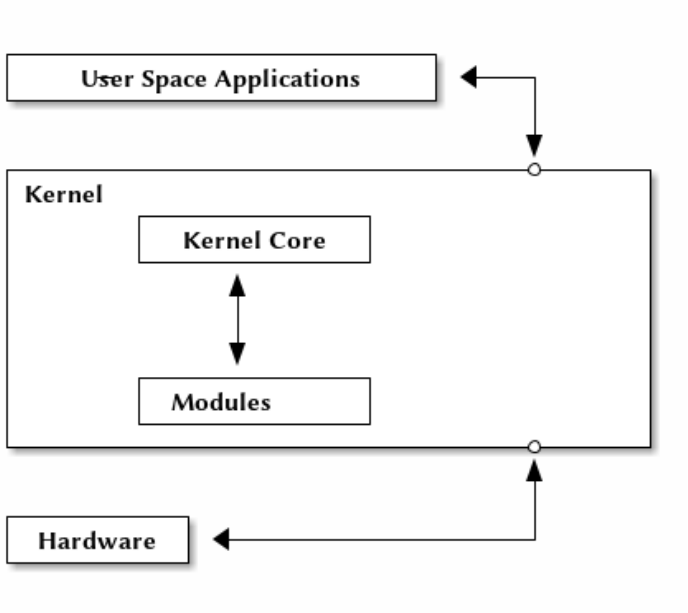


Figure 1: Όταν ένα kernel module φορτώνεται είναι πλέον μέρος του πυρήνα.

Για να είναι η ανάπτυξη του κώδικα γρηγορότερη, χρήσιμο θα ήταν να χρησιμοποιήσετε τη δυνατότητα "shared folder" του Virtual Box, έτσι ώστε να γίνεται η συγγραφή του κώδικα στο κανονικό σας σύστημα και έπειτα η μεταγλώττιση και φόρτωση των modules στο εικονικό μηχάνημα. Οδηγίες για αυτό μπορείτε να βρείτε εδώ και εδώ. Εναλλακτικά μπορείτε να κάνετε ακόμα και τη συγγραφή του κώδικα στο εικονικό σας μηχάνημα εγκαθιστώντας σε αυτό κάποιον επεξεργαστή κειμένου.

1.3.1 Απαραίτητα Πακέτα

Θα χρειαστεί να υπάρχουν στο σύστημά σας ορισμένα πακέτα και τα σχετικά header files:

Για διανομές βασισμένες στα Debian:

Command Line

```
$ sudo apt-get update

$ sudo apt-get install build-essential kmod
$ sudo apt-get install linux-headers-`uname -r`
```

Για διανομές βασισμένες στα Arch:

Command Line

```
$ sudo pacman -S gcc kmod
$ sudo pacman -S linux-headers
```

Προσοχή Φροντίστε να έχετε κατεβάσει τους κατάλληλους linux-headers για την έκδοση του Linux kernel που τρέχει στο σύστημά σας. Μπορείτε να τη βρείτε με την εντολή `uname -r`.

1.4 Ένα Παράδειγμα

Για αρχή, στο σύστημά σας ανοίξτε ένα τερματικό (terminal). Εκτελέστε την εντολή `su` και πληκτρολογήστε το root password ώστε να εκτελέσετε όλες τις υπόλοιπες εντολές ως διαχειριστής (root). Πολλές από τις εντολές διαχείρισης modules πρέπει να εκτελούνται με δικαιώματα root.

Μπορείτε να δείτε τα modules που είναι τώρα φορτωμένα στο σύστημά σας με την εντολή `sudo lsmod`.

Φτιάξτε ένα φάκελο. Εκεί θα βρίσκονται όλα τα αρχεία που θα αφορούν το module που θα φτιάξουμε.

Command Line

```
$ mkdir ~/hello-world
$ cd ~/hello-world
```

1.4.1 Kernel Module - Hello World

Αποθηκεύστε τον παρακάτω κώδικα σε ένα αρχείο με όνομα `hello.c`.

hello.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("Hi\n");
        return 0;
}

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

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

Ο κώδικας που βρίσκεται στη συνάρτηση `my_init` θα εκτελεστεί όταν φορτώσουμε το module μας. Εδώ γίνονται τυχόν αρχικοποιήσεις πόρων που θα χρειαστεί το module. Επιστρέφει έναν ακέραιο ανάλογα με τον αν ήταν επιτυχής ή όχι η αρχικοποίηση του module. Ο κώδικας που βρίσκεται στη συνάρτηση `my_exit` θα εκτελεστεί όταν αποφορτώσουμε το module μας. Εδώ θα πρέπει το module μας να επιστρέψει πίσω στο σύστημα τυχόν πόρους που έχει δεσμεύσει.

Οι μακροεντολές (macros) `module_init` και `module_exit` ορίζουν τις συναρτήσεις μας ως συναρτήσεις "εκίνησης" και "εξόδου" για το τελικό kernel object.

Οι μακροεντολές `MODULE_DESCRIPTION`, `MODULE_AUTHOR` και `MODULE_LICENSE` ορίζουν metadata για το module μας.

Παρατηρήστε τη χρήση της συνάρτησης `printk` έναντι κάτι όπως `printf`. Όταν αναπτύσετε κώδικα σε kernel space, δεν έχετε διαθέσιμη τη βασική βιβλιοθήκη της C. Έχετε διαθέσιμα μόνο σύμβολα τα οποία

έχουν οριστεί από τον πυρήνα. Για να δείτε τα σύμβολα που είναι ορισμένα από τον πυρήνα στο σύστημά σας εκτελέστε την εντολή `cat /proc/kallsyms`.

Αποθηκεύστε τον παρακάτω κώδικα σε ένα αρχείο με όνομα `Makefile`.

Makefile

```
obj-m += hello.o # Change this for different module names.

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
```

Αυτό το αρχείο ορίζει τις εντολές οι οποίες θα εκτελεστούν για να μεταγλωτιστεί το module που φτιάχνουμε.

Σημείωση: Στο αρχείο `Makefile` (και σε όλα τα `makefiles` γενικότερα) πρέπει κάτω από το όνομα του κάθε κανόνα (`all, clean`) να υπάρχει χαρακτήρας `<TAB>`. Αυτό είναι παρόμοιο με τον τρόπο που η γλώσσα Python χειρίζεται τις block δομές της.

Για να μεταγλωτίσετε το module, στον φάκελο που έχετε όλα τα αρχείο εκτελέστε την εντολή `make`. Για να σβήσετε τα αρχεία που δημιουργούνται από τη μεταγλώτισση μπορείτε να χρησιμοποιήσετε την εντολή `make clean`.

Ο φάκελός σας πρέπει τώρα να περιέχει τα παρακάτω αρχεία:

Command Line

```
$ ls hello-world

../hello-world
   hello.c
   hello.ko
   hello.mod
   hello.mod.c
   hello.mod.o
   hello.o
   Makefile
   modules.order
   Module.symvers
```

Από αυτά, το αρχείο `hello.ko` είναι το τελικό kernel object το οποίο και θα φορτώσουμε στον πυρήνα του συστήματός μας.

Μπορείτε να δείτε μερικές πληροφορίες για το module με την εντολή:

Command Line

```
$ modinfo hello.ko
```

Μπορούμε πλέον να **φορτώσουμε** το module στο σύστημά μας με την εντολή:

Command Line

```
$ sudo insmod hello.ko
```

Τώρα θα δείτε το module μας στη λίστα με τα modules του συστήματος αν εκτελέσουμε:

Command Line

```
$ sudo lsmod | grep hello
```

Για να **αφαιρέσουμε** το module μας εκτελούμε την εντολή:

Command Line

```
$ sudo rmmod hello
```

Τώρα αν ελέγξουμε τα logs του συστήματος με την εντολή:

Command Line

```
$ journalctl --since "10 minutes ago" | grep kernel
```

Μπορούμε να δούμε τα μηνύματα που είχαμε προσθέσει στις συναρτήσεις `my_init` και `my_exit`

Το module μας δεν έχει δυνατότητα εκτύπωσης μυνημάτων σε γραφική κονσόλα (υπάρχει βέβαια δυνατότητα εξόδου σε κονσόλα τύπου tty). Για αυτό πρέπει να βλέπουμε τα μυνήματα από τα logs του συστήματος.

2 Ασκήσεις

Άσκηση 1

Τροποποιήστε το αρχικό module που φτιάξαμε ώστε να τυπώνει ένα διαφορετικό μήνυμα. Ακολουθήστε την ίδια διαδικασία για την φόρτωση/εκφόρτωσή του.

Σημείωση: Μήν βασιστείτε σε απλή αντιγραφή-επικόλληση του κώδικα. Προσπαθήστε να γράψετε μόνοι σας από την αρχή το πρόγραμμα γραμμή-γραμμή ώστε να γίνει κατανοητή η δομή του.

Άσκηση 2

Πηγαίνετε στον πηγαίο κώδικα του Linux kernel και διαβάστε τον ορισμό για το `task_struct` (στο αρχείο `linux/include/linux/sched.h`. Τί πληροφορίες αποθηκεύονται σε αυτή τη δομή; Τί αντιπροσωπεύει;

Σημείωση: Για μια λεπτομερή παρουσίαση της δομής `task_struct` και του διαχειρισμού διεργασιών στο Linux Kernel μπορείτε να βρείτε στο βιβλίο Linux Kernel Development (Κεφάλαιο 3).

Άσκηση 3

Γράψτε ένα module πυρήνα το οποίο Θα τυπώνει πληροφορίες για όλες τις υπάρχουσες διεργασίες όταν φορτώνεται. Για να πάρουμε έναν δείκτη προς την διεργασία που εκτελείται αυτή τη στιγμή μπορούμε να χρησιμοποιήσουμε το macro `current`. Μπορείτε να βρείτε τον ορισμό του macro αυτού μέσα στο αρχείο `linux/arch/x86/include/asm/current.h`.

Ξεκινήστε με τον παρακάτω κώδικα σαν βάση.

Κάντε τις κατάλληλες προσθήκες στη συνάρτηση `my_proc_init` (σημείο `TODO`) ώστε να τυπωθούν οι αντίστοιχες πληροφορίες για όλες τις διεργασίες του συστήματος.

Για το πώς γίνεται αυτό κοιτάξτε στο αρχείο (`linux/include/linux/sched/signal.h` στο macro `for_each_process`).

list-processes.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");
    /* TODO */

    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);
```

Σημείωση Θα πρέπει για το νέο module να φτιάξετε νέο φάκελο και εκεί να τοποθετήσετε τον πηγαίο κώδικα (.c αρχείο) και το `Makefile` κάνοντας κατάλληλη αλλαγή σε αυτό.

Άσκηση 4

Στην άσκηση αυτή σας ζητείται να φτιάξετε ένα kernel module το οποίο θα τυπώνει πληροφορίες για μία διεργασία και τις θυγατρικές τις διεργασίες.

Ξεκινήστε δημιουργώντας ένα πρόγραμμα όπως το παρακάτω.

forking.c

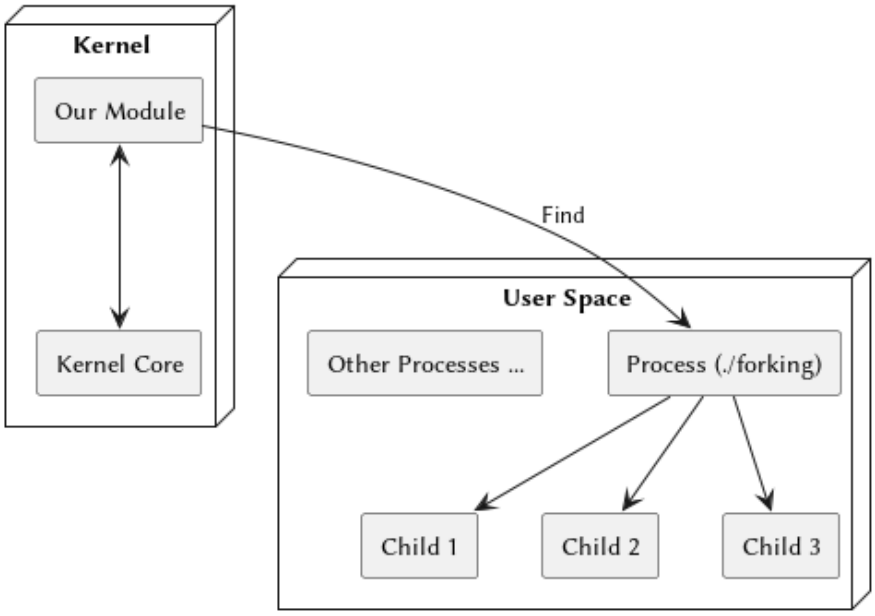
```c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>

#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) exit(0);

        printf("Forked! PID: %d\n", fork());
    }

    return 0;
}
```

Μεταγλωτίστε και έπειτα εκτελέστε το πρόγραμμα με τις εντολές:

Command Line

```
$ gcc forking.c -o forking
$ ./forking
```

Αυτό το πρόγραμμα ανά `SLEEP_TIME` δευτερόλεπτα εκτελεί την κλήση συστήματος (system call) `fork`. Ζητείται μέσα από το kernel που θα γράψετε να βρείτε τη δομή `task_struct` που αντιστοιχεί σε αυτή τη διεργασία με βάση το `PID` της και έπειτα να τυπώσετε το `PID` καθεμίας από τις διεργασίες παιδιά της.



Το πρόγραμμα `forking` θα το αφήσετε να τρέχει. Σε άλλο τερματικό θα χρειαστεί να μεταγλωττίσετε και να φορτώσετε το module σας.

Στο αρχείο `list-children.c` βρίσκεται ο σκελετός με βάση τον οποίο θα προσθέσετε τη ζητούμενη λειτουργικότητα. Συμπληρώστε τον κώδικά σας στο σημείο `TODO`.

Θα πρέπει να εκτελέσετε το πρόγραμμά σας, να θέσετε το κατάλληλο PID στο module σας για να βρει τη διεργασία, να το μεταγλωττίσετε και έπειτα να το φορτώσετε.

Προσοχή Αν κάνετε επανεκκίνηση τη διεργασία σας, πιθανότατα θα έχει ένα νέο, διαφορετικό `PID`. Έτσι, για να τυπώσετε πληροφορίες για τα παίδια της από το module σας θα πρέπει να εκφορτώσετε το module σας, να αλλάξετε το `PID` το οποίο θα αναζητήσει, να το επαναμεταγλωτίσετε και να το επαναφορτώσετε.

Για να έχετε πρόσβαση στις διεργασίες-παιδιά μιας διεργασίας, χρησιμοποιήστε το μέλος `children` του `task_struct`. Αυτό είναι μία δομή διπλά διασυνδεδεμένης λίστας στους κόμβους της οποίας βρίσκονται δομές `task_struct *`. Στο Linux Kernel υπάρχουν διάφορα βοηθητικά macros για το χειρισμό λιστών.

1. Εντοπίστε τα πεδία `children` και `sibling` στην δομή `task_struct` (στον πηγαίο κώδικα του Linux Kernel), τί αντιπροσωπεύει το καθένα;
2. Διαβάστε τον τρόπο χρήσης του βοηθητικού macro `list_for_each_entry` (στο αρχείο `linux/tools/include/linux/list.h`)
3. Προσθέστε στο σκελετό της άσκησης τον κατάλληλο κώδικα ώστε να τυπώνεται το PID κάθε διεργασίας παιδί της αρχικής σας διεργασίας, τί παρατηρείτε;

Περισσότερες Πηγές

- Linux Kernel Development
- Linux Kernel Module Programming Guide
- Linux Device Drivers
- Operating System Concepts 9th Edition - Silbershatz, Galvin, Gagne (ch. 18)