Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Λειτουργικά Συστήματα (ECE318)

Ακαδημαϊκό Έτος 2020-2021

Οδηγός Προσθήκης Κλήσεων Συστήματος στον Πυρήνα του Λειτουργικού Linux

Τελευταία Ενημέρωση: 16 Μαρτίου 2021

Περιεχόμενα

1 Εισαγωγή

Σε κάθε σύγχρονο λειτουργικό σύστημα, ο πυρήνας είναι υπεύθυνος για να προσφέρει μία σειρά από υπηρεσίες προς τις εφαρμογές επιπέδου χρήστη. Οι εφαρμογές μπορούν να ζητήσουν αυτές τις υπηρεσίες μέσω των *κλήσεων συστήματος (system calls)*. Οι κλήσεις συστήματος είναι ιδιαίτερα σημαντικές στο λειτουργικό σύστημα Linux. Το Linux βασίζεται σε αρχιτεκτονική *μονολιθικού πυρήνα*, όπου το λειτουργικό σύστημα στο σύνολό του εκτελείται στον χώρο του πυρήνα (*kernel space*). Έτσι, οι κλήσεις συστήματος είναι ο πρωτεύων τρόπος[1] για την επικοινωνία των εφαρμογών με το λειτουργικό σύστημα.

Οι κλήσεις συστήματος μπορούν να έχουν μηδέν ή περισσότερα ορίσματα και μία επιστρεφόμενη τιμή τύπου *long* που χρησιμοποιείται για να υποδηλώσει την επιτυχημένη ή όχι εκτέλεση της κλήσης συστήματος. Όσον αφορά στον τρόπο χρήσης μιας κλήσης συστήματος, αυτή πραγματοποιείται είτε μέσω της C library (για τις πιο συνήθεις κλήσεις συστήματος), είτε απευθείας χρησιμοποιώντας κάποια macros που προσφέρει το λειτουργικό σύστημα.

Η εκτέλεση κλήσης συστήματος από μία εφαρμογή χρήστη χωρίζεται σε τέσσερις φάσεις. Στην πρώτη φάση, η εφαρμογή που ενεργοποιεί την κλήση συστήματος μπαίνει σε αναμονή. Στη συνέχεια, ο πυρήνας είναι υπεύθυνος για τον έλεγχο του τύπου της κλήσης συστήματος καθώς και των δικαιωμάτων πρόσβασης που έχει η καλούσα διεργασία. Στην τρίτη φάση – και στην περίπτωση που η κλήση είναι έγκυρη – ο πυρήνας μεταφέρει τα όποια δεδομένα έχουν δοθεί ως ορίσματα στη συνάρτηση στον δικό του χώρο δεδομένων και καλεί τη συνάρτηση που υλοποιεί την κλήση συστήματος. Τέλος, μετά την επιστροφή της συνάρτησης (τέταρτη φάση), ο πυρήνας μεταφέρει (σε περίπτωση που υπάρχουν) τα αποτελέσματα επιστροφής στον χώρο δεδομένων της καλούσας διεργασίας και η τελευταία μπορεί πλέον να συνεχίσει τη εκτέλεσή της.

Το κύριο έργο για την υλοποίηση μίας κλήσης συστήματος στον πυρήνα του Linux είναι η συγγραφή του κώδικα της συνάρτησης που υλοποιεί τη λειτουργικότητα της κλήσης συστήματος και η προσθήκη αυτής στις διαθέσιμες κλήσεις συστήματος του λειτουργικού. Το δευτερεύον τμήμα στη διαδικασία υλοποίησης μίας κλήσης συστήματος είναι η υλοποίηση μίας βιβλιοθήκης επιπέδου χρήστη (*wrapper library*), η οποία θα προσφέρει μία συνάρτηση που θα αναλαμβάνει να εκτελέσει την κλήση συστήματος, παρέχοντας την απαραίτητη αφαίρεση στις αντίστοιχες εφαρμογές.

Στο υπόλοιπο τμήμα αυτού του οδηγού περιγράφεται η διαδικασία για την προσθήκη μίας νέας κλήσης συστήματος στον πυρήνα του Linux και η δημιουργία της wrapper library. Ως παράδειγμα, θα υλοποιήσουμε την κλήση συστήματος *hello_syscall()*, η οποία θα εκτυπώνει το μήνυμα ”Hello

[1] Τα kernel modules μπορούν να θεωρηθούν ως δευτερεύων τρόπος επικοινωνίας.

from kernel space”.

1.1 Υλικό για Μελέτη

Κεφάλαιο 5 του βιβλίου ”Linux Kernel Development” (3η έκδοση).

2 Προσθήκη Κλήσης Συστήματος

Το πρώτο βήμα στη διαδικασία προσθήκης μίας νέας κλήσης συστήματος είναι η προσθήκη του κώδικα που υλοποιεί την κλήση συστήματος στον πηγαίο κώδικα του πυρήνα. Η προσθήκη του κώδικα μπορεί να γίνει είτε σε ένα υπάρχον, είτε σε ένα νέο αρχείο κώδικα, στον κατάλληλο φάκελο ανάλογα με τη λειτουργικότητα που προσφέρει η κλήση συστήματος. Για παράδειγμα, ο κώδικας μίας κλήσης συστήματος που προσφέρει λειτουργικότητα δικτύωσης θα προστεθεί στον φάκελο /net. Σε περίπτωση που προστεθεί ένα νέο αρχείο κώδικα, θα πρέπει να γίνουν οι κατάλληλες αλλαγές στο αντίστοιχο Makefile ώστε το νέο αρχείο να προστεθεί στη διαδικασία μεταγλώττισης. Για τους σκοπούς αυτού του οδηγού, θα προχωρήσουμε στη δημιουργία ενός νέου αρχείου κώδικα. Δημιουργήστε ένα αρχείο με όνομα *my_syscall.c*, εντός του φακέλου /usr/src/linux-5.4.86-dev/kernel και προσθέστε το κείμενο ”my_syscall.o” στο τέλος της γραμμής 13 του αντίστοιχου Makefile. Ο κώδικας της συνάρτησης που υλοποιεί την κλήση συστήματος παρουσιάζεται παρακάτω.

Κώδικας νέου χειριστή κλήσης συστήματος

```
#include <linux/kernel.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE0(hello_syscall) {

    printk(”Hello from kernel space\n”);
    return(0);
}
```

Το *SYSCALL_DEFINE0* είναι ένα macro το οποίο δηλώνει μία κλήση συστήματος χωρίς ορίσματα. Μετά την επεξεργασία από τον προ-επεξεργαστή, η γραμμή αυτή αντικαθίσταται από την ακόλουθη:

```
asmlinkage long hello_syscall(void)
```

Το προσδιοριστικό asmlinkage ενημερώνει τον μεταγλωττιστή ότι τα ορίσματα της συνάρτησης βρίσκονται στη στοίβα, ενώ ο τύπος long στην επιστρεφόμενη τιμή χρησιμοποιείται για λόγους συμβατότητας μεταξύ 32-bit και 64-bit αρχιτεκτονικών. Τέλος, αξίζει να σημειώσουμε τη σύμβαση ονοματοδοσίας στον πυρήνα του Linux. Όπως παρατηρούμε, η κλήση συστήματος με όνομα hello_syscall υλοποιείται από τον χειριστή (συνάρτηση) sys_hello_syscall.

Το επόμενο βήμα είναι η δήλωση της επικεφαλίδας (prototype) της συνάρτησης. Αυτή θα τοποθετηθεί στο αρχείο syscalls.h, το οποίο βρίσκεται στον φάκελο /usr/src/linux-5.4.86-dev/include/linux. Ανοίξτε αυτό το αρχείο και τοποθετήστε την παρακάτω γραμμή κώδικα στο τέλος του αρχείου:

```
asmlinkage long sys_hello_syscall(void);
```

Στη συνέχεια, θα πρέπει να δηλώσετε ένα μοναδικό κωδικό για τη νέα κλήση συστήματος. Σε κάθε κλήση συστήματος στον πυρήνα του Linux έχει δοθεί ένας μοναδικός κωδικός αριθμός (*syscall* number). Όταν μία εφαρμογή χρήστη ζητήσει τις υπηρεσίες μιας κλήσης συστήματος, αυτός ο μοναδικός αριθμός χρησιμοποιείται ώστε να προσδιορισθεί ποια συνάρτηση θα κληθεί. Το αρχείο /usr/src/linux-5.4.86-dev/arch/x86/entry/syscalls/syscall_64.tbl περιέχει τους κωδικούς όλων των κλήσεων συστήματος για την αρχιτεκτονική x86.[2] Σε αυτό το αρχείο θα πρέπει να προστεθεί ο κωδικός για τη νέα κλήση συστήματος.

Listing 1: Οι πρώτες εγγραφές του αρχείου syscall_64.tbl.

```
0       common  read            __x64_sys_read
1       common  write           __x64_sys_write
2       common  open            __x64_sys_open
3       common  close           __x64_sys_close
4       common  stat            __x64_sys_newstat
5       common  fstat           __x64_sys_newfstat
```

Ο κώδικας 1 παρουσιάζει τις πρώτες εγγραφές από το αρχείο syscall_64.tbl. Η κλήση συστήματος της πρώτης γραμμής αντιστοιχεί στον αριθμό 0, η κλήση συστήματος της δεύτερης γραμμής αντιστοιχεί στον αριθμό 1, κ.ο.κ. Παρατηρήστε τη δομή του αρχείου: η κάθε γραμμή ορίζει ένα μοναδικό system call χρησιμοποιώντας τέσσερις στήλες. Στην πρώτη στήλη βρίσκεται ο κωδικός της συνάρτησης και χρησιμοποιείται για τη δεικτοδότηση (indexing) σε ένα πίνακα με δείκτες προς

[2] Αυτή η διαδικασία θα πρέπει να πραγματοποιηθεί για κάθε αρχιτεκτονική που θα υποστηρίζει τη συγκεκριμένη κλήση συστήματος.

τις συναρτήσεις που υλοποιούν τις κλήσεις συστήματος, προκειμένου για οποιαδήποτε κλήση συστήματος να κληθεί η σωστή συνάρτηση-χειριστής που την υλοποιεί. Στη δεύτερη στήλη βρίσκεται ο τύπος του Application Binary Interface (ABI), το οποίο ορίζει τη διεπαφή μεταξύ της κλήσης συστήματος και του λειτουργικού. Η διεπαφή προσδιορίζει όλες τις λεπτομέρειες αναφορικά με το πως πρέπει να εκτελεστεί η συνάρτηση που χειρίζεται την κλήση συστήματος. Εν προκειμένω, εφόσον εμείς θα ασχοληθούμε με αρχιτεκτονική x86-64bit χρησιμοποιούμε την διεπαφή common, που ορίζει συμβατότητα μεταξύ x32 & x64 bit. Η τρίτη στήλη είναι το όνομα της κλήσης συστήματος και η τέταρτη στήλη είναι το σημείο εισόδου ή αλλιώς η συνάρτηση-χειριστής που υλοποιεί την κλήση συστήματος.

Για το τρέχον παράδειγμα (κώδικας 2) πρέπει να ορίσουμε μία νέα κλήση συστήματος γράφοντας μία νέα γραμμή στο αρχείο syscalls_64.tbl μετά το τέλος των x64 syscalls, ως εξής:

```
433     common  fspick                  __x64_sys_fspick
434     common  pidfd_open              __x64_sys_pidfd_open
435     common  clone3                  __x64_sys_clone3/ptregs
436     common  hello_syscall           __x64_sys_hello_syscall
            . . .
512     x32     rt_sigaction            __x32_compat_sys_rt_sigaction
```

Αρχικά, παρατηρήσαμε πως το τελευταίο syscall x64 στη γραμμή 359 του αρχείου syscalls_64.tbl ξεκινούσε με το αναγνωριστικό 435 στην πρώτη της στήλη οπότε εμείς επιλέξαμε τον αμέσως μεγαλύτερο αριθμό 436 σαν αναγνωριστικό της κλήσης συστήματος. Στην συνέχεια γράψαμε το όνομα της συνάρτησης και τέλος την συνάρτηση-διαχειριστή.

Αφού μεταγλωττίσουμε το αρχείο my_syscall.c που γράψαμε νωρίτερα, θα εμφανιστεί στο κάτω μέρος του αρχείου unistd_64.h στο φάκελο /usr/src/linux-5.4.86-dev/arch/x86/include/generated/uapi/asm ένα #define το οποίο θα μας δίνει την ικανότητα να χρησιμοποιούμε την νέα κλήση του συστήματος σε εφαρμογές που θα εκτελούνται στο user-space.

```
#ifndef _ASM_X86_UNISTD_64_H
        . . .
#define __NR_fsmount 432
#define __NR_fspick 433
#define __NR_pidfd_open 434
#define __NR_clone3 435
#define __NR_hello_syscall 436

#endif /* _ASM_X86_UNISTD_64_H */
```

Για να είναι ορατός ο νέος αριθμός κλήσης συστήματος στις εφαρμογές χρήστη, θα πρέπει να αντιγράψετε το αρχείο unistd_64.h στον φάκελο /usr/include/x86_64-linux-gnu/asm, ο οποίος συμπεριλαμβάνεται στους φακέλους στους οποίους ψάχνει ο μεταγλωττιστής για header files, όταν μεταγλωττίζουμε μια εφαρμογή επιπέδου χρήστη.

Μετά από αυτή την αλλαγή, θα πρέπει να μεταγλωττίσετε τον πυρήνα ώστε η νέα κλήση συστήματος να ενσωματωθεί στο εκτελέσιμο. Για τη μεταγλώττιση, μπορείτε να ακολουθήσετε τις οδηγίες από τον οδηγό ”Οδηγίες για τις Εργαστηριακές Ασκήσεις”, ο οποίος είναι διαθέσιμος στη σελίδα του εργαστηρίου. Μετά την επιτυχημένη μεταγλώττιση του πυρήνα, ένας εύκολος τρόπος να ελέγξετε εάν η διαδικασία ενσωμάτωσης έχει ολοκληρωθεί με επιτυχία είναι εκτελώντας την εντολή *grep ”hello_syscall” System.map* μέσα στον κατάλογο που βρίσκεται ο κώδικας του λειτουργικού. Το αρχείο System.map περιέχει όλες τις κλήσεις συστήματος, οπότε η εντολή θα πρέπει να μπορεί να εντοπίσει και να σας εμφανίσει το όνομα της συνάρτησης hello_syscall.

Τέλος, θα πρέπει να εγκαταστήσετε τον νέο πυρήνα και να επανεκκινήσετε το σύστημά σας. Και αυτή η διαδικασία περιγράφεται στον οδηγό που προαναφέρθηκε.

3 Χρήση της Νέας Κλήσης Συστήματος

Σημείωση: Τα αρχεία του κώδικα επιπέδου χρήστη δεν θα πρέπει να τοποθετηθούν στο source tree του πυρήνα!

Στη γενική περίπτωση, μία εφαρμογή επιπέδου χρήστη δεν απαιτείται να χρησιμοποιεί απευθείας μία κλήση συστήματος. Η χρήση πραγματοποιείται έμμεσα, μέσω συναρτήσεων που προσφέρει η C library. Για να αποφύγουμε την ανάγκη να αλλάξετε — πλέον όλων των υπολοίπων — και τη C library, σας παρουσιάζουμε παρακάτω τον τρόπο απευθείας χρήσης μιας κλησης συστήματος από τις εφαρμογές, χωρίς τη μεσολάβηση της C library.

Το λειτουργικό σύστημα Linux επιλύει αυτό το πρόβλημα προσφέροντας στις εφαρμογές επιπέδου χρήστη κάποια macros, τα οποία μπορούν να εκτελέσουν όλες τις απαραίτητες ενέργειες που απαιτούνται για τη χρήση μίας κλήσης συστήματος. Τα macros που προσφέρονται είναι τα ακόλουθα:

- syscall(int number, arg1, arg2, ...)
- _syscallX(type, name, type1, arg1, ..., typeX, argX), όπου X ο αριθμός των ορισμάτων της συνάρτησης (0...6)

Το πρώτο macro καλεί τη συνάρτηση με αριθμό κλήσης number και τα αντίστοιχα ορίσματα. Για να χρησιμοποιήσουμε αυτό το macro, ο αριθμός κλήσης της συνάρτησης θα πρέπει να έχει δηλωθεί στο αρχείο syscalls.h και το macro να υποστηρίζεται από την έκδοση της C library που είναι εγκατεστημένη στο σύστημά μας. Αξίζει να σημειωθεί ότι το macro syscall είναι και αυτό που προτείνεται για την κλήση συναρτήσεων συστήματος από εφαρμογές επιπέδου χρήστη.

Όσον αφορά το δεύτερο macro, ο προ-επεξεργαστής το αντικαθιστά στον κώδικα της εφαρμογής με κώδικα assembly που αναλαμβάνει να εκτελέσει την κλήση συστήματος. Αυτή η διαδικασία επιτρέπει την κλήση μίας συνάρτησης συστήματος με το όνομά της, εφόσον το macro έχει ορισθεί στον κώδικα της εφαρμογής.

Με χρήση των προαναφερθέντων macros, μία εφαρμογή επιπέδου χρήστη μπορεί να καλέσει μία συνάρτηση συστήματος. Η κλήση μπορεί να πραγματοποιηθεί είτε με κώδικα που εμπεριέχεται στον κώδικα της εφαρμογής είτε μέσω μίας βιβλιοθήκης που παρέχει μία wrapper συνάρτηση (όμοια με την C library) για την κλήση της συνάρτησης συστήματος. Για τους σκοπούς του παρόντος οδηγού, θα εκτελέσουμε μία κλήση συστήματος μέσω μίας βιβλιοθήκης επιπέδου χρήστη, χρησιμοποιώντας το macro *syscall*.

Δημιουργήστε στο home directory τα αρχεία *syscall_wrapper.h* και *syscall_wrapper.c*, με τον παρακάτω κώδικα:

Περιεχόμενα αρχείου syscall_wrapper.h

```
int hello_syscall_wrapper(void);
```

Περιεχόμενα αρχείου syscall_wrapper.c

```
#include <sys/syscall.h>
#include <unistd.h>
#include "syscall_wrapper.h"

int hello_syscall_wrapper(void) {
    return( syscall(__NR_hello_syscall) );
}
```

Όπως παρατηρούμε, η συνάρτηση *hello_syscall_wrapper* είναι υπεύθυνη για την ενεργοποίηση της κλήσης συστήματος. Επίσης, η συνάρτηση δεν λαμβάνει κάποιο όρισμα (λογικό, εφόσον ούτε και η αντίστοιχη συνάρτηση που υλοποιεί την κλήση συστήματος λαμβάνει κάποιο όρισμα). Σε διαφορετική περίπτωση, η συνάρτηση *hello_syscall_wrapper* θα έπρεπε να λαμβάνει αντίστοιχο

αριθμό ορισμάτων.

Στη συνέχεια, μεταγλωττίστε το αρχείο syscall_wrapper.c και δημιουργήστε τη στατικά διασυνδεόμενη βιβλιοθήκη με όνομα *libhello_syscall_wrapper.a*.

Εκτελώντας τα παραπάνω βήματα, η κλήση συστήματος μπορεί εύκολα να χρησιμοποιηθεί, όπως φαίνεται στον ακόλουθο κώδικα. Απαραίτητη προϋπόθεση είναι προφανώς η σύνδεση της εφαρμογής με την βιβλιοθήκη που δημιουργήθηκε στο προηγούμενο βήμα κατά τη διαδικασία παραγωγής του εκτελέσιμου της εφαρμογής.

Κώδικας για τη χρήση της κλήσης συστήματος

```
#include "syscall_wrapper.h"

int main(int argc, char *argv[]) {
    hello_syscall_wrapper();

    return(0);
}
```