1η Εργαστηριακή Άσκηση

Στα πλαίσια της παρούσας εργαστηριακής άσκησης καλείστε να προσθέσετε στο σύστημα αρχείων του MINIX υποστήριξη για λίστες ελέγχου προσπέλασης (ACL ή access control list). Το υπάρχον σύστημα αρχείων του MINIX – όπως και κάθε UNIX λειτουργικό σύστημα – υποστηρίζει έλεγχο προσπέλασης βασισμένο στον προσδιορισμό των βασικών δικαιωμάτων προσπέλασης (read/write/execute) για τα τρία προκαθορισμένα σύνολα χρηστών (owner/group/other). Τα δικαιώματα αυτά ισχύουν για απλά αρχεία, καταλόγους αρχείων (directories) και αρχεία συσκευών (device files). Η προσθήκη ACL δίνει τη δυνατότητα στο διαχειριστή και τους χρήστες ενός συστήματος να προσδιορίσουν την πρόσβαση στα αρχεία του συστήματος με μεγαλύτερη σαφήνεια και ευελιξία προσφέροντας ένα λεπτότερα καταμερισμένο έλεγχο δικαιωμάτων.

Η πλήρης υποστήριξη ACL στο MINIX απαιτεί τα εξής:

1. την υλοποίηση των κλήσεων συστήματος (system calls) που θα παρέχουν την δυνατότητα στον χρήστη να προσπελάσει/εισάγει/διαγράψει ACL σε ένα αρχείο
2. την ενσωμάτωση των νέων μηχανισμών ελέγχου προσπέλασης σε όλα τα υποσυστήματα του λειτουργικού συστήματος
3. την υλοποίηση εργαλείων που μπορούν να χρησιμοποιηθούν από τους χρήστες για τον έλεγχο των ACL από την γραμμή εντολών (command line)
4. την υποστήριξη των ACL από τα υπάρχοντα εργαλεία του συστήματος (π.χ. *cp, tar, fsck*)

Στα πλαίσια της άσκησης θα υλοποιηθούν μόνο τα 1 και 2 καθώς και οι απαραίτητες τροποποιήσεις στο εργαλείο *fsck* που είναι υπεύθυνο για τον έλεγχο και την επιδιόρθωση του συστήματος αρχείων.

Σκοπός της άσκησης αυτής είναι:

- η εκμάθηση της διαδικασίας υλοποίησης και εισαγωγής μίας νέας κλήσης συστήματος στο λειτουργικό σύστημα MINIX.
- η εξοικείωση με τον τρόπο διαδιεργασιακής επικοινωνίας (IPC) που χρησιμοποιεί το MINIX, δηλαδή το πέρασμα μηνυμάτων με χρήση rendezvous.
- η κατανόηση των δομών και της λειτουργίας των βασικότερων υποσυστημάτων του FS Server και γενικότερα του συστήματος αρχείων του MINIX.
- η κατανόηση και η εξοικείωση με την οργάνωση του πηγαίου κώδικα του MINIX και ειδικότερα με τον FS Server, την libc αλλά και το πλήθος των header files που βρίσκονται διασκορπισμένα σε διάφορα directories του source code tree.

1. Η κλήση συστήματος *acl()*

Στη συνέχεια παρατίθεται η ακριβής περιγραφή του interface της κλήσης συστήματος αλλά και της λειτουργικότητας που θα παρέχει το ACL σύστημα που καλείστε να υλοποιήσετε[1].

```
#include <sys/acl.h>
#include <sys/types.h>

int acl(char *name, int cmd, int nacls, acl_t *aclbufp)
```

Περιγραφή

Η `acl()` χρησιμοποιείται για την διαχείριση της ACL ενός αρχείου

[1] Η λειτουργικότητα που περιγράφεται στη συνέχεια, αν και εμφανίζει ομοιότητες με αντίστοιχες υλοποιήσεις που προσφέρουν αρκετά σύγχρονα Unix λειτουργικά συστήματα, είναι κατάλληλα τροποποιημένη για τους σκοπούς της παρούσας εργαστηριακής άσκησης. Σε καμία περίπτωση δεν είναι επαρκής στα πλαίσια ενός λειτουργικού συστήματος γενικού σκοπού.

`name` δείχνει στο *path* του αρχείου του οποίου η ACL θα επεξεργαστεί.

`aclbufp` είναι ένας δεικτής στην πρώτη ACL εγγραφή ενός *array* από δομές `acl_t`. Η δήλωση αυτής της δομής βρίσκεται στο αρχείο `<sys/acl.h>` και είναι η εξής:

```
typedef struct acl {
    long        acl_flags;  /* acl flags */
    uid_t       acl_id;     /* user/group id */
    mode_t      acl_perm;   /* permission bits */
} acl_t;
```

Οι δυνατές τιμές του μέλους `acl_flags` είναι οι ακόλουθες:

`ACL_UID` η μεταβλητή `acl_perm` περιέχει τα *permissions* για τον χρήστη με *user id* `acl_id`.

`ACL_GID` η μεταβλητή `acl_perm` περιέχει τα *permissions* για τους χρήστες που ανήκουν στο *group* με *group id* `acl_id`.

Η τιμή του μέλους `acl_perm` είναι η ORed τιμή οποιουδήποτε υποσυνόλου των σταθερών ACL_RBIT, ACL_WBIT, ACL_XBIT, οι οποίες ορίζονται στο αρχείο <sys/acl.h> και αντιστοιχούν στην παροχή του δικαιώματος ανάγνωσης, εγγραφής και εκτέλεσης αντίστοιχα.

`nacls` προσδιορίζει το πλήθος των ACL εγγραφών που δείχνει ο pointer `aclbufp`.

`cmd` Οι δυνατές τιμές της παραμέτρου `cmd` είναι οι ακόλουθες:

`ACL_SET` `nacls` σε πλήθος ACL εγγραφές, από τον *buffer* `aclbufp` αποθηκεύονται στην ACL του αρχείου. Οποιεσδήποτε υπάρχουσες ACL εγγραφές θα αντικατασταθούν από τις νέες. Η εκτέλεση αυτής της λειτουργίας είναι δυνατή μόνο από τον τρέχοντα ιδιοκτήτη του αρχείου ή από τον υπερχρήστη. Όλοι οι κατάλογοι στην διαδρομή του `name` θα πρέπει να είναι προσπελάσιμοι (*execute bit*).

`ACL_GET` στον *buffer* `aclbufp` αντιγράφονται το πολύ `nacls` σε πλήθος ACL εγγραφες από την ACL του αρχείου. Για την εκτέλεση αυτής της λειτουργίας δεν απαιτείται δικαίωμα ανάγνωσης στο αρχείο αλλά όλοι οι κατάλογοι στην διαδρομή του `name` θα πρέπει να είναι προσπελάσιμοι.

`ACL_CLR` Όλες οι ACL εγγραφές του αρχείου διαγράφονται. Το *block* στο οποίο αποθηκεύονται οι ACL εγγραφές θα πρέπει να αποδεσμευτεί. Για την εκτέλεση αυτής της λειτουργίας δεν απαιτείται δικαίωμα ανάγνωσης στο αρχείο αλλά όλοι οι κατάλογοι στην διαδρομή του `name` θα πρέπει να είναι προσπελάσιμοι.

`ACL_CNT` Επιστρέφεται το πλήθος των ACL εγγραφών του αρχείου. Για την εκτέλεση αυτής της λειτουργίας δεν απαιτείται δικαίωμα ανάγνωσης στο αρχείο αλλά όλοι οι κατάλογοι στην διαδρομή του `name` θα πρέπει να είναι προσπελάσιμοι

Η λειτουργία `ACL_SET` επιτυγχάνει μόνο αν ισχύουν όλες οι παρακάτω προτάσεις για τις εγγραφές που περιέχονται στον *buffer* `aclbufp`:

- Δεν υπάρχουν διπλές εγγραφές. Μία εγγραφή είναι διπλή αν είναι του ιδίου τύπου (*user* ή *group*) και περιέχει το ίδιο `acl_id`.
- Το πλήθος των εγγραφών είναι το ≤ του μέγιστου πλήθος ACL εγγραφών που μπορούν να αποθηκευτούν σε ένα block. Το μέγιστο αυτό πλήθος ορίζεται ως η σταθερά `NACLS` στο

αρχείο <sys/acl.h>.

- Όλες οι εγγραφές περιέχουν έγκυρες τιμές των πεδίων acl_id και acl_flags. Η μηδενική τιμή για το πεδίο acl_flags δεν θεωρείται έγκυρη.

Επιστρεφόμενη τιμή

Στην περίπτωση που η κλήση της acl() πετύχει επιστέφει το πλήθος των ACL εγγραφών που αντιγράφηκαν στον *buffer* aclbufp για cmd ACL_GET, ACL_CNT και 0 για cmd ACL_SET, ACL_CLR. Σε περίπτωση αποτυχίας η acl() επιστρέφει -1 και αποθηκεύει τον κωδικό λάθους στην *global* μεταβλητή errno.

Κωδικοί λαθους

Αν η acl() αποτύχει η *global* μεταβλητή errno περιέχει μία από τις παρακάτω τιμές:

EACCES	ο χρήστης δεν έχει δικαίωμα προσπέλασης σε κάποιο τμήμα του pathname
EINVAL	η τιμής της παραμέτρου cmd δεν είναι έγκυρη
EINVAL	η παράμετρος cmd είναι ACL_SET αλλά οι εγγραφές που δόθηκαν δεν ικανοποιούν τις προϋποθέσεις που περιγράφονται παραπάνω
EINVAL	η τιμή της παραμέτρου nacls είναι < 0 ή > NACLS
EIO	ένα I/O σφάλμα συνέβη κατά την αποθήκευση ή την ανάκτηση της ACL
EPERM	η παράμετρος cmd είναι ACL_SET ή ACL_CLR αλλά το *effective id* του χρήστη δεν είναι ίσο με αυτό του ιδιοκτήτη του αρχείου ή του superuser
EPERM	το αρχείο είναι ειδικό αρχείο τύπου *named pipe*
ENOENT	το αρχείο ή κάποιο τμήμα του *pathname* δεν υπάρχει
ENOENT	η παράμετρος cmd είναι ACL_CNT ή ACL_GET αλλά το αρχείο δεν έχει ACL *block*
EROFS	το αρχείο βρίσκεται σε *readonly* σύστημα αρχείων
ENOSYS	το σύστημα αρχείων δεν είναι MINIX *ver*. 2

Σχόλια

Η acl() υλοποιείται μόνο για σύστημα αρχείων MINIX έκδοσης 2. Για τα ειδικά αρχεία τύπου *named pipe* δεν υλοποιούνται ACL. Επίσης λόγω έλλειψης υποστήριξης από τα εργαλεία του συστήματος η ACL ενός αρχείου δεν διατηρείται κατά την αντιγραφή ενός αρχείου (με χρήση εντολών όπως *cp, tar etc.*).

Ο αλγόριθμός για τον έλεγχο προσπέλασης για ένα αρχείο που διαθέτει ACL είναι ο ακόλουθος:

Για τον υπολογισμό των δικαιωμάτων πρόσβασης μίας διεργασίας με *effective user id* (EUID) και *effective group id* (EGID) αντίστοιχα γίνονται οι παρακάτω έλεγχοι, με την σειρά που αναγράφονται. Άπαξ και καταστεί δυνατός ο προσδιορισμός των δικαιωμάτων μίας διεργασίας βάσει ενός από τους παρακάτω ελέγχους, οι υπόλοιποι έλεγχοι δεν πραγματοποιούνται.

1. Αν το EUID της διεργασίας είναι το ίδιο με το *user id* του ιδιοκτήτη του αρχείου ο έλεγχος πρόσβασης γίνεται βάσει των δικαιωμάτων ιδιοκτήτη (*owner permissions*).
2. Αν το EUID της διεργασίας είναι το ίδιο με το acl_id οποιασδήποτε ACL εγγραφής τύπου

`ACL_UID` του αρχείου ο έλεγχος πρόσβασης γίνεται βάσει των δικαιωμάτων της συγκεκριμένης ACL εγγραφής.

3. Αν το EGID της διεργασίας είναι το ίδιο με το *group id* του αρχείου ο έλεγχος πρόσβασης γίνεται βάσει των δικαιωμάτων ομάδας του αρχείου (*group permissions*).
4. Αν το EGID της διεργασίας είναι το ίδιο με το `acl_id` οποιασδήποτε ACL εγγραφής τύπου `ACL_GID` του αρχείου ο έλεγχος πρόσβασης γίνεται βάσει των δικαιωμάτων της συγκεκριμένης ACL εγγραφής.
5. Σε κάθε άλλη περίπτωση ο έλεγχος πρόσβασης γίνεται βάσει των δικαιωμάτων του αρχείου για τους λοιπούς χρήστες (*other permissions*).

Ο υπερχρήστης όπως και στην περίπτωση του συστήματος προστασίας χωρίς ACL έχει ειδικά προνόμια όσον αφορά το σύστημα αρχείων. Τα προνόμια αυτά παραμένουν ίδια και επεκτείνονται ως εξής:

- ο υπερχρήστης έχει δικαίωμα ανάγνωσης και εγγραφής σε κάθε αντικείμενο του συστήματος αρχείων.
- ο υπερχρήστης έχει δικαίωμα προσπέλασης (*execute bit*) σε κάθε κατάλογο αρχείων του συστήματος αρχείων.
- ο υπερχρήστης έχει δικαίωμα εκτέλεσης για κάποιο αρχείο - εκτός των καταλόγων - μόνο αν το *execution bit* είναι 1 σε μία τουλάχιστον εγγραφή της ACL ή ένα από τα τρία *execution bits* του υπάρχοντος συστήματος προστασίας είναι 1.

2. Ζητούμενα

1. Τροποποιείστε τον File System Server έτσι ώστε να υποστηρίζει την κλήση συστήματος `acl()` όπως ακριβώς αυτή προσδιορίστηκε στην προηγούμενη ενότητα. Η υλοποίηση σας θα πρέπει να ακολουθεί τους εξής κανόνες:

- ο έλεγχος προσπέλασης βασισμένος σε ACL υποστηρίζεται τόσο για αρχεία όσο και για καταλόγους
- το ID του μηνύματος για τη νέα κλήσης συστήματος είναι αυστηρά το `77`
- οι απαραίτητες δηλώσεις και σταθερές για την υποστήριξη της `acl()` αποθηκεύονται στο αρχείο `<sys/acl.h>`, τα περιεχόμενα του οποίου πρέπει να είναι αυστηρά αυτά που σας δίνονται στο Παράρτημα Β.
- για την αποστολή του μηνύματος της acl() ο χρήστης θα χρησιμοποιεί την κλήση της βιβλιοθήκης συστήματος που εσείς θα υλοποιήσετε στο αρχείο */usr/src/lib/other/acl.c* όπως περιγράφεται στο Παράρτημα Α. Ο τύπος μηνύματος που θα χρησιμοποιήσετε θα είναι ο *m1*. Η δομή του μηνύματος που θα χρησιμοποιήσετε θα είναι αυστηρά η εξής:
  - `m1_i1`: το μήκος του pathname συμπεριλαμβανομένου και του *null character*
  - `m1_i2`: η παράμετρος `cmd`
  - `m1_i3`: η παράμετρος `nacls`
  - `m1_p1`: η παράμετρος `name`
  - `m1_p2`: η παράμετρος `aclbufp`
- Η υλοποίηση σας θα περιορίζεται σε συστήματα αρχείων MINIX *ver.2*. Η ACL ενός αρχείου πρέπει να αποθηκεύετε σαν ένα *array* από `NACLS` σε πλήθος δομές τύπου `acl_t`. Το *array* αυτό αποθηκεύετε σε ένα μοναδικό block και ο δείκτης σε αυτό το block θα πρέπει να αποθηκεύεται στην τελευταία θέση για δείκτη σε *block* που διαθέτει ένα MINIX V2 *i-node*. Η θέση αυτή είναι δεσμευμένη για μελλοντική χρήση ως *triple indirect block pointer* αλλά δεν χρησιμοποιείται κατά αυτόν τον τρόπο στο υπάρχον σύστημα. Προσέξτε ότι τα ειδικά αρχεία τύπου *named pipe* χρησιμοποιούν αυτό το πεδίο για άλλο σκοπό.
- Όπως σημειώνεται και στην προηγούμενη ενότητα η λειτουργία `ACL_CLR` θα πρέπει να αποδεσμεύει το ACL *block*.
- Η υλοποίηση σας (για λόγους *interoperability*) δεν θα πρέπει να υποθέτει σε καμία περίπτωση ότι το *array* των δομών `acl_t` περιέχει τις ACL εγγραφές με κάποια ορισμένη σειρά.

2. Τροποποιείστε κατάλληλα το πρόγραμμα *fsck* (*/usr/src/commands/simple/fsck.c*) που είναι υπεύθυνο για τον έλεγχο της συνέπειας του συστήματος αρχείων έτσι ώστε να αναγνωρίζει την χρήση του ACL *block*.

3. Παραδοτέα

Στα πλαίσια της άσκησης θα πρέπει να παραδώσετε τα εξής:

1. Ένα αρχείο σε .zip *format* που θα περιέχει τα ακόλουθα:
   i. Όλα τα αρχεία του πηγαίου κώδικα του συστήματος που αλλάξατε/προσθέσατε. Ο κώδικας θα πρέπει να είναι επαρκώς σχολιασμένος. Στα αρχεία κώδικα που προϋπαρχαν, οι προσθήκες κώδικα θα πρέπει να εσωκλείονται στις εξής γραμμές σχολίων:

      ```
      /* OSLAB START */
      κώδικας που προσθέσατε
      /* OSLAB END */
      ```

      Αντίστοιχα τα τμήματα του αρχικού κώδικα που δεν θέλετε να περιληφθούν πρέπει να σχολιάζονται με την ένδειξη `CROP START` και `CROP END` στην αρχή και στο τέλος τους αντίστοιχα.

   ii. Το *binary* αρχείο του *kernel* με όνομα "`2.0.r<α/α ομάδας>`" (π.χ. για την ομάδα με α/α 17 το όνομα του αρχείου θα πρέπει να είναι `2.0.r17)`.

2. Τεχνική αναφορά (σε μορφή .pdf κατά προτίμηση) όπου θα εξηγείτε με παράθεση σύντομων σχολίων τις αλλαγές που πραγματοποιήσατε στον πηγαίο κώδικα του MINIX.

Τα παραπάνω θα πρέπει να σταλούν με e-mail στο *oslab@ceid.upatras.gr* πριν την καταληκτική ημερομηνία παράδοσης που θα ανακοινωθεί σύντομα στο *forum* του μαθήματος. Το e-mail θα πρέπει να αποσταλεί με *Subject* της μορφής "`OSLAB_1 <α/α ομάδας>`" (π.χ. για την ομάδα με α/α 17, Subject: `OSLAB_1 17`).

4. Υποδείξεις

1. Ακολουθείστε τη γενικότερη υπόδειξη που δίνεται για την υλοποίηση ασκήσεων τέτοιου τύπου και η οποία είναι η σταδιακή προσέγγιση του προβλήματος. Μην προσπαθείτε να υλοποιήσετε το σύνολο των ζητουμένων απευθείας. Κατασκευάστε ενδιάμεσες *λειτουργικές* εκδόσεις που υλοποιούν ένα υποσύνολο των ζητουμένων και των οποίων την ορθότητα μπορείτε εύκολα να επαληθεύσετε. Για παράδειγμα η συγκεκριμένη άσκηση μπορεί να χωριστεί στα εξής τμήματα:
   i. υλοποίηση ενός "dummy" system call που εκτυπώνει απλά ένα μήνυμα
   ii. επέκταση του (i) έτσι ώστε να εκτυπώνονται τα ορίσματα που περιέχονται στο μήνυμα του system call
   iii. επέκταση του (ii) έτσι ώστε να υποστηρίζεται η εισαγωγή ACL σε ένα αρχείο
   iv. επέκταση του (iii) έτσι ώστε να υποστηρίζεται η ανάκτηση της ACL ενός αρχείου
   v. επέκταση του (iv) έτσι ώστε να λαμβάνεται υπόψιν η ACL ενός αρχείου στην διαδικασία υπολογισμού των δικαιωμάτων πρόσβασης μίας διεργασίας σε αυτό

   Κάποια από τα παραπάνω βήματα μπορούν να ενοποιηθούν. Αυτό καθορίζεται ελεύθερα και δυναμικά από εσάς βάσει της εμπειρίας σας.

2. Επωφεληθείτε από το γεγονός ότι σας διατίθεται ο πηγαίος κώδικας πλήθους άλλων systems calls που ενδέχεται να υλοποιούν λειτουργίες που επιθυμείτε να χρησιμοποιήσετε και στην `acl()`. Συγκεκριμένα

αναζητείστε την υλοποίηση συναρτήσεων όπως:

i. `chmod()` που λαμβάνει ως όρισμα το path προς ένα αρχείο
ii. `stat()` που επιστρέφει το αποτέλεσμα της σε μία θέση μνήμης στο *address space* της διεργασίας

3. Διαβάστε προσεκτικά τις δομές που περιέχονται στα εξής αρχεία:

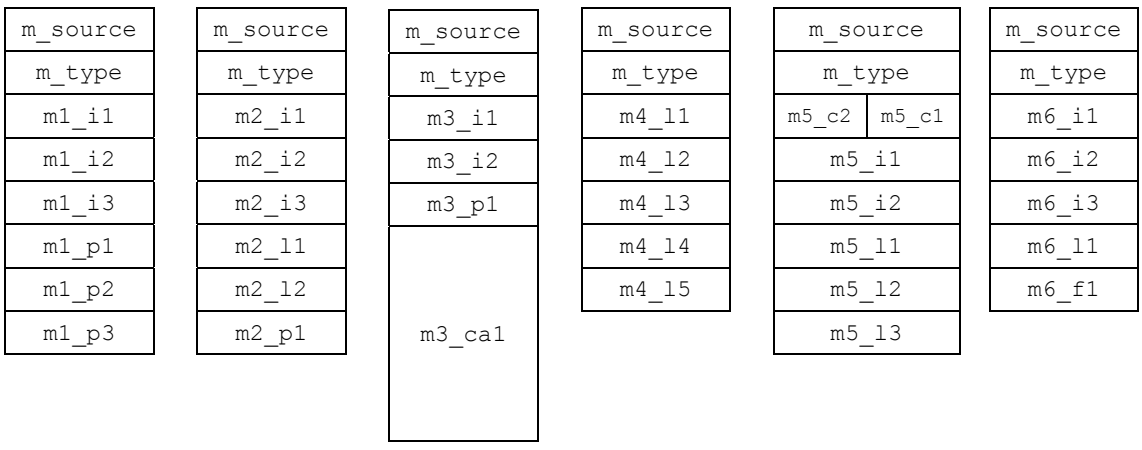
i. *fproc.h*: περιέχει τη δομή όπου διατηρεί ο FS Server την πληροφορία για κάθε διεργασία του συστήματος
ii. *inode.h*: περιέχει τη δομή όπου αποθηκεύονται τα εν χρήση *i-nodes* (i-node table)
iii. *buf.h*: περιέχει τη δομή όπου αποθηκεύεται ένα *cache block*
iv. *super.h*: περιέχει τη δομή όπου αποθηκεύεται το *super block* ενός συστήματος αρχείων

4. Η γενική ανάλυση της οργάνωσης του συστήματος αρχείων ενός λειτουργικού συστήματος Unix που παρατίθεται στο [1] μπορεί να αποτελέσει βασικό εργαλείο κατανόησης του τρόπου λειτουργίας του FS Server του MINIX. Επιπλέον στο [2] περιγράφεται με πληρότητα και λεπτομέρεια τόσο η σχεδίαση και λειτουργία όλων των τμημάτων του MINIX όσο και ο κώδικας υλοποίησης τους. Ιδιαίτερης σημασίας είναι οι υποενότητες 5.5.1, 5.5.2, 5.6.1 – 5.6.7, 5.7.

5. Υπάρχουν συνολικά έξι διαφορετικές εκδοχές της δομής message που χρησιμοποιείται από τον μηχανισμό περάσματος μηνυμάτων του MINIX, καθώς αυτή ορίζεται ως μία struct το ένα μέλος της οποίας είναι ένα `union`. Ο ορισμός των διαφορετικών εκδοχών του `union` καθώς και του `message` βρίσκονται στο αρχείο */usr/include/minix/type.h*. Προσέξτε ότι μετά από τους ορισμούς στο αρχείο περιέχονται `#define` δηλώσεις που ως σκοπό έχουν να κάνουν πιο απλή την πρόσβαση στα μέλη του message ανάλογα με το ποιος από τους εναλλακτικούς ορισμούς του union έχει επιλεχθεί. Έτσι είναι αρκετά απλό να διαπιστώσετε ότι αν επιλέξετε να χρησιμοποιήσετε το 1^{ο} είδος μηνύματος και θέλετε να αναφερθείτε στο στοιχείο `m1_i1` του union `mu` θα μπορούσατε αντί του δύσχρηστου και δυσμνημόνευτου `m.m_u.m_m1.m1_p1` να χρησιμοποιείται το σύντομο `m.m1_p1`. Επίσης προσέξτε ότι τα ονόματα των πεδίων του μηνύματος υποδηλώνουν και τον τύπο τους. Συγκεκριμένα τα ονόματα ακολουθούν τον εξής κανόνα:

m*x_yz,* όπου:
x = 1..6, δηλώνει το είδος του m_u που έχουμε επιλέξει
y = {i, p, l, f, ca, c}, δηλώνει τον τύπο δεδομένων του συγκεκριμένου μέλους
i : int
p: pointer
l : long
f: function
ca: character array
c: char
z : αύξων αριθμός του τύπου *y* στο συγκεκριμένο είδος μηνύματος

Συνοπτικά τα έξι διαφορετικά είδη μηνυμάτων απεικονίζονται στο ακόλουθο σχήμα:



6. Σαν *debugging tool* χρησιμοποιείστε την `printf()` (macro name της `printk()`). Η σύνταξη και τα ορίσματα της είναι ίδια με αυτά της συνάρτησης βιβλιοθήκης `printf()` με τη διαφορά ότι τα ορίσματα της εκτυπώνονται πάντοτε στο `tty0`.

7. Για τον έλεγχο της συνέπειας του συστήματος αρχείων ενός partition μπορείτε να εκτελείται την εντολή
   `fsck -r /dev/hd1a` (για το /)
   `fsck -r /dev/hd1c` (για το /usr)
   Πριν την τροποποίηση του κώδικα του *fsck* είναι βέβαιο ότι θα λαμβάνετε κάποιο λάθος για *out-of-range zones*. Το σφάλμα αυτό αρχικά μπορείτε να το αγνοήσετε επιλέγοντας "n*"* στην ερώτηση εάν θέλετε να διορθωθεί το πρόβλημα που εντοπίστηκε. Εντούτοις το *fsck* μπορεί να χρησιμοποιηθεί για να εντοπίσει και να επιδιορθώσει *blocks* που δεν έχετε αποδεσμεύσει από το *zone map* κατά τη δημιουργία/διαγραφή των ACLs.

8. Μην σβήσετε τον αρχικό kernel */minix/2.0.0* για να μπορέσετε, στη περίπτωση που κάτι απρόβλεπτο συμβεί λόγω του καινούργιου image, να ξεκινήσετε το σύστημα και να διορθώσετε το πρόβλημα που προέκυψε. Λόγω της φύσης της συγκεκριμένης άσκησης υπάρχει ο κίνδυνος ανεπανόρθωτης καταστροφής του συστήματος αρχείων. Διατηρείται τουλάχιστον ένα αντίγραφο ασφαλείας ολόκληρου του *virtual* δίσκου του *Bochs* στον οποίο εργάζεστε.

9. Ο κώδικας του FS έχει την ιδιαιτερότητα ότι απαιτεί μεγάλο πλήθος ελέγχων. Η ορθότητα της υλοποίησης εξαρτάται κατά πολύ από τον εντοπισμό και την ταξινόμηση των ελέγχων αυτών. Για την δική σας διευκόλυνση προσπαθήστε να καταγράψετε σε μία λίστα τους ελέγχους που πιστεύετε ότι είναι απαραίτητοι πριν ξεκινήσετε την υλοποίηση του συστήματος.

Παράρτημα Α

Προσθήκη κλήσης συστήματος στο λειτουργικό σύστημα MINIX

Η διαδικασία υλοποίησης μία νέας κλήσης συστήματος στο MINIX αποτελείται από δύο στάδια. Στο 1ο στάδιο υλοποιείται στον FS(MM)[2] Server η διαδικασία εξυπηρέτησης του system call. Στο 2ο στάδιο υλοποιείται η συνάρτηση βιβλιοθήκης που θα καλεί ο χρήστης μέσα από τα προγράμματα του (σε επίπεδο χρήστη) και η οποία αναλαμβάνει να αποστείλει το κατάλληλο μήνυμα στον FS Server. Μετά το πέρας της ακόλουθης διαδικασίας ο χρήστης θα έχει τη δυνατότητα να χρησιμοποιεί το νέο system call απλά συμπεριλαμβάνοντας στον πηγαίο κώδικα του προγράμματος του το header file *unistd.h* (*/usr/include/unistd.h*) Η ακόλουθη διαδικασία προϋποθέτει ότι έχετε κάνει login ως χρήστης *bin* και όχι ως *root* και αυτό διότι το σύστημα είναι εγκατεστημένο κατά τέτοιο τρόπο έτσι ώστε όλα τα source files του MINIX να ανήκουν στον χρήστη *bin*. Στη συνέχεια περιγράφονται συνοπτικά και τα δύο στάδια της διαδικασίας.

1. System Call Handler:

Βήμα 1ο: Ανοίξτε το αρχείο */usr/include/minix/callnr.h*. Αυξήσετε κατά ένα (78) το συνολικό πλήθος των system calls (`NCALLS`). Προσθέστε στο τέλος του αρχείου μία `#define` statement με το ID της system call που θα υλοποιήσετε (77).

Βήμα 2ο: Ανοίξτε το αρχείο */usr/src/fs/table.c*. Στο τέλος της ανάθεσης τιμών στο `call_vector` προσθέστε μία γραμμή με το όνομα της συνάρτησης εξυπηρέτησης που θα υλοποιήσετε στη συνέχεια. Το όνομα που θα δώσετε στη συνάρτηση δεν είναι απαραίτητα ίδιο με αυτό που θα καλεί ο χρήστης. Μάλιστα το *coding style* του Tanenbaum μάλλον υπαγορεύει να είναι της μορφής `do_systemcallname` (π.χ. `do_foobar`).

[2] Η διαδικασία για την προσθήκη μίας κλήσης συστήματος που εξυπηρετείται από τον MM είναι αντίστοιχη.

Βήμα 3ο: Ανοίξτε το αρχείο */usr/src/mm/table.c*. Επαναλάβετε την διαδικασία του 2ου βήματος με τη διαφορά ότι αντί του ονόματος της συνάρτησης διαχείρισης πρέπει να εισάγεται στο τέλος του `call_vector` το όνομα της ειδικής συνάρτησης εξυπηρέτησης `no_sys` που επιστρέφει απλά την τιμή `EINVAL`. Αυτό γίνεται διότι και τα δύο tables (των FS και MM) πρέπει να περιέχουν `NCALLS` στο πλήθος στοιχεία.

Βήμα 4ο: Ανοίξτε το αρχείο */usr/src/fs/proto.h*. Προσθέστε τη δήλωση της συνάρτησης εξυπηρέτησης που θα υλοποιήσετε. Το όρισμα της πρέπει να είναι τύπου `void` και η επιστρεφόμενη τιμή της τύπου `int`, όπως ίσως να παρατηρήσατε και στον ορισμό του τύπου του `call_vector`. Η δήλωση γίνεται μέσω της macro-εντολής _PROTOTYPE, που ορίζεται στο αρχείο */usr/include/ansi.h* και στην ουσία εξασφαλίζει την συμβατότητα μεταξύ compilers που υποστηρίζουν K&R ή ANSI C στυλ δήλωσης ορισμάτων. Το πρώτο όρισμα της macro είναι ο επιστρεφόμενος τύπος και το όνομα της συνάρτησης και το δεύτερο είναι τα ορίσματα της συνάρτησης μέσα σε παρενθέσεις και διαχωρισμένα με κώμα.

Βήμα 5ο: Στο βήμα αυτό πρέπει να υλοποιήσετε τη συνάρτηση εξυπηρέτησης που δηλώσατε στο *table.c*. Για ευκολία [3] ανοίξτε το αρχείο */usr/src/fs/misc.c* και στο τέλος του προσθέστε τον ορισμό (κυρίως σώμα) της συνάρτησης. Προσέξτε ότι η συνάρτηση πρέπει να δηλωθεί ως public με τη βοήθεια της macro `PUBLIC`.

2. Library Call:

Στο επίπεδο του χρήστη πρέπει να υλοποιηθεί μία συνάρτηση η οποία κατασκευάζει ένα μήνυμα βάσει των ορισμάτων που έδωσε ο χρήστης και στέλνει το μήνυμα στον κατάλληλο εξυπηρετητή του 3ου layer του MINIX (π.χ. FS Server). Η συνάρτηση αυτή πρέπει να ενσωματώθεί στη βασική βιβλιοθήκη συστήματος του MINIX (libc).

Βήμα 1ο: Ανοίξτε το αρχείο */usr/include/unistd.h*. Προσθέστε στο μπλοκ που περιέχεται μέσα στη συνθήκη `#ifdef _MINIX`, τον ορισμό της συνάρτησης που θα παρέχεται στον χρήστη για την χρήση της system call.

Βήμα 2ο: Δημιουργείστε ένα αρχείο στον κατάλογο */usr/src/lib/posix* με κατάλληλο όνομα (π.χ. *_foobar.c* για την system call *foobar*). Στο αρχείο προσθέστε κώδικα αντίστοιχο του ακόλουθου[4]:

```
#include <lib.h>
#define foobar _foobar
#include <unistd.h>

int foobar(…) {

message m;

/* αντιγραφή των παραμέτρων της συνάρτησης στα πεδία του  μήνυματος m */
...
...
...

return (_syscall(FS, FOOBAR, &m));
}
```

Η `_syscall` είναι μία συνάρτηση βιβλιοθήκης η οποία αναλαμβάνει να αποστείλει το μήνυμα `m` (3ο όρισμα) στον εξυπηρετητή που της δηλώνεται μέσω του 1ου ορίσματος και στη συνέχεια σε περίπτωση που η system call επέστρεψε με κάποιον κωδικό λάθους να επιστρέψει την τιμή –1 και να αποθηκεύσει τον κωδικό λάθους στην global μεταβλητή `errno`. Ο πηγαίος κώδικας της βρίσκεται στο αρχείο */usr/src/lib/other/syscall.c*.

[3] Διαφορετικά αν δημιουργήσετε ένα νέο αρχείο .c θα πρέπει να αλλάξετε κατάλληλα και το αρχείο Makefile που βρίσκεται στον κατάλογο */usr/src/fs*

[4] Στον κατάλογο */usr/src/lib/posix* περιέχονται πλήθος αρχείων από τα οποία μπορείτε να βρείτε χρήσιμα παραδείγματα

Βήμα 3ο: Δημιουργείστε ένα αρχείο στον κατάλογο */usr/src/lib/syscall* με κατάλληλο όνομα (π.χ. *foobar.s*). Το αρχείο αυτό είναι απαραίτητο για το linking της libc και απλά πρέπει να περιέχει τον ακόλουθο assembly κώδικα:

```
.sect .text
   .extern __foobar
   .define _foobar
   .align 2

   _foobar:
        jmp     __foobar
```

Βήμα 4ο: Ανοίξτε τα αρχεία */usr/src/lib/posix/Makefile* και */usr/src/lib/syscall/Makefile* και προσθέστε τους κατάλληλους κανόνες έτσι ώστε κατά την κατασκευή της libc να συμπεριληφθούν τα αρχεία *_foobar.c* και *foobar.s* αντίστοιχα. Η διαδικασία αυτή είναι αρκετά απλή αν παρατηρήσετε τους κανόνες που περιέχονται ήδη στα αρχεία *Makefile* και αφορούν τα υπόλοιπα αρχεία κώδικα που περιέχονται στον αντίστοιχο κατάλογο.

Kernel Compilation:

Για να χρησιμοποιήσετε την νέα κλήση συστήματος θα πρέπει να κατασκευάσετε ένα καινούργιο kernel image πέραν αυτού που ήδη περιέχεται προεγκατεστημένο στο σύστημα (*/minix/2.0.0*) και να το εγκαταστήσετε. Η διαδικασία αυτή είναι απλή, αν και χρονοβόρα τουλάχιστον την πρώτη φορά οπότε και θα γίνουν compile όλα τα αρχεία τόσο του kernel όσο και των fs, mm και init. Η διαδικασία είναι η ακόλουθη:

Βήμα 1ο: Αφού θέσετε ως τρέχοντα (εκτελώντας `cd`) τον κατάλογο */usr/src/tools*. Εκτελέστε τις εντολές:

```
make clean
make hdboot
```

Η δεύτερη εντολή όχι μόνο θα κατασκευάσει ένα νέο kernel image αλλά και θα το εγκαταστήσει στον κατάλογο */minix*. Το πρόγραμμα */boot*, που αναλαμβάνει το bootstrapping του MINIX, κοιτάζει σε αυτόν τον κατάλογο και φορτώνει στη μνήμη το νεότερο image που περιέχει. Είναι αρκετά χρήσιμο σε περίπτωση που το νέο image που κατασκευάσατε δημιουργεί σοβαρά προβλήματα (π.χ. το σύστημα δεν μπορεί καν να αρχικοποιηθεί ή προκαλεί kernel panic) μπορείτε να επιλέξετε κάποιο παλιότερο kernel image με τον εξής τρόπο. Κατά την εκκίνηση του συστήματος αντί να πιέσετε το πλήκτρο "=" πιέστε "Esc" και στη συνέχεια εισάγετε στο prompt την εντολή `image=<image_path>`, όπου `<image_path>` είναι το πλήρες path ενός λειτουργικού image (π.χ. */minix/2.0.0*). Στη συνέχεια απλά εκτελέστε την εντολή `boot`.

Την επόμενη φορά που θα προσπαθήσετε να κατασκευάσετε ένα image, εκτελέστε απλά την εντολή `make hdboot`, η οποία θα κάνει recompile μόνο τα αρχεία που έχουν υποστεί κάποια αλλαγή.

Library Compilation:

Πέρα από τον νέο τροποποιημένο πυρήνα που μόλις εγκαταστήσατε, θα πρέπει να κατασκευάσετε και μία νέα βιβλιοθήκη συστήματος την οποία θα χρησιμοποιήσει ο χρήστης για να καλέσει το νέο system call. Πριν πραγματοποιήσετε τα επόμενα βήματα κρατήστε ένα αντίγραφο της αρχικής βιβλιοθήκη συστήματος που βρίσκετε στο αρχείο */usr/src/lib/libc.a*. Η διαδικασία κατασκευής της νέας libc είναι η ακόλουθη:

Βήμα 1ο: Αφού θέσετε ως τρέχοντα (εκτελώντας `cd`) τον κατάλογο */usr/src/lib*. Εκτελέστε τις εντολές:

```
make
make install
```

Η πρώτη εντολή κατασκευάζει τη νέα βιβλιοθήκη συστήματος και την αποθηκεύει στο αρχείο */usr/src/lib/libc.a*. Η δεύτερη εντολή την αντιγράφει στον κατάλογο */usr/lib/i386* (ή στον αντίστοιχο κατάλογο που ορίζει η τρέχουσα αρχιτεκτονική του συστήματος).

Παράρτημα Β

Τα περιεχόμενα του αρχείου <sys/acl..h> είναι τα εξής:

```
#ifndef _ACL_H
#define _ACL_H

#include <sys/types.h>

typedef struct acl {
    long        acl_flags;     /* acl flags */
    uid_t       acl_id;        /* user id */
    mode_t      acl_perm;      /* permission bits */
} acl_t;

#define NACLS   (BLOCK_SIZE/sizeof(acl_t))  /* max # of ACL entries / file */

/* acl option definitions */
#define ACL_SET         1       /* set ACL */
#define ACL_GET         2       /* get ACL */
#define ACL_CLR         3       /* remove ACL block */
#define ACL_CNT         4       /* # ACL entries */

/* acl type definitions */
#define ACL_GID         01      /* acl_id contains a group id*/
#define ACL_UID         02      /* acl_id contains a user id*/

/* acl permission bits */
#define ACL_RBIT        00004   /* read permission */
#define ACL_WBIT        00002   /* write permission */
#define ACL_XBIT        00001   /* execute permission */

#ifndef _ANSI_H
#include <ansi.h>
#endif

_PROTOTYPE( int acl, (char *name, int cmd, int nacls, acl_t *aclbufp) );

#endif /* _ACL_H */
```

Βιβλιογραφία

[1] *Modern Operating Systems*, Andrew S. Tanenbaum, 2nd ed., Prentice Hall 2001

[2] *Operating Systems, Design and Implementation,* Andrew S. Tanenbaum and Albert S. Woodhull, 2nd ed., Prentice Hall 1996