Εργαστήριο Λειτουργικών Συστημάτων (330Ε)

Εργαστηριακή Άσκηση 2005-2006

Θέμα

Υλοποίηση ενός υποτυπώδους /proc Filesystem για το MINIX 2.0.0

Θεωρία

Η έννοια του FileSystem είναι κατά κανόνα συνδεδεμένη με την έννοια μιας υπηρεσίας επιπέδου πυρήνα η οποία αναλαμβάνει τη διαχείριση αρχείων και καταλόγων στο σύστημα μας. Η λειτουργικότητας μιας οργάνωσης όπως αυτή που συνεπάγεται ένα κοινό FileSystem, ειδικά στα *nix συστήματα, με τον κύριο κατάλογο (root: /) και τους ένθετους σε αυτόν υποκαταλόγους (/etc, /bin, /usr κτλ), είναι ιδιαίτερα σαφής και οικεία σε όλους τους χρήστες υπολογιστών. Η λειτουργικότητα αυτή, που επιτρέπει στον χρήστη να οργανώνει τα αρχεία του σε καταλόγους και υποκαταλόγους, να τα ανοίγει, να τα διαβάζει και να τα αλλάζει κατά βούληση, είναι αυτή την οποία θέλησαν να εκμεταλλευθούν οι σχεδιαστές του /proc FileSystem. Πέρα όμως από αυτή την παράμετρο, το /proc ουδεμία σχέση έχει με FileSystems διαχείρισης αρχείων: δεν αποθηκεύει αρχεία με τη μορφή binary ή text που γνωρίζουμε. Θα χρησιμοποιήσουμε για την (αφαιρετική) περιγραφή του σαν παράδειγμα το λειτουργικό σύστημα GNU/Linux, μιας και μπορεί ο καθένας να έχει πρόσβαση εύκολα σε ένα τέτοιο (ο γνωστός server του τμήματος diogenis τρέχει κάποια έκδοση του).

Ας υποθέσουμε ότι έχουμε σε ένα GNU/Linux σύστημα δύο σκληρούς δίσκους: έναν formatted με FAT32 FileSystem (κοινό στα συστήματα Windows), και έναν με EXT3 (κοινό στα GNU/Linux συστήματα). Τότε θα πρέπει να «προσδέσουμε» (mount) τις αντίστοιχες συσκευές κάτω από τον κοινό root / κατάλογο για να μπορούμε να τις δούμε. Το ίδιο γίνεται για το /proc FileSystem: μια εικονική συσκευή, η proc device, προσδένεται στον κατάλογο /proc. Τυπικά, η εικονική αυτή συσκευή είναι “formatted” σαν /proc FileSystem.

Κάτω από τον κατάλογο /proc, αμέσως μόλις γίνει mount, υπάρχουν μια σειρά από αρχεία και κατάλογοι. Τα περισσότερα από τα αρχεία όμως έχουν μηδενικό μέγεθος και χαρακτηριστικά ονόματα. Για παράδειγμα υπάρχει το αρχείο cpuinfo και το iomem, ο κατάλογος ide και μια σειρά από καταλόγους με ονόματα αριθμούς. Μάλιστα αυτά τα αρχεία ποτέ δεν τα δημιουργήσατε εσείς ή κάποιος άλλος: δε σχετίζονται και δεν υπάρχουν κάπου στον υπολογιστή σας, εκτός ίσως από τη μνήμη (RAM) του. Είναι απλά μια μέθοδος να δει κανείς την εικόνα του πυρήνα για το σύστημα, να του ζητήσει ή και να του δώσει πληροφορίες.

Τι συμβαίνει ακριβώς: Το /proc FS είναι μια παράμετρος που ενεργοποιήθηκε κατά το compilation του πυρήνα του GNU/Linux. Η παράμετρος αυτή υπαγόρευσε στον πυρήνα να κατασκευάσει εσωτερικές δομές (structs) στις οποίες, στη μνήμη, θα κρατά πληροφορίες για την τρέχουσα κατάσταση του συστήματος. Η ανάγνωση κάθε αρχείου στον κατάλογο /proc αντιστοιχεί στην ανάγνωση τέτοιων δομών απευθείας από τον πυρήνα.

Οι πληροφορίες που περιέχουν τα εν λόγω αρχεία αφορούν πρώτα πρώτα το υλικό, όπως από ποιόν κατασκευαστή και ποια έκδοση είναι η CPU, τι εντολές υποστηρίζει, ποια είναι η ανάθεση καταχωρητών μνήμης στο σύστημα μας, και ανάλογες πληροφορίες. Μάλιστα, όταν υπάρχουν πολλές ομοειδείς πληροφορίες, τοποθετούνται κάτω από έναν κοινό κατάλογο, πχ στον ide τοποθετούνται αρχεία με πληροφορίες που αφορούν τον ide controller. Οι πληροφορίες αυτές μπορεί να είναι αναγνώσιμες και να έχουν νόημα για τον άνθρωπο που τις διαβάζει, μπορεί όμως και όχι.

Πλέον αυτών, το /proc FS διατηρεί πληροφορίες για την τρέχουσα κατάσταση του συστήματος σε ό,τι αφορά το λογισμικό, και φυσικά πιο συγκεκριμένα τις τρέχουσες διεργασίες. Κάτω από τον /proc κατάλογο λοιπόν, δημιουργούνται φάκελοι με ονόματα τα PIDs των τρεχουσών διεργασιών του συστήματος. Όλα αυτά γίνονται και παραμένουν τελείως δυναμικά: μια διεργασία που δε θα υπάρχει πλέον στο σύστημα, δε θα έχει και κατάλογο στο /proc FS. Ο κατάλογος αυτός περιέχει για κάθε διεργασία χρήσιμες πληροφορίες, όπως την κατάσταση της, τα δικαιώματα της, τον κατάλογο στον οποία εργάζεται (working directory), πληροφορίες για τη μνήμη που έχει δεσμεύσει η διεργασία και το πως τη διαχειρίζεται κλπ. Όλες αυτές οι πληροφορίες είναι οργανωμένες και πάλι σε μια σειρά από χαρακτηριστικά αρχεία: cmdline, που αναγράφει την εντολή που αντιστοιχεί στη διεργασία, environ, που αφορά τις τρέχουσες μεταβλητές περιβάλλοντος, status, που περιέχει pending signals κ.α. Επιπλέον, για να μπορεί ο κάθε χρήστης να έχει πρόσβαση σε πληροφορίες που αφορούν διεργασίες δικές του ή κοινού ενδιαφέροντος, κάθε κατάλογος και αρχείο που δημιουργείται, έχει τα δικαιώματα που έχει και η αντίστοιχη διεργασία.

Όπως έχουμε πει, η διεπαφή του λειτουργικού συστήματος με τον χρήστη – προγραμματιστή είναι κυρίως τα system calls του. Όπως μπορεί κανείς να αντιληφθεί, η ανάγνωση ενός αρχείου στο /proc FS δε μπορεί να είναι ίδια με αυτή με ένα πραγματικό αρχείο σε FAT32 FS: στη δεύτερη περίπτωση ψάχνουμε inodes, ανοίγουμε filedescriptors, κινούμαστε μέσα στο αρχείο, ενώ στη δεύτερη παρακάμπτουμε τη συνήθη διαδικασία και διαβάζουμε κατευθείαν τις πληροφορίες που μας δίνει ο πυρήνας, φροντίζοντας να τις επιστρέψουμε σε αναγνώσιμη μορφή. Αυτό σημαίνει μια διαφοροποίηση σε system calls διαχείρισης αρχείων: αλλιώς θα χειρίζονται τα πραγματικά αρχεία και αλλιώς τα εικονικά στο /proc FS. Επιπλέον, το γεγονός ότι το σύστημα είναι δυναμικό, σημαίνει πως οι ανάλογες δομές θα πρέπει να ενημερώνονται με κάθε αλλαγή στην κατάσταση του συστήματος. Έτσι ένα fork θα πρέπει να σημάνει και τη δημιουργία ενός καταλόγου στο /proc FS, με τα δικαιώματα και το PID της διεργασίας που το εκτέλεσε. Όλα αυτά σημαίνουν τροποποίηση βασικών system calls. Τέλος, θα πρέπει να σημειωθεί πως ειδικά στο GNU/Linux υπάρχουν system calls που επιτρέπουν στο χρήστη να ανοίξει ένα αρχείο, να αλλάξει πληροφορίες εντός του και αυτές να περάσουν στον πυρήνα: πχ θα μπορούσε να αλλάξει το uid του χρήστη που εκτελεί μια διεργασία δυναμικά.

Στόχοι

Στα πλαίσια της παρούσας άσκησης, θα προσπαθήσουμε να έχουμε μια πρώτη επαφή με την έννοια ενός /proc FS. Η υλοποίηση που θα επιχειρήσουμε θα είναι εντελώς υποτυπώδης και θα στερείτε των βασικών χαρακτηριστικών μιας πλήρους υλοποίησης, όπως η περιγραφείσα. Αυτό οφείλεται τόσο στους περιορισμούς που εισάγει η ίδια η υποδομή του MINIX, όσο και σε περιορισμούς που

αφορούν την πολυπλοκότητα της υλοποίησης. Ωστόσο θα μας φέρει σε επαφή με τη διαδικασία τροποποίησης και προσθήκης system calls στο λειτουργικό σύστημα MINIX και το μηχανισμό περάσματος μηνυμάτων ανάμεσα στο χρήστη και στα διάφορα επίπεδα του πυρήνα.

Ζητούμενα

Η υλοποίηση του /proc FS για το MINIX 2.0.0 που χρησιμοποιούμε στο εργαστήριο θα είναι υποτυπώδης. Το /proc FS που σας ζητείται δε θα είναι δυναμικό. Αυτό σημαίνει πως θα «χτίζεται» και θα περιέχει πληροφορίες που θα ανανεώνονται αποκλειστικά κατόπιν αίτησης του χρήστη. Επιπλέον τα αρχεία που το αφορούν θα είναι **πραγματικά**: δε θα έχουν δηλαδή μηδενικό μέγεθος, αλλά θα περιέχουν τα δεδομένα τους όπως τα συνήθη αρχεία κειμένου. Το ίδιο θα ισχύει και για τους καταλόγους: δε θα έχουν τίποτα το ιδιαίτερο, θα είναι συνήθεις κατάλογοι του Minix FS. Περιττό να αναφερθεί πως το proc FS δε θα αντιστοιχεί σε καμία εικονική συσκευή. Εν ολίγοις θα πρέπει να κατασκευάσετε και να προσθέσετε στον FS δυο system calls: το ένα εξ αυτών θα χτίζει τη δομή καταλόγου ενός /proc FS κάτω από το ζητούμενο directory και το δεύτερο θα το ανανεώνει (δηλαδή τα αρχεία που αυτό περιέχει σε κάθε υποκατάλογο) με την τρέχουσα κατάσταση του συστήματος, σε ό, τι αφορά μόνο τις τρέχουσες διεργασίες.

Το interface που θα χρησιμοποιείτε[1] για την κλήση των δυο αυτών system calls θα έχει ως εξής:

int procfs_create(const char * path_name)

και θα καλείται από τον root για να κατασκευαστεί το proc FS και όπου:

- το string path_name θα είναι ο υπαρκτός κατάλογος κάτω από τον οποίο θα δημιουργηθεί το proc FS (για παράδειγμα αν δώσετε σαν όρισμα τον κατάλογο /usr , το proc FS θα δημιουργηθεί κάτω από τον κατάλογο /usr/proc)
- η επιστρεφόμενη τιμή int θα είναι:
  - EINVAL αν το path_name δεν είναι έγκυρο
  - EPERM αν ο χρήστης δεν είναι ο root
  - EGENERIC για οποιοδήποτε άλλο μη προσδοκώμενο λάθος

int procfs_update(void)

και θα καλείται από οποιονδήποτε χρήστη για να ανανεωθεί το proc FS με τις τρέχουσες πληροφορίες του συστήματος και όπου

- η επιστρεφόμενη τιμή int θα είναι:
  - EGENERIC για οποιοδήποτε μη προσδοκώμενο λάθος

Σε ότι αφορά τις πληροφορίες κάθε διεργασίας, αυτές θα περιέχονται σε 5 αρχεία, με τα εξής ονόματα, και με μορφοποίηση της επιλογής σας:

- status : για την κατάσταση μιας διεργασίας (πχ pending signals)
- psinfo : με πληροφορίες για τη διεργασία (πχ runtime)
- credentials : για τα δικαιώματα της διεργασίας, πχ uid, gid, egid κτλ
- memap : για το address map της διεργασίας
- cwd : για το working directory της διεργασίας

Τα αρχεία αυτά μπορείτε να τα εμπλουτίσετε όσο εσείς θέλετε με πληροφορίες που θα πάρετε από

[1] Περιγράφεται το interface της βιβλιοθήκης: οδηγίες για το πώς θα το αξιοποιήσετε περιέχονται στο παράρτημα.

τον πυρήνα και τα services του λειτουργικού (kernel, mm, fs).
Επιπλέον θα πρέπει να κατασκευάσετε ένα απλό και τυπικό user level πρόγραμμα που θα αξιοποιεί το υποτυπώδες proc filesystem για να δίνει πληροφορίες στον χρήστη – ένα πρόγραμμα επίδειξης του proc FS σας. Μια καλή ιδέα για την εμφάνιση ενός τέτοιου προγράμματος είναι η εντολή ps.

Συμβουλές

- 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 ζητούμενων απευθείας. Κατασκευάστε ενδιάμεσες λειτουργικές εκδόσεις που υλοποιούν ένα υποσύνολο των ζητούμενων και των οποίων την ορθότητα μπορείτε εύκολα να επαληθεύσετε τυπώνοντας μηνύματα ελέγχου σε κάθε φάση. Για παράδειγμα η συγκεκριμένη άσκηση μπορεί να χωριστεί στα εξής τμήματα:

  procfs_create: εξυπηρέτηση από τον fs
  1. υλοποίηση ενός "dummy" system call που εκτυπώνει απλά ένα μήνυμα
  2. επέκταση του (i) έτσι ώστε να εκτυπώνονται τα ορίσματα που περιέχονται στο μήνυμα του system call
  3. προσθήκη κατάλληλων δομών που θα συγκρατούν τη θέση του τρέχοντος proc fs (όρισμα library call)
  4. προσθήκης μιας εσωτερικής συνάρτησης που θα επιτυγχάνει την αλλαγή (αν είναι δυνατή), στον κατάλογο – όρισμα
  5. επέκταση της παραπάνω προσθήκης ώστε στη γενική της μορφή να λειτουργεί σαν change_and_make_directory(which_directory): εφαρμογή ώστε να δημιουργείται το directory proc, αν δεν υπάρχει ήδη
  6. προσθήκη κατάλληλων δομών που θα συγκρατούν τη θέση του τρέχοντος proc fs

procfs_update: εξυπηρέτηση από τον fs
  1. υλοποίηση ενός "dummy" system call που εκτυπώνει απλά ένα μήνυμα
  2. επέκταση του ώστε να επιβεβαιώνει την ύπαρξη procfs και να εκτυπώνει τη θέση του
  3. προσθήκη κατάλληλων δομών που θα συγκρατούν τα ζητούμενα δεδομένα από όλα τα process tables: kernel, fs, mm
  4. προσθήκη συναρτήσεων που να ζητούν απαραίτητες πληροφορίες από τα έτερα services: πχ αν το system call υλοποιείται στο fs, να στέλνεται ένα μήνυμα στον mm και η απάντηση να περιέχει τις ζητούμενες πληροφορίες από το process table
  5. προσθήκη συνάρτησης που να δημιουργεί το αρχείο με το κατάλληλο όνομα
  6. προσθήκη συνάρτησης που να προσθέτει τις πληροφορίες που δέχεται (πχ σε buffer ή κατευθείαν από την κατάλληλη δομή) στο ανάλογο αρχείο
  7. Κάποια από τα παραπάνω βήματα μπορούν να ενοποιηθούν. Αυτό καθορίζεται ελεύθερα και δυναμικά από εσάς βάσει της εμπειρίας σας.

  Η επιλογή του κατάλληλου τύπου μηνύματος (1-6) για κάθε system call είναι απλή υπόθεση αν λάβετε υπ' όψιν σας τα ζητούμενα.

- Προσπαθήστε να εντοπίσετε σημεία στον υπάρχον κώδικα του MINIX που είναι κοντά στα ζητούμενα της άσκησης ή σε αυτά των επιμέρους βημάτων τους, όπως τα περιγράψαμε παραπάνω ή όπως τα φαντάζεστε εσείς. Χαρακτηριστικά μπορεί να σας βοηθήσει ιδιαίτερα η ανάγνωση / εντοπισμός των εξής:

  - kernel/proc.h, mm/mproc.h, fs/fproc.h: περιλαμβάνουν τα process tables και τις

σταθερές που περιγράφουν την κατάσταση μίας διεργασίας
  - tools/ps.c: η γνωστή "ps". Μην παρασυρθείτε από τη λύση που περιγράφει, να θυμάστε πως αυτή λειτουργεί σε user level επίπεδο.
  - chdir, mkdir, open, write, close : βρείτε τα system calls που χειρίζονται τις εν λόγω εντολές - μπορεί να σας βοηθήσουν στη δημιουργία καταλόγων/αρχείων
  - σημείων στον κώδικα όπου ένα service, πχ ο mm επικοινωνεί με ένα άλλο (fs) ή τον kernel. Πιθανότατα θα χρησιμοποιήσετε τον ίδιο ή παρεμφερή τρόπο για τις ανάγκες επικοινωνίας της δικιάς σας κλήσης.
  - main.c : είτε τον fs είτε στον mm είτε στον kernel, είναι το αρχείο που περιέχει τη main και εξηγεί πως δουλεύει κάθε service: πως περιμένει μηνύματα, πως τα χειρίζεται κοκ.

- Δώστε ιδιαίτερη προσοχή στην προσπάθεια κατανόησης της επικοινωνίας μεταξύ διεργασιών στο MINIX. Θα πρέπει οπωσδήποτε να θυμάστε το εξής: αφού κάθε διεργασία έχει το δικό της user space, το να περάσει κανείς ένα δείκτη σε μια θέση μνήμης στο user space της διεργασίας-αποστολέα, σε μια άλλη διεργασία-παραλήπτη δίχως καμία τροποποίηση, θα δώσει στη διεργασία παραλήπτη έναν δείκτη σε μια ακαθόριστη θέση μνήμης, στο user space της διεργασίας-παραλήπτη και όχι του αποστολέα. Προσέξτε λοιπόν το μηχανισμό περάσματος μηνυμάτων και πως θα λύσετε τέτοια ζητήματα όπου τα χρειαστείτε.

- Το γεγονός ότι μεγάλο μέρος της ζητούμενης λειτουργικότητας υπάρχει ήδη, πχ η κατασκευή καταλόγων, σημαίνει πως θα ήταν έξυπνο αντί να αντιγράφετε τον κώδικα και να τον προσαρμόζετε στη δικιά σας περίπτωση, να προσπαθήσετε να στείλετε κατάλληλα μηνύματα ώστε εμμέσως να κληθούν οι αντίστοιχες system calls και να κάνουν τη δουλειά για σας. Μελετείστε τη διαδικασία αυτή λήψης και εξυπηρέτησης ενός μηνύματος. Προσέξτε πως ίσως σε κάποια σημεία θα πρέπει να βρείτε έναν έξυπνο τρόπο ώστε μια διεργασία να στέλνει μηνύματα στον εαυτό της.

- Μεγάλο μέρος του user level programming interface δεν είναι διαθέσιμο στο επίπεδο του πυρήνα (συμπεριλαμβανομένων των services). Η δυναμική δέσμευση μνήμης είναι χαρακτηριστική τέτοια έλλειψη. Φροντίστε να δεσμεύστε μνήμη εξ' αρχής και γενικά, όπου μπορείτε να κάνετε τη δουλειά σας με το υπάρχον interface (συναρτήσεις πυρήνα / services) να μην αναβαίνετε σε επίπεδο library calls. Η τελευταία συμβουλή είναι κανόνας μόνον όταν δημιουργούνται προβλήματα που μπορούν να αποφευχθούν, πχ κατά τη δυναμική δέσμευση μνήμης.

- Αν χρειαστεί να προσθέσετε δομές / μεταβλητές στα επίπεδα πυρήνα/services. μην ξεχνάτε να τις αρχικοποιείτε την κατάλληλη στιγμή.

- 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 hda.img στον οποίο εργάζεστε. Η zipped έκδοση του είναι μικρή σε μέγεθος: θα ήταν ίσως καλό να παίρνετε τακτικά compressed backups που ξέρετε πως δουλεύουν, σαν restore points.

- Χρησιμοποιείστε το forum του μαθήματος για απορίες και επικοινωνία, μεταξύ σας κατά κύριο λόγο αλλά και με τους υπεύθυνους. Εναλλακτικά, για απορίες που δεν έχουν κοινό ενδιαφέρον, στείλτε e-mail στο *menihtas@ceid.upatras.gr* . Έχετε το νου σας καθώς ανάλογα με το feedback θα αποσαφηνίζονται κοινές απορίες και στο δίκτυο, όπως και στο μάθημα. Προσπαθήστε να είστε σαφείς και χρήσιμοι στις ερωταπαντήσεις σας. Χρησιμοποιείστε το

δίκτυο και τα βιβλία ([1], [2]), και ιδιαίτερα το index του [2]

- Ανεκτίμητες εντολές είναι οι “grep -r” και “vi”. Όσοι έχετε συνηθίσει τον pico σαν editor ίσως προτιμήσετε το “mined” αντί του vi, ή ακόμα τον “elle”, που μοιάζει πιο πολύ στον emacs. Πάντως cheatsheets, reference cards και tutorials για τον vi μπορεί να φανούν χρήσιμα, μιας και είναι πιο δυνατός editor στο MINIX.

Παραδοτέα

Η εργασία αυτή θα υλοποιηθεί από **ομάδες τριών ατόμων**, αυστηρά. Στα πλαίσια της θα πρέπει να παραδώσετε τα εξής:

- Ένα αρχείο text, με τα ονόματα και τη θέση στο filesystem των αρχείων που τροποποιήσατε/προσθέσατε. Για παράδειγμα
  /usr/src/fs/procfs.c: lines 1-100 (new file)
  /usr/src/fs/table.c: lines 1-10 (addition)
  /usr/src/fs/param.h: lines 15-16 (modification)

- Ένα zip-ped αρχείο που να περιέχει το image του σκληρού δίσκου (hda.img) στο οποίο δουλεύατε και στο οποίο περιέχεται ο κώδικας σας. Φροντίστε να επιλέξετε τη μεγαλύτερη δυνατή συμπίεση.

  Ο κώδικας σας θα πρέπει να είναι επαρκώς σχολιασμένος. Κάθε προσθήκη κώδικα θα πρέπει να εσωκλείεται 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 ***OSLAB CROP START*** και ***OSLAB CROP END*** στην αρχή και στο τέλος τους αντίστοιχα και να συνεχίσουν να υπάρχουν στην ίδια θέση με τη μορφή σχολίων.

- Τεχνική αναφορά (σε μορφή .pdf κατά προτίμηση) όπου θα περιέχονται τα εξής
  - Πλήρης περιγραφή της λύσης σας και των βημάτων που ακολουθήσατε για να φτάσετε σε αυτή.
  - Πλήρης περιγραφή του κώδικα που προσθέσατε / τροποποιήσατε και σχολιασμός της λειτουργίας που εκτελεί. Κατά προτίμηση ο κώδικας θα πρέπει να εμφανίζεται με τη μορφή που δίνει τις διαφορές μεταξύ δυο αρχείων (παλιά – νέα έκδοση) η εντολή diff.

Τα παραπάνω θα πρέπει να σταλούν με e-mail στο *oslab@ceid.upatras.gr* πριν την ημερομηνία παράδοσης που θα ανακοινωθεί αμέσως μετά την ανακοίνωση του προγράμματος εξεταστικής του Ιουνίου, στο forum και στην ιστοσελίδα του μαθήματος. Το e-mail θα πρέπει να αποσταλεί με Subject της μορφής “OSLAB <α/α ομάδας>” (π.χ. για την ομάδα με α/α 17, Subject: OSLAB 17). Η δήλωση των ομάδων θα γίνει συμπληρώνοντας κατάλληλη φόρμα στην ιστοσελίδα του μαθήματος, κατόπιν σχετικής ανακοίνωσης.

Τέλος να σημειωθεί πως αν και η εκφώνηση είναι αρκετά σαφής ως προς τα ελάχιστα ζητούμενα, προφανώς και δεν υπάρχει πάνω όριο: οποιοσδήποτε θέλει να ασχοληθεί περισσότερο μπορεί να δοκιμάσει ακόμα και μια δυναμική λύση. Σε αυτή την περίπτωση όμως δε θα έχει υποστήριξη.

Η βαθμολόγηση θα είναι ανάλογη της ανταπόκρισης στα ζητούμενα της άσκησης και το κάτω όριο (προβιβάσιμος βαθμός) θα τεθεί δυναμικά, ανάλογα με τη δουλειά που θα παρουσιάσει η κάθε ομάδα. Για το λόγο αυτό προσπαθήστε έστω και για ένα κομμάτι της άσκησης που βρίσκετε πιο κατανοητό – θα βαθμολογηθείτε ανάλογα.

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. Η διαδικασία είναι προφανώς ανάλογη αν κάποιος θέλει να προσθέσει system calls που να τα διαχειρίζεται ο Mermory Manager. 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_procfs_create).

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 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.

Βήμα 5 - Εναλλακτική περίπτωση: Αν ο κώδικας που έχετε να γράψετε είναι αρκετός, ίσως θελήσετε να τον εντάξετε σε ένα αυτοτελές αρχείο. Για να κάνετε κάτι τέτοιο, για παράδειγμα για να προσθέσετε το αρχείο procfs.c εξαρτώμενο από το procfs.h στον κατάλογο fs, θα πρέπει να κάνετε τα εξής:

- Ανοίγετε το Makefile στον κατάλογο /usr/src/fs και προσθέτετε στο τέλος της λίστας με τα περιεχόμενα της μεταβλητής OBJ το όνομα procfs.o
- Προσθέτετε στο τέλος του Makefile δύο γραμμές (ή όσες χρειάζονται) με τις κατάλληλες εξαρτήσεις του object file procfs.o:
  procfs.o: fs.h
  procfs.o: procfs.h

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. εδώ 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. /usr/src/oslab) ανοίξτε ένα καινούργιο header αρχείο (π.χ procfs_create.h) και συμπεριλάβέτε κάτι ανάλογο με το ακόλουθο:

```
#ifndef _PROCFS_CREATE_H_
#define _PROCFS_CREATE_H_

#include <lib.h>
#define procfs_create _procfs_create
#include <unistd.h>

int procfs_create(const char *_name)
{
  int len;
  message m;

/* αντιγραφή των παραμέτρων της συνάρτησης στο μήνυμα m */
/* σημειώστε τη μεταβλητή len: το μήκος της συμβολοσειράς στέλνεται */
/* από αυτό το σημείο – φροντίστε να το εφαρμόσετε */
...
...
...
  return (_syscall(FS, PROCFS, &m));
}

#endif
```

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 επιστρέψει 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 procfs_create() απλά κάντε include το procfs_create.h.

Kernel Compilation:

Για να χρησιμοποιήσετε την συνάρτηση procfs_create() 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.

Βιβλιογραφία

[1] Modern Operating Systems, Andrew S. Tanenbaum, 1st ed., Prentice Hall 1992
[2] Operating Systems, Design and Implementation, Andrew S. Tanenbaum and Albert S. Woodhull, 2nd ed., Prentice Hall 1996