Πρότυπη Εργαστηριακή Άσκηση

Στα πλαίσια της παρούσας προπαρασκευαστικής άσκησης καλείστε να υλοποιήσετε μία κλήση συστήματος (*system call*) η οποία θα δέχεται ως είσοδο το path ενός αρχείου και θα επιστρέφει τα process IDs (*pids*) των διεργασιών που έχουν ανοικτό αυτό το αρχείο[1]. Η χρήση της κλήσης αυτής θα πρέπει να είναι δυνατή μόνο για τον χρήστη *root*. Στο MINIX οι system calls υλοποιούνται ως μηνύματα που αποστέλλονται από την διεργασία επιπέδου χρήστη (4ο επίπεδο) σε μία από τις διεργασίες επιπέδου "Server Processes" (3ο επίπεδο). Η κλήση που σας ζητείται, θα υλοποιηθεί ως αποστολή μηνύματος προς την διεργασία του File System Server (*FS*).

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 blocking πέρασμα μηνυμάτων.
- 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.
- Κατανόηση μερικών από τις βασικότερες δομές δεδομένων που χρησιμοποιούνται από τον FS Server και γενικότερα το λειτουργικό σύστημα.

Στη συνέχεια παρατίθεται η δήλωση και η ακριβής περιγραφή της λειτουργικότητας της κλήσης που θα υλοποιήσετε.

Δήλωση

```
int getprocs(const char *_fpath, const int _fpsize, pid_t *_procs,
             const int _nprocs);
```

`_fpath`: η διεργασία επιστρέφει τα pids των διεργασιών που έχουν ανοικτό το αρχείο στο οποίο δείχνει το path `_fpath`.

`_fpsize`: το μήκος του `_fpath` σε χαρακτήρες συμπεριλαμβανομένου του χαρακτήρα '\0' (null).

`_procs`: δείκτης σε μνήμη, που έχει δεσμευτεί προηγούμενα από τον χρήστη και στην οποία θα αποθηκεύσει ο FS Server τα pids των διεργασιών που βρήκε ότι έχουν ανοίξει το αρχείο που δείχνει το `_fpath`.

`_nprocs`: ακέραιος, που δηλώνει το μέγιστο πλήθος pids που μπορεί να επιστρέψει η system call. Το μέγεθος της δεσμευμένης μνήμης που δείχνει ο `_procs` πρέπει να είναι αρκετή για να αποθηκευτούν `_nprocs` σε πλήθος pids.

Επιστρεφόμενες τιμές

Η κλήση αν είναι επιτυχής επιστρέφει το πλήθος των διεργασιών που εντόπισε (≥ 0 και $\leq$ `_ncprocs`) και αποθηκεύει τα process IDs τους στη μνήμη που δείχνει ο δείκτης `_procs`. Σε περίπτωση λάθους επιστέφεται –1 και η μεταβλητή errno τίθεται σε μία από τις ακόλουθες τιμές:

`EPERM`: Η τιμή αυτή επιστρέφεται εάν η συνάρτηση κληθεί από κάποιον χρήστη πέραν του υπερχρήστη *root*.

`ENOENT`: Η τιμή αυτή επιστρέφεται εάν δεν υπάρχει το αρχείο με path `_fpath`.

`ENAMETOOLONG`: Η τιμή αυτή επιστρέφεται εάν το μήκος του `_fpath` (`_fpsize`) ξεπερνά την σταθερά `PATH_MAX`[2] (συμπεριλαμβανομένου του χαρακτήρα *null*).

---

[1] Αναφέροντας ότι μία UNIX διεργασία έχει ανοικτό ένα αρχείο εννοούμε ότι η διεργασία έχει ανοικτό κάποιο *file descriptor* που δείχνει στο αρχείο αυτό.

[2] Η `PATH_MAX` ορίζεται στο αρχείο */usr/include/limits.h*.

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 έκδοσης" του system call
iii. επέκταση του (ii) έτσι ώστε να εκτυπώνονται οι ενεργές διεργασίες του συστήματος
iv. επέκταση του (iii) έτσι ώστε να εκτυπώνονται και τα ανοικτά αρχεία κάθε διεργασίας
v. επέκταση του (iv) έτσι ώστε τα αποτελέσματα να επιστρέφονται στις θέσεις μνήμης που έδωσε ο χρήστης ως όρισμα

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 `getprocs()`. Συγκεκριμένα αναζητείστε την υλοποίηση συναρτήσεων όπως:

i. `open()` που λαμβάνει ως όρισμα το path προς ένα αρχείο
ii. `read()` που επιστρέφει το αποτέλεσμα της σε μία *user-provided* θέση μνήμης
iii. `mount()` που επιτρέπει μόνο στον *root* να προσαρτήσει ένα file system

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. *file.h*: περιέχει την δομή όπου αποθηκεύεται ένας *file descriptor*. Βρείτε το μέλος της δομής που χρησιμοποιείται για να προσπελαστεί η πληροφορία για ένα αρχείο (μέγεθος, όνομα, κτλ.)
iii. *inode.h*: περιέχει τη δομή όπου αποθηκεύονται τα εν χρήση *i-nodes* (i-node table)

4. Η γενική (*generic*) 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.

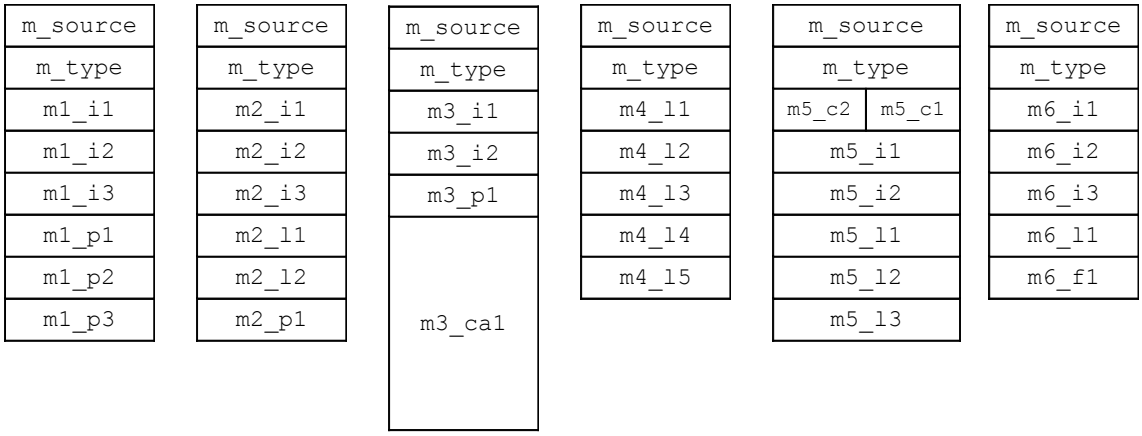
5. Προσέξτε ότι στον σχολιασμό του κώδικα της δομής `fproc` (*fproc*.h) τονίζεται το γεγονός ότι στη δομή δεν περιλαμβάνεται πληροφορία για το ποια από τα στοιχεία του πίνακα διεργασιών που διατηρεί ο FS Server αντιστοιχούν σε υπάρχουσες διεργασίες. Προφανώς θα πρέπει εσείς να προσθέσετε κάποιο πεδίο στη βασική αυτή δομή που να παρέχει αυτή την πληροφορία και στη συνέχεια να εντοπίσετε τα σημεία του κώδικα του FS Server όπου αυτή θα πρέπει να αρχικοποιείται και να αλλάζει τιμές. Σαν υπόδειξη θυμηθείτε τον κύκλο ζωής μίας διεργασίας σε ένα Unix λειτουργικό σύστημα, όπου όλες οι διεργασίες έχουν μακρινό "πρόγονο" τη διεργασία *init*, η οποία έχει `pid=1`, και οι διεργασίες γεννώνται μέσω των κλήσεων `fork()` και `exec()` ενώ τερματίζουν όταν κληθεί η `exit()`. Τις κλήσεις αυτές στο MINIX εξυπηρετεί αρχικά ο Memory Management Server (*MM*), ο οποίος στη συνέχεια αναλαμβάνει να ενημερώσει σχετικά με τα γενόμενα τους ενδιαφερόμενους servers (FS, etc) και τον kernel (layers 1 και 2 της αρχιτεκτονικής του MINIX). Εντοπίστε στον πηγαίο κώδικα του MM ποια μηνύματα αποστέλλει στον FS Server κατά την εξυπηρέτηση των `fork`,`exec` και `exit` αλλά και που στον κώδικα του FS διαχειρίζεται η λήψη αυτών των μηνυμάτων. Στο σημείο αυτό το χρησιμότερο εργαλείο είναι το πρόγραμμα `grep` έτσι ώστε εύκολα να βρείτε τα αρχεία που περιέχεται αυτό που αναζητάτε. Για αρχή αναζητείστε την λέξη `fork`.

6. Για την υλοποίηση της συνάρτησης επιπέδου χρήστη που θα στέλνει το μήνυμα για εξυπηρέτηση στον FS Server θα πρέπει να επιλέξετε κάποιο κατάλληλο είδος μηνύματος. 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 1ο είδος μηνύματος και θέλετε να αναφερθείτε στο στοιχείο `m1i1` του union `mu` θα μπορούσατε αντί του δύσχρηστου και δυσμνημόνευτου `m.m_u.m_m1.m1p1` 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:



Στα πλαίσια της άσκησης θα πρέπει να αποφασίσετε ποιο είδος θα χρησιμοποιήσετε για την υλοποίηση της `getrprocs()` κάτι το οποίο είναι αρκετά απλό χρησιμοποιώντας το παραπάνω σχήμα.

7. Στον κατάλογο */usr/src/lib/posix* μπορείτε να βρείτε πλήθος παραδειγμάτων συναρτήσεων βιβλιοθήκης που ορίζονται από το πρότυπο POSIX και υλοποιούνται από το MINIX ως κλήσεις προς system calls. Κάτι ανάλογο καλείστε να υλοποιήσετε στο 2ο βήμα της διαδικασίας υλοποίησης ενός system call που περιγράφεται στο "Παράρτημα Α".

8. 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`.

9. Μην σβήσετε τον αρχικό kernel */minix/2.0.0* για να μπορέσετε στη περίπτωση που κάτι απρόβλεπτο συμβεί με το καινούργιο image, να ξεκινήσετε το σύστημα και να διορθώσετε το πρόβλημα που προέκυψε.

Παράρτημα

Προσθήκη κλήσης συστήματος στο λειτουργικό σύστημα MINIX

Η διαδικασία υλοποίησης μία νέας κλήσης συστήματος στο MINIX χωρίζεται σε δύο στάδια. Στο 1ο στάδιο υλοποιείται στον FS 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 στο FS Server. 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_getprocs`).

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 ενσωματωθεί στη βασική βιβλιοθήκη κάθε Unix συστήματος, τη libc. Για λόγους απλότητας και ταχύτερης ανάπτυξης όμως στη συνέχεια παρατίθεται ένας εναλλακτικός τρόπος ο οποίος ουσιαστικά ενσωματώνει τη συνάρτηση βιβλιοθήκης στατικά στο πρόγραμμα του χρήστη.

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 (βλ. "Δήλωση").

Βήμα 2ο: Στο directory που θα χρησιμοποιείστε για να αποθηκεύσετε τα user-level test προγράμματα (π.χ. */root*) ανοίξτε ένα καινούργιο header αρχείο (π.χ *getprocs.h*) και συμπεριλάβετε κάτι ανάλογο με

[3] Διαφορετικά θα πρέπει να αλλάξετε κατάλληλα και το αρχείο Makefile που βρίσκεται στον κατάλογο */usr/src/fs*

το ακόλουθο (βλ. υπόδειξη 7):

```
#include <lib.h>
#define getprocs _getprocs
#include <unistd.h>

int getprocs(const char *_fpath, const int _fpsize, pid_t *_procs,
             const int _nprocs) {

message m;

/* αντιγραφή των παραμέτρων της συνάρτησης στο μήνυμα m */
...
...
...

return (_syscall(FS, GETPROCS, &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*.

Σε κάθε αρχείο όπου θέλετε να καλέσετε τη system call `getprocs()` απλά κάντε include το *getprocs.h*.

Kernel Compilation:

Για να χρησιμοποιήσετε την συνάρτηση `getprocs()` 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 σηκωθεί 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.

Αναφορές

[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