Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Λειτουργικά Συστήματα (ECE318)

Ακαδημαϊκό Έτος 2020-2021

1η Εργαστηριακή Άσκηση

Τελευταία Ενημέρωση: 16 Μαρτίου 2021

Περιεχόμενα

1 Εισαγωγικά

1.1 Προαπαιτούμενα

Πριν ξεκινήσετε την υλοποίηση θα πρέπει να έχετε διαβάσει τις οδηγίες που σας έχουν δοθεί, όπως επίσης τα κεφάλαια 1 και 2 και 5 από το βιβλίο ”Linux Kernel Development” και το κεφάλαιο 2 από το βιβλίο ”Linux Device Drivers”. Επίσης θα πρέπει προφανώς να έχετε καταφέρει να εγκαταστήσετε σωστά το λειτουργικό Ubuntu και να μπορείτε να μετακινήσετε αρχεία μεταξύ του λειτουργικού συστήματος ξενιστή (host) και του Ubuntu που τρέχει στο πλαίσιο της ιδεατής μηχανής.

1.2 Προθεσμία και Τρόπος Παράδοσης

Η άσκηση θα πρέπει να παραδοθεί έως τα **μεσάνυχτα της Τρίτης 31/3/2021**. Η προθεσμία είναι τελική και δεν πρόκειται να δοθεί καμία παράταση – συνολικά ή ατομικά – για κανένα λόγο. Ο χρόνος που σας δίνεται υπερεπαρκεί. Χρησιμοποιήστε τον ”σοφά”.

Η παράδοση θα γίνει στο e-class. Για τις λεπτομέρειες της παράδοσης δείτε την παράγραφο 3.

1.3 Κώδικας Ηθικής

Κάθε ομάδα θα πρέπει να εργαστεί ανεξάρτητα. Είναι δεκτό και θεμιτό διαφορετικές ομάδες να ανταλλάξουν απόψεις σε επίπεδο γενικής ιδέας ή αλγόριθμου. Απαγορεύεται όμως κάθε συζήτηση / ανταλλαγή κώδικα ή ψευδοκώδικα. Σημειώστε ότι οι ασκήσεις θα ελεγχθούν τόσο από αυτόματο σύστημα όσο και από εμάς για ομοιότητες μεταξύ των ομάδων αλλά και για ομοιότητες με τυχόν λύσεις που μπορεί να βρεθούν στο διαδίκτυο ή λύσεις που έχουν δοθεί σε προηγούμενα έτη. Σε περίπτωση αντιγραφής οι ασκήσεις (όλων των φάσεων) όλων των εμπλεκόμενων φετινών ομάδων μηδενίζονται χωρίς καμία περαιτέρω συζήτηση.

Όλα τα μέλη στο εσωτερικό κάθε ομάδας θα πρέπει να έχουν ισότιμη συμμετοχή στην ανάπτυξη κάθε φάσης. Διατηρούμε το δικαίωμα να επιβεβαιώσουμε αν αυτό τηρείται με προσωπικές συνεντεύξεις / προφορική εξέταση.

2 Ζητούμενα

2.1 Πρώτο Μέρος: Δημιουργία μίας Νέας Κλήσης Συστήματος

Ως ένα πρώτο, εισαγωγικό βήμα στη διαδικασία τροποποίησης του πυρήνα του Linux, καλείστε να προσθέσετε μία νέα κλήση συστήματος. Η κλήση συστήματος θα ονομάζεται *find_roots* και θα έχει το ακόλουθο πρότυπο:

```
long find_roots(void)
```

Η νέα κλήση συστήματος θα εκτυπώνει το αναγνωριστικό (process id) και το όνομα της διεργασίας που την κάλεσε. Έπειτα, θα εκτυπώνει το όνομα και το αναγνωριστικό για κάθε διεργασία-πρόγονο της καλούσας διεργασίας. Ένα ενδεικτικό στιγμιότυπο από την κλήση της *find_roots()* παρουσιάζεται στην εικόνα 1.

find_roots system call called by process 9446
id: 9446, name: find_roots_lib
id: 9425, name: bash
id: 2284, name: gnome-terminal
id: 1668, name: init
id: 1358, name: lightdm
id: 1203, name: lightdm
id: 1, name: init

Εικόνα 1: Ενδεικτική έξοδος από την εκτέλεση της κλήσης συστήματος.

Στο λειτουργικό σύστημα Linux, πρόγονος όλων των διεργασιών είναι η διεργασία *init*, η οποία έχει process id ίσο με 1. Συνεπώς, ο βρόγχος εκτύπωσης θα πρέπει να τερματίζεται όταν βρεθεί η *init*.

Υλοποιήστε όλες τις απαραίτητες αλλαγές στον πυρήνα προκειμένου να προσθέσετε τη νέα κλήση. Γι’ αυτή τη διαδικασία, μπορείτε να ακολουθήσετε τις οδηγίες του αντίστοιχου οδηγού που είναι διαθέσιμος στη σελίδα του εργαστηρίου. Στη συνέχεια μεταγλωττίστε και εγκαταστήστε τον νέο πυρήνα και επανεκκινήστε το σύστημά σας.

Το τελευταίο βήμα, μετά την επανεκκίνηση του συστήματος, είναι η ενεργοποίηση της νέας κλήσης συστήματος. Αυτό θα πραγματοποιηθεί μέσω μίας βιβλιοθήκης επιπέδου χρήστη, όπως περιγράφεται στον αντίστοιχο οδηγό και εφαρμογής που θα τη χρησιμοποιεί. Δημιουργήστε την

wrapper library με όνομα *libroots.a* (ονομάστε τα σχετικά αρχεία κώδικα *roots.c* και *roots.h*) και ένα αρχείο προγράμματος με όνομα *find_roots_lib.c*, το οποίο θα χρησιμοποιεί την wrapper library για να πραγματοποιήσει την κλήση συστήματος. **Για ακόμη μία φορά υπενθυμίζεται ότι δεν θα πρέπει να αποθηκεύσετε τα αρχεία του κώδικα επιπέδου χρήστη στο source tree του πυρήνα. Μπορείτε να τα αποθηκεύσετε στο home directory σας.**

Παρακάτω, μπορείτε να βρείτε κάποια βασικά στοιχεία που θα σας διευκολύνουν στην υλοποίηση της ζητούμενης λειτουργικότητας:

- Το PCB (process control block) της κάθε διεργασίας είναι μία δομή τύπου task_struct. Αυτή ορίζεται στο αρχείο *include/linux/sched.h*. Θα πρέπει να μελετήσετε προσεκτικά τη δομή της και να προσθέσετε τη γραμμή

  ```
  #include <linux/sched.h>
  ```

  στο αρχείο με την υλοποίηση της κλήσης συστήματος.

- Ο δείκτης προς τη δομή *task_struct* της τρέχουσας διεργασίας, δηλαδή της διεργασίας που ενεργοποίησε το system call αποθηκεύεται στην global μεταβλητή με όνομα *current*.

- Το id της διεργασίας, το όνομα του εκτελέσιμου που αυτή εκτελεί και ο δείκτης προς τον ”πατέρα” της αποτελούν και αυτά πεδία της δομής *task_struct*.

2.2 Δεύτερο Μέρος: Τροποποίηση ενός Kernel Module

Το δεύτερο μέρος της παρούσας εργασίας έχει ως στόχο να σας εξοικειώσει με τον κώδικα ενός kernel module και με τις διαδικασίες μεταγλώττισης και φόρτωσης-αφαίρεσης από τον πυρήνα. Ως παράδειγμα, θα χρησιμοποιηθεί ένα module που χρησιμοποιείται από τον πυρήνα για δρομολόγηση Εισόδου/Εξόδου. Το module είναι ο *Kyber* I/O scheduler, ένας δρομολογητής που εξυπηρετεί τις διάφορες αιτήσεις για Είσοδο/Εξοδο σε συσκευές βάσει ενός αλγορίθμου πολλαπλών ουρών που βασίζεται σε token.

2.2.1 Διαδικασία Τροποποίησης

Δημιουργήστε στο home directory σας έναν φάκελο με όνομα project1_module. Αντιγράψτε σε αυτόν το αρχείο *block/kyber-iosched.c* και μετονομάστε το σε *project1-kyber.c*. Στη συνέχεια θα

προχωρήσετε σε μερικές απλές τροποποιήσεις του κώδικα, οι οποίες δεν αφορούν σε τροποποιήσεις στις δομές ή τους αλγορίθμους που χρησιμοποιούνται.

Ως πρώτο βήμα, προσθέστε το πρόθεμα *teamXX_*, όπου XX ο αριθμός της ομάδας σας, σε όλες τις συναρτήσεις του δρομολογητή *kyber* που ξεκινάνε με το πρόθεμα *kyber_*. Επίσης, αλλάξτε το όνομα της δομής *elevator-type* και τροποποιήστε κατάλληλα τις εγγραφές της προκειμένου να αναγνωρίζονται τα ονόματα συναρτήσεων μετά τις τροποποιήσεις. Οι συναρτήσεις που αναφέρονται σε αυτή τη δομή, είναι αυτές που χρειάζονται την παραπάνω προσθήκη. **ΠΡΟΣΟΧΗ**: Μην πειράξετε τις κλήσης των συναρτήσεων που ξεκινάνε με το πρόθεμα *trace_*.

Στη συνέχεια, αντικαταστήστε το περιεχόμενο του πεδίου *elevator_name* της δομής με το όνομα της ομάδας σας και τροποποιήστε κατάλληλα τα ορίσματα στα macros *MODULE_AUTHOR* και *MODULE_DESCRIPTION*. Τέλος, προσθέστε τη γραμμή

```
printk( "In teamXX_kyber_dispatch_request function\n" );
```

ως δεύτερη εντολή στη συνάρτηση *teamXX_kyber_dispatch_request*. Η εντολή αυτή θα εκτυπώνει το αντίστοιχο μήνυμα κάθε φορά που θα καλείται η συνάρτηση για την εξυπηρέτηση κάποιας αίτησης. Με αυτό τον τρόπο θα μπορούμε να διαπιστώσουμε κατά πόσο η δρομολόγηση πραγματοποιείται από το τροποποιημένο module.

Επιπλέον, προκειμένου να γίνει σωστά το compilation πρέπει να αλλάξετε στο αρχείο *project1-kyber.c* τις παρακάτω γραμμές

```
#include "blk.h"
#include "blk-mq.h"
#include "blk-mq-debugfs.h"
#include "blk-mq-sched.h"
#include "blk-mq-tag.h"
```

ώστε να συμπεριλαμβάνουν τη διαδρομή στην οποία βρίσκονται αυτά τα header files, που είναι στο φάκελο /usr/src/linux-5.4.86-dev/block. Θα πρέπει δηλαδή οι παραπάνω γραμμές να αλλάξουν σε

```
#include "/usr/src/linux-5.4.86-dev/block/blk.h"
#include "/usr/src/linux-5.4.86-dev/block/blk-mq.h"
#include "/usr/src/linux-5.4.86-dev/block/blk-mq-debugfs.h"
#include "/usr/src/linux-5.4.86-dev/block/blk-mq-sched.h"
#include "/usr/src/linux-5.4.86-dev/block/blk-mq-tag.h"
```

Σε αυτό το σημείο μπορείτε να μεταγλωττίσετε τα αρχεία του module. Ακολουθήστε τις οδηγίες του φυλλαδίου ”Εισαγωγικός Οδηγός στα Linux Kernel Modules” προκειμένου να δημιουργήσετε το κατάλληλο Makefile και να φορτώσετε το module στον πυρήνα. Μπορείτε να προχωρήσετε στο compile και το module σας θα είναι έτοιμο προς χρήση.

2.2.2 Έλεγχος Λειτουργίας του Kernel Module

Προκειμένου να μπορεί να χρησιμοποιηθεί ένα kernel module που υλοποιεί I/O scheduling, θα πρέπει να ενημερώσουμε τον πυρήνα για τη συσκευή την οποία θα αναλάβει να διαχειρίζεται το συγκεκριμένο module. Για λόγους επίδειξης, θα χρησιμοποιήσουμε τη συσκευή /dev/sda η οποία, με βάση την ονοματολογία που χρησιμοποιεί το Linux για τις συσκευές, αντιστοιχεί στον πρώτο σκληρό δίσκο του συστήματος. Σε περίπτωση που εργάζεστε σε native installation, αντί για sda στις παρακάτω οδηγίες χρησιμοποιήστε το όνομα του δίσκου που χρησιμοποιεί η εγκατάσταση linux σας (sdb, sdc ...). Μπορείτε να διαπιστώσετε ποιος είναι αυτός εκτελώντας την εντολή *mount*, η οποία σας αποκαλύπτει ποιες συσκευές αποθήκευσης χρησιμοποιείτε και από ποιο σημείο του συστήματος αρχείου είναι προσπελάσιμη η κάθε συσκευή. Παρακάτω βλέπετε ένα ενδεικτικό αποτέλεσμα εκτέλεσης της *mount*.

```
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620...
tmpfs on /run type tmpfs (rw,nosuid,nodev,noexec,relatime,size=200716k...
/dev/mapper/ubuntu--vg-ubuntu--lv on / type ext4 (rw,relatime)
/dev/sda2 on /boot type ext4 (rw,relatime)
...
```

Από το στιγμιότυπο προκύπτει ότι ο κατάλογος / (η ρίζα του συστήματος αρχείων) αντιστοιχεί στη συσκευή mapper/ubuntu–vg-ubuntu–lv, ενώ ο κατάλογος /boot στη συσκευή sda2 (το 2ο partition του 1ου δίσκου του συστήματος, δηλαδή του sda). Οι συσκευές mapper είναι εικονικές συσκευές που χρησιμοποιούνται από το σύστημα διαχείρισης λογικών δίσκων (LVM) των Ubuntu, το οποίο είναι η προεπιλογή για τις σύγχρονες εκδόσεις. Προκειμένου να δείτε σε ποιά συσκευή αντιστοιχεί το mapper/ubuntu–vg-ubuntu–lv εκτελέστε την εντολή *lsblk*. Παρακάτω βλέπετε ένα ενδεικτικό αποτέλεσμα της εκτέλεσης της *lsblk*.

```
sda                     8:0    0   20G  0 disk
|-sda1                  8:1    0    1M  0 part
|-sda2                  8:2    0    1G  0 part /boot
```

```
|-sda3                      8:3    0   19G  0 part
  |-ubuntu--vg-ubuntu--lv   253:0  0   19G  0 lvm  /
```

Από το παραπάνω στιγμιότυπο βλέπουμε πως η λογική συσκευή ubuntu–vg-ubuntu–lv αντιστοιχεί στη συσκευή sda, και συγκεκριμένα στο 3ο partition αυτής της συσκευής.

Εισάγετε το module στον πυρήνα χρησιμοποιώντας την εντολή *insmod*. Έπειτα, θα πρέπει να ελέγξετε τα I/O scheduling modules που είναι διαθέσιμα για τη συσκευή /dev/sda. Αυτό μπορεί να πραγματοποιηθεί με την εντολή *cat /sys/block/sda/queue/scheduler*. Μία ενδεικτική έξοδος από την εκτέλεση της εντολής είναι η παρακάτω:

```
[mq-deadline] team00_kyber none
```

Όπως μπορούμε να παρατηρήσουμε, υπάρχουν τρεις διαθέσιμοι schedulers. Το όνομα του scheduler που χρησιμοποιείται είναι αυτό που περικλείεται από τις αγκύλες. Για να ενεργοποιήσετε το δικό σας scheduler θα πρέπει να εκτελέσετε την εντολή

sudo bash -c ’echo team00_kyber > /sys/block/sda/queue/scheduler’

Μετά από αυτή την εντολή, η έξοδος από την εκτέλεση της εντολής

cat /sys/block/sda/queue/scheduler

θα πρέπει να είναι η ακόλουθη:

```
mq-deadline [team00_kyber] none
```

Όπως παρατηρείτε, η διαχείριση του I/O scheduling θα πραγματοποιείται πλέον από τον τροποποιημένο kyber driver. Για να επιβεβαιώσετε ότι αυτό συμβαίνει, δημιουργήστε ένα αρχείο στο home directory σας και προσθέστε κάποιο κείμενο. Έπειτα, εκτελέστε την εντολή *dmesg | tail*. Ανάμεσα στα μηνύματα που εμφανίζονται, θα πρέπει να είστε σε θέση να δείτε και το μήνυμα ”In teamXX_kyber_dispatch_request function”.

Μετά την επιβεβαίωση της σωστής λειτουργίας, θα πρέπει *οπωσδήποτε* να επαναφέρετε τον αρχικό scheduler για τη συσκευή /dev/sda. Γι’ αυτό το σκοπό, εκτελέστε την εντολή

sudo bash -c ’echo mq-deadline > /sys/block/sda/queue/scheduler’

Τέλος, αφαιρέστε το module *project1-kyber* από τον πυρήνα χρησιμοποιώντας την εντολή *rmmod*.

2.3 Τρίτο Μέρος: Δημιουργία sysfs αρχείων μέσω Kernel Module

Στο πρώτο μέρος της εργασίας είδατε πως επικοινωνεί κάποιο πρόγραμμα χρήστη με τον πυρηνά του λειτουργικού συστήματος Linux μέσω ενός system call που εσείς ορίσατε στον πυρήνα. Ένας άλλος, έμμεσος τρόπος επικοινωνίας, που δεν απαιτεί την μεταγλώττιση και εγκατάσταση εκ νέου του πυρήνα, είναι τα αρχεία sysfs. Στο δεύτερο μέρος της άσκησης χρησιμοποιήσατε τέτοια αρχεία sysfs που αφορούν τον I/O scheduler (/sys/block/sda/queue/scheduler) για να διαβάσετε τους διαθέσιμους schedulers (cat /sys/block/sda/queue/scheduler) και να ορίσετε τον δικό σας (sudo bash -c 'echo mq-deadline > /sys/block/sda/queue/scheduler').

Τα αρχεία sysfs είναι εικονικά αρχεία που χρησιμοποιούνται στο λειτουργικό σύστημα Linux για την εξαγωγή πληροφοριών και την ρύθμιση διάφορων υποσυστημάτων του πυρήνα Linux, όπως ο I/O scheduler.

Σε αυτό το μέρος της εργασίας καλείστε να δημιουργήσετε ενα δικό σας kernel module που θα δημιουργεί το sysfs αρχείο /sys/kernel/teamXX/find_roots. Όταν αυτό το αρχείο διαβάζεται από το επίπεδο χρήστη με την εντολή cat θα έχει την ίδια λογική με τον κώδικα που γράψαμε στο πρώτο μέρος: θα εμφανίζει το παρακάτω μήνυμα στην έξοδο του *dmesg*, ενώ η ανάγνωση από το αρχείο θα επιστρέφει την τιμή του pid της διεργασίας που το διάβασε (δηλαδή 8315 στο παρακάτω παράδειγμα).

find_roots sysfs opened by process 8315
id: 8315, name: cat
id: 8314, name: sudo
id: 2292, name: bash
id: 2284, name: gnome-terminal
id: 1668, name: init
id: 1358, name: lightdm
id: 1203, name: lightdm
id: 1, name: init

Εικόνα 2: Ενδεικτική έξοδος του dmesg κατά την ανάγνωση του sysfs αρχείου.

Παρακάτω, θα βρείτε έτοιμο κώδικα ενός kernel module που χρησιμοποιεί sysfs. Μπορείτε να βασιστείτε πάνω σε αυτόν και να κάνετε τις αντιστοιχές αλλαγές που ζητάει η εργασία:

```
#include <linux/module.h>
#include <linux/printk.h>
```

```
#include <linux/kobject.h>
#include <linux/sysfs.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/string.h>
#include <linux/kernel.h>
#include <linux/sched.h>

static struct kobject *example_kobject;
volatile int roots = 0;

static ssize_t foo_show(struct kobject *kobj, struct kobj_attribute *attr,
                        char *buf)
{
    return sprintf(buf, "Hello World!\n");
}

struct kobj_attribute foo_attribute = __ATTR(foo, 0660, foo_show, NULL);

static int __init mymodule_init (void)
{
        int error = 0;

        example_kobject = kobject_create_and_add("kobject_example",
                                                 kernel_kobj);
        if(!example_kobject)
                return -ENOMEM;

        error = sysfs_create_file(example_kobject, &foo_attribute.attr);
        if (error) {
                printk("failed to create the foo file in /sys/kernel/
kobject_example \n");
        }

        return error;
}

static void __exit mymodule_exit (void)
{
```

```
        printk("Module un initialized successfully \n");
        kobject_put(example_kobject);
}

module_init(mymodule_init);
module_exit(mymodule_exit);
MODULE_LICENSE("GPL");
```

Επεξήγηση επιμέρους συναρτήσεων του έτοιμου κώδικα:

- `kobject_create_and_add`:

  Δημιουργεί έναν εικονικό φάκελο στο /sys/kernel με το όνομα που ορίζεται από την πρώτη παράμετρο κατα την κλήση της

- `sysfs_create_file`:

  Δημιουργεί ένα εικονικό αρχείο στον εικονικό φάκελο (1η παράμετρος) με τις κατάλληλες ιδιότητες (2η παράμετρος)

- `kobject_put`:

  Διαγράφει εξολοκλήρου τον εικονικό φάκελο (1η παράμετρος) και όλα τα περιεχόμενα του

- `__ATTR`:

  Πρόκειται για μία μακροεντολή που ορίζει τις ιδιότητες ενός sysfs αρχείου, όπως το όνομα (1η παράμετρος), τα δικαιώματα ανάγνωσης/εγγραφής (2η παράμετρος), τη συνάρτηση που θα κληθεί κατά το διάβασμα του αρχείου (3η παράμετρος) και τη συνάρτηση που θα κληθεί κατά την εγγραφή του αρχείου (4η παράμετρος).
  ΣΗΜΕΙΩΣΗ: Στις παραμέτρους των συναρτήσεων ανάγνωσης και εγγραφής μπορούμε να περάσουμε την τιμή NULL οταν θέλουμε να αγνοήσουμε κάποια.

- `ssize_t foo_show(struct kobject *kobj,struct kobj_attribute *attr,char *buf)`:

  Πρότυπο συνάρτησης το οποίο μπορούμε να περάσουμε σαν την 3η παράμετρο στη μακροεντολή __ATTR.
  ΠΡΟΣΟΧΗ: Η συνάρτηση πρέπει πάντα να επιστρέφει το πλήθος των χαρακτήρων που έγραψε στο char *buf.

Δημιουργήστε στο home directory σας έναν φάκελο με όνομα sysfs_module και στο εσωτερικό του δημιουργήσετε ένα αρχείο με το όνομα sysfs_module.c, όπου θα περιέχει και τον κώδικα για

την ζητούμενη λειτουργικότητα της άσκησης. Μπορείτε να αντιγράψετε τον κώδικα που σας δώθηκε παραπάνω και να προβείτε σε τροποποιήσεις για την ολοκλήρωση της άσκησης.

3 Πακετάρισμα των Αλλαγών – Δημιουργία του προς Υποβολή Συνημμένου Αρχείου

3.1 Απομόνωση των Αλλαγών στον Κώδικα του Λειτουργικού

Προφανώς δεν είναι ιδιαίτερα πρακτικό να παραδώσετε τις αλλαγές στέλνοντας όλο το νέο κώδικα του λειτουργικού, δεδομένου και ότι τα αρχεία τα οποία στην πραγματικότητα θα έχετε πειράξει είναι ελάχιστα. Η εφαρμογή *diff* αναλαμβάνει να αναγνωρίσει τις αλλαγές που κάνατε στον πηγαίο κώδικα του λειτουργικού, συγκρίνοντάς τον με το αντίγραφο του αρχικού κώδικα που έχουμε κρατήσει στον κατάλογο /usr/src/linux-5.4.86-orig.

Κατ’ αρχήν πηγαίνετε στον κατάλογο /usr/src/linux-5.4.86-dev (cd /usr/src/linux-5.4.86-dev) και δώστε την εντολή *make distclean*. Με τον τρόπο αυτό σβήνονται όλα τα αρχεία (.o, εκτελέσιμα κλπ) που δημιουργήθηκαν κατά τη μεταγλώττιση. Κάνετε το ίδιο – για καλό και για κακό – και στον κατάλογο /usr/src/linux-5.4.86-orig. **ΜΗΝ ΞΕΧΑΣΕΤΕ ΑΥΤΑ ΤΑ 2 ΒΗΜΑΤΑ!!!**

Πλέον είστε έτοιμοι να “συγκρίνετε” τους 2 καταλόγους. Πηγαίνετε στο /usr/src και από εκεί δώστε την εντολή

sudo bash -c ’diff -ruN linux-5.4.86-orig linux-5.4.86-dev > patch_1’

Το πρόγραμμα diff εντοπίζει αναδρομικά τις αλλαγές μεταξύ των καταλόγων linux-5.4.86-orig και linux-5.4.86-dev. Το τελικό αποτέλεσμα αποθηκεύεται στο αρχείο patch_1 το οποίο δημιουργείται μέσα στον κατάλογο από τον οποίο καλέσατε την *diff* (δηλαδή τον κατάλογο /usr/src). Αν θέλετε, διαβάστε το αρχείο patch_1 (είναι ένα αρχείο κειμένου) και προσπαθήστε να καταλάβετε τη δομή του.

Αν θέλετε να επιβεβαιώσετε ότι το patch φτιάχτηκε σωστά, μπορείτε να κάνετε τα ακόλουθα: Η εφαρμογή *patch* μπορεί να πάρει ένα patch και να εφαρμόσει τις αλλαγές που περιγράφει το patch σε έναν κατάλογο πηγαίου κώδικα. Κατόπιν, μεταβείτε στον κατάλογο /usr/src/linux-5.4.86-orig και δώστε την εντολή

patch -p1 –dry-run < ../patch_1

Με την εντολή αυτή θα γίνει προσομοίωση εφαρμογής στον κατάλογο που βρισκόμαστε του patch που περιέχεται στο αρχείο patch_1 (το οποίο patch_1 είναι τοποθετημένο στον στον αμέσως προηγούμενο κατάλογο, γι' αυτό και το ../). Η παράμετρος *–dry-run* λέει στην εφαρμογή patch να προσποιηθεί ότι εφαρμόζει το patch χωρίς να κάνει στην πραγματικότητα οποιεσδήποτε αλλαγές στα αρχεία. Το patch θα πρέπει να εφαρμοστεί "καθαρά", δε θα πρέπει δηλαδή να σας εμφανιστούν μηνύματα λάθους. Δώστε προσοχή σε αυτό το βήμα, γιατί προϋπόθεση για τη βαθμολόγηση κάθε άσκησης είναι το patch που θα μας στείλετε να μπορεί να εφαρμοστεί "καθαρά".

3.2 Πακετάρισμα των Προς Υποβολή Στοιχείων σε Αρχείο

Το τελευταίο βήμα είναι να πακετάρετε όλα τα αρχεία που πρέπει να μας στείλετε σε ένα αρχείο. Πηγαίνετε στον home directory του λογαριασμού σας. Φτιάξτε εκεί με την εντολή *mkdir* έναν κατάλογο με όνομα project_1_omada_<AEM_omadas>. Για παράδειγμα, η ομάδα με ΑΕΜ 456 123 789 θα πρέπει να δώσει την εντολή *mkdir project_1_omada_123_456_789*. Αντιγράψτε (με την εντολή *cp*) το patch που φτιάξατε σε αυτό τον κατάλογο με την εντολή. Π.χ. η ομάδα 123_456_789 θα δώσει την εντολή: *cp /usr/src/patch_1 /project_1_omada_123_456_789*.

Κάντε το ίδιο για τα αρχεία roots.c, roots.h, find_roots_lib.c καθώς επίσης και για τους φακέλους project1_module και sysfs_module. Μπείτε σε κάθε φάκελο project1_module και sysfs_module, και δώστε την εντολή *make clean*, ώστε να σβηστούν τα αρχεία που παράχθηκαν κατά τη μεταγλώττιση του εκάστοτε module και να μείνει μόνο ο κώδικας και το Makefile. **ΜΗΝ ΞΕΧΑΣΕΤΕ ΑΥΤΟ ΤΟ ΒΗΜΑ!!!**

Επίσης, δημιουργήστε μέσα στον κατάλογο ένα αρχείο με όνομα README.txt στον οποίο θα γράψετε τα ονόματα, ΑΕΜ και e-mail των μελών της ομάδας, καθώς και αναλυτικές οδηγίες για τη μεταγλώττιση της εφαρμογής find_roots_lib και του module. Μπορείτε να συμπεριλάβετε στο αρχείο και οτιδήποτε άλλο θέλετε να έχουμε υπόψη μας κατά τη διόρθωση.

Ακολούθως, πηγαίνετε στον πηγαίο κατάλογο του λογαριασμού σας και από εκεί δώστε την εντολή

tar -cvf project_1_omada_<AEM_omadas>.tar project_1_omada_<AEM_omadas>

Για παράδειγμα, η ομάδα με μέλη με ΑΕΜ 456 123 789 θα δώσει την εντολή

tar -cvf project_1_omada_123_456_789.tar project_1_omada_123_456_789

Με την εντολή tar θα πακεταριστούν τα περιεχόμενα του καταλόγου σε ένα αρχείο με κατάληξη .tar.

Τέλος, δώστε την εντολή *bzip2 project_1_omada_<AEM_omadas>.tar*. Θα δημιουργηθεί ένα αρχείο με κατάληξη .bz2, το οποίο περιέχει συμπιεσμένο το αρχείο με κατάληξη .tar. Το αρχείο με κατάληξη .bz2 είναι αυτό που θα πρέπει να ανεβάσετε στο e-class. Για παράδειγμα, η ομάδα με μέλη με ΑΕΜ 456 123 789 θα δώσει την εντολή

bzip2 project_1_omada_123_456_789.tar

Αφού φτιαχτεί το αρχείο, ελέγξτε το μέγεθός του. Αν το μέγεθος είναι αδικαιολόγητα μεγάλο (θυμηθείτε ότι το αρχείο στην ουσία περιέχει μερικά πολύ μικρά αρχεία κειμένου) έχετε κάνει κάτι λάθος (το πιθανότερο είναι ότι έχετε ξεχάσει το βήμα make distclean ή έχετε συμπεριλάβει και εκτελέσιμα αρχεία).

Η εργασία θα πρέεπι να υποβληθεί από *ένα μόνο μέλος*, εκ μέρους όλης της ομάδας.