Τμήμα Μηχ. Η/Υ & Πληροφορικής **Εαρινό 2019**
Πανεπιστήμιο Ιωαννίνων **Διδάσκων: Σ. Αναστασιάδης**

MYY601 Λειτουργικά Συστήματα
Ανακοίνωση: Τρίτη 26 Μαρτίου, Παράδοση: Παρασκευή 19 Απριλίου στις 21:00
Εργαστήριο 2: Υλοποίηση δίκαιης χρονοδρομολόγησης στο MINIX 3.2.0
Βαρύτητα 2ου εργαστηρίου: 20/30

1 Εισαγωγή

Σας δίνεται ο πηγαίος κώδικας του λειτουργικού συστήματος MINIX 3.2.0. Θα πρέπει να κάνετε εκκίνηση ενός αντιγράφου του συστήματος στο περιβάλλον VMware και να βεβαιωθείτε ότι η εγκατάσταση περνά κάποιες δοκιμές. Μετά θα επεκτείνετε το σύστημα για να υποστηρίξετε πολιτική δίκαιης χρονοδρομολόγησης και θα κάνετε ό,τι άλλες αλλαγές χρειάζονται για να δείξετε ότι ο χρονοδρομολογητής σας δουλεύει σωστά.

Εκκίνηση του Minix στο VMware

Κατεβάστε το αρχείο Minix3.2.0.zip (~419MB) και αποσυμπιέστε (με **unzip**) σε δίσκο USB μνήμης flash. Μετά ξεκινήστε τον VMware player που είναι εγκατεστημένος στα μηχανήματα Ubuntu του Τμήματος. Εναλλακτικά μπορείτε να κατεβάσετε και να εγκαταστήσετε τον δωρεάν VMware Workstation Player 15.0 στον προσωπικό σας υπολογιστή (ή 12.0 για παλιότερα μηχανήματα). Επιλέξτε την επιλογή **Open a Virtual Machine** προκειμένου να ανοίξετε το αρχείο **Minix3.2.0/Minix3.2.0.vmx**. Κάνετε κλικ στην επιλογή **Play virtual machine**. Μπορείτε ανά πάσα στιγμή να απελευθερώσετε το ποντίκι από το VMware πιέζοντας Ctr-Alt.

Στην συνέχεια κάνετε login ως χρήστης **root** (με κωδικό πρόσβασης 'root') για το υπόλοιπο της άσκησης. Προκειμένου να δοκιμάσετε την εγκατάσταση, εκτελέστε την εντολή **cd /usr/src/test; make; ./run**. Κανονικά, οι 63 (από τις 64) δοκιμές θα ολοκληρωθούν επιτυχώς χωρίς προβλήματα (αγνοήστε τη δοκιμή 48 που αλληλοεπιδρά με το www.minix3.org και αποτυγχάνει). Οι κατάλογοι **/usr/bin** και **/usr/sbin** περιέχουν διάφορα εργαλεία που μπορείτε να τρέξετε στο Minix 3.2.0.

Σε αυτό το σημείο εξασφαλίστε ότι είστε εξοικειωμένοι με το συντάκτη **vi**. Μπορείτε να ανοίξετε και χρησιμοποιήσετε πολλαπλά τερματικά ταυτόχρονα πατώντας Alt-F2, Alt-F3 κλπ. Εναλλακτικά μπορείτε να εισάγετε την εντολή **startx** για να εκκινήσετε το σύστημα X Windows. Μπορείτε να αλλάξετε το μέγεθος ενός παραθύρου κάνοντας αριστερό κλικ οπουδήποτε στην οθόνη, επιλέγοντας Resize στο μενού και κάνοντας κλικ στο παράθυρο του οποίου το μέγεθος θέλετε να αλλάξετε. Μπορείτε να κάνετε μετακίνηση προς τα πάνω/κάτω σε ένα τερματικό πιέζοντας Shift+PhUp/Down. Μπορείτε να βγείτε από το σύστημα X Windows για να επιστρέψετε πίσω στην κονσόλα του συστήματος πιέζοντας Alt+Ctrl+Backspace.

Τερματίστε το Minix εισάγοντας **halt** στη γραμμή εντολής. Σταματήστε την εικονική μηχανή κάνοντας κλικ στην επιλογή Shut Down Guest στο μενού κάτω από το Power. Αν σταματήσετε τη μηχανή χωρίς halt το σύστημα θα βρεθεί σε ασυνεπή κατάσταση την οποία θα πρέπει να διορθώσει (αυτόματα) στην επόμενη εκκίνηση.

Δομή του πηγαίου κώδικα του Minix

Μπορείτε να βρείτε τον πηγαίο κώδικα του Minix 3.2.0 διαθέσιμο στο διαδίκτυο. Στην παρουσίαση που ακολουθεί, περιγράφουμε σύντομα τον κώδικα που εμφανίζεται στον κατάλογο **/usr/src** του εγκατεστημένου Minix 3.2.0:

- Ο κατάλογος **include** περιέχει σημαντικά πρότυπα αρχεία. Για παράδειγμα, ο υποκατάλογος **include/minix** περιέχει αρχεία (π.χ., **com.h**, **syslib.h**, **type.h**) με σημαντικές δηλώσεις που χρησιμοποιούνται από το σύστημα.
- Ο υποκατάλογος **src/kernel** υλοποιεί μηνύματα, διεργασίες, και βασικούς οδηγούς στον πυρήνα του συστήματος, ο **src/servers/pm** περιέχει το διαχειριστή διεργασιών, ο **src/servers/vfs** περιέχει το σύστημα αρχείων, και ο **src/servers/sched** περιέχει τον

χρονοδρομολογητή.

- Ο κατάλογος **src/lib** περιέχει τον πηγαίο κώδικα των βιβλιοθηκών του συστήματος. Ο υποκατάλογος **src/lib/libsys** υλοποιεί την επικοινωνία του πυρήνα συστήματος με τις δεργασίες διακομιστή διεργασιών, αρχείων και χρονοδρομολογητή.
- Ο κατάλογος **src/tools** παρέχει εργαλεία που διευκολύνουν την οικοδόμηση (build) και εγκατάσταση του πυρήνα. Μπορείτε να εκτελέσετε **make** μέσα στον κατάλογο για να δείτε τις διαθέσιμες επιλογές.

Τροποποίηση του Minix

Μπορείτε να τροποποιήσετε τον κώδικα του Minix 3.2.0 στον κατάλογο **/usr/src**:

- Οι εικόνες (images) εκκίνησης του συστήματος είναι αποθηκευμένες στον κατάλογο **/boot/minix**. Αν τροποποιήσετε τον πηγαίο κώδικα, χρειάζεται να εκτελέσετε **make install** στον κατάλογο **/src/tools** προκειμένου να οικοδομήσετε το σύστημα. Η νέα σας εικόνα θα εμφανιστεί κάτω από τον κατάλογο **/boot/minix** ως υποκατάλογος με τον αντίστοιχο αριθμό διάθεσης (release number). Όταν κάνετε επανεκκίνηση το σύστημα, ο **boot monitor** επιλέγει την πιο πρόσφατα εγκαταστημένη εικόνα με την προκαθορισμένη ρύθμιση 2. Εναλλακτικά μπορείτε να εκκινήσετε το σύστημα με άλλη εικόνα επιλέγοντας τον κατάλληλο αριθμό που φαίνεται στο μενού του boot monitor, ή μπορείτε να εισάγετε 3 για να γυρίσετε σε χειρονακτική κατάσταση και να επιλέξετε την εικόνα από έναν κατάλογο που καθορίζετε με τις επόμενες εντολές:

  load_mods /boot/minix/3.2.0r7/mod*
  multiboot /boot/minix/3.2.0r7/kernel rootdevname=c0d0p0s0

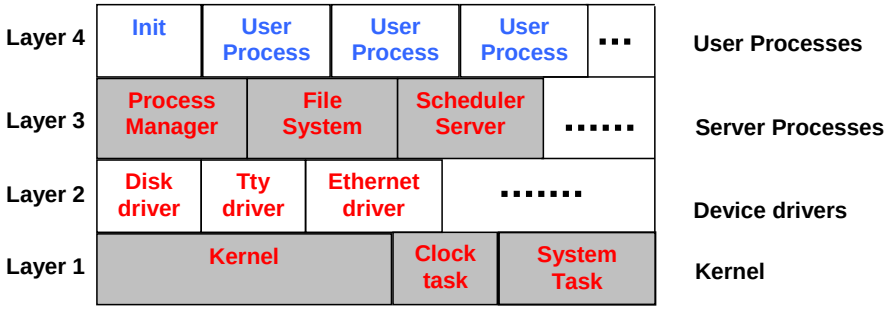
  Στο παραπάνω παράδειγμα χρησιμοποιείτε την εικόνα που βρίσκεται στον κατάλογο **/boot/minix/3.2.0r7**.

- Στον κώδικα των καταλόγων **src/kernel/, src/servers/pm**, **src/servers/fs**, και **src/servers/sched** μπορείτε να εμφανίσετε στην οθόνη μηνύματα εκσφαλμάτωσης εισάγοντας την εντολή **printf**.
- Μπορείτε να συνδεθείτε με το σύστημα Minix από το μηχάνημα στο οποίο τρέχει (π.χ., Linux, Windows, MacOS) με κάποια εφαρμογή ssh (π.χ., FileZilla για την μεταφορά αρχείων) αφού λάβετε την διεύθυνση IP του συστήματος Minix στην οποία θα συνδεθείτε με την εντολή **ifconfig**.
- Μπορείτε πάντα να δοκιμάσετε τις τροποποιήσεις του συστήματος εκτελώντας τις δοκιμές στον κατάλογο **/usr/src/test** εκτελώντας **./run** ως root.

Για παραπάνω πληροφορίες δείτε τους επίσημους οδηγούς User and Developer Guides του Minix 3.2.0.

Εσωτερική δομή του Minix

Το Minix 3 είναι δομημένο σε τέσσερα επίπεδα, καθένα από τα οποία εκτελεί μια σαφώς ορισμένη λειτουργία.



Το **επίπεδο 1** χειρίζεται τις διακοπές και τις παγίδες, αποθηκεύει και ανακτά τους καταχωρητές, κάνει βασική χρονοδρομολόγηση και προσφέρει στα ανώτερα επίπεδα ένα μοντέλο ανεξάρτητων ακολουθιακών διεργασιών που επικοινωνούν με μηνύματα. Επιπλέον το **επίπεδο 1** περιλαμβάνει την **εργασία ρολογιού (clock task)** που αλληλεπιδρά με το υλικό για τη δημιουργία σημάτων χρονισμού. Τέλος, περιλαμβάνει την **εργασία συστήματος (system task)** που διαβάζει/γράφει θύρες εισόδου/εξόδου και αντιγράφει δεδομένα μεταξύ των χώρων διευθύνσεων. Επικοινωνεί με το σύστημα αρχείων και το διαχειριστή διεργασιών μέσω μηνυμάτων (**src/lib/syslib**). Επιπλέον η **εργασία συστήματος** έχει πρόσβαση σε όλους τους πίνακες του πυρήνα.

Το **επίπεδο 2** περιέχει τις διεργασίες που προσπελάζουν τις συσκευές Ε/Ε. Τις ξεχωρίζουμε από τις άλλες διεργασίες χρησιμοποιώντας τον όρο **οδηγοί συσκευών**, επειδή έχουν ειδικά προνόμια για να ζητούν από την **εργασία συστήματος** να προσπελάσει συσκευές εισόδου/εξόδου.

Το **επίπεδο 3** περιέχει διεργασίες **διακομιστή** που παρέχουν χρήσιμες υπηρεσίες στις διεργασίες χρήστη. Τρέχουν σε προνομιακό επίπεδο χαμηλότερο των **επιπέδων 1** και **2**, δε μπορούν να προσπελάσουν τις θύρες Ε/Ε απευθείας και περιορίζονται στη μνήμη που τους δόθηκε. Ο διαχειριστής διεργασιών υλοποιεί κλήσεις συστήματος που σχετίζονται με τη διαχείριση διεργασιών (π.χ. **fork**), ενώ το σύστημα αρχείων υλοποιεί κλήσεις συστήματος αρχείων (π.χ. **read**). Οι διεργασίες **διακομιστή** ξεκινούν κατά την εκκίνηση του συστήματος και δεν τερματίζουν εφόσον το σύστημα είναι ενεργό.

Όλος ο κώδικας του **επιπέδου 1** διασυνδέεται σε ένα μοναδικό πρόγραμμα που αποτελεί τον πυρήνα. Ο κώδικας του **επιπέδου 2** υλοποιείται ως διεργασίες σε επίπεδο χρήστη. Ο κώδικας των διεργασιών **διακομιστή** του **επιπέδου 3** διασυνδέεται σε ξεχωριστά προγράμματα π.χ. **pm**, **vfs**, **sched**. Τέλος, ο κώδικας της διεργασία **init** διασυνδέεται ως ένα ξεχωριστό πρόγραμμα. Επομένως, το Minix 3 αποτελείται από διάφορα ανεξάρτητα προγράμματα τα οποία επικοινωνούν μεταξύ τους μόνο με μηνύματα.

Οι πληροφορίες ελέγχου των διεργασιών είναι διάσπαρτες σε πολλαπλούς πίνακες διεργασιών. Τα στοιχεία του πίνακα διεργασιών πυρήνα έχουν τύπο **struct proc** και ορίζονται στο αρχείο **src/kernel/proc.h**. Παρομοίως τα στοιχεία του πίνακα διεργασιών του διαχειριστή διεργασιών έχουν τύπο **struct mproc** και ορίζονται στο αρχείο **src/pm/mproc.h**. Είναι ενδιαφέρον ότι οι πληροφορίες που σχετίζονται με την ταυτότητα του χρήστη, την ομάδα των χρηστών και την ομάδα των διεργασιών συντηρούνται από το διαχειριστή διεργασιών στη δομή **mproc**.

Διαδιεργασιακή επικοινωνία στο Minix

Το Minix προσφέρει τρεις βασικές εντολές για αποστολή και λήψη μηνυμάτων. Μπορούν να κληθούν με χρήση των συναρτήσεων βιβλιοθήκης της C: **send(dest, &message)** για αποστολή **message** στην διεργασία **dest**, **receive(source, &message)** για λήψη **message** από τη διεργασία **source** ή **ANY**, και **send_rec(src_dst, &message)** για αποστολή **message** και αναμονή για απάντηση που αντικαθιστά το αρχικό **message**.

Όταν η διεργασία στέλνει ένα μήνυμα σε μια διεργασία που δεν περιμένει το μήνυμα, ο αποστολέας μπαίνει σε αποκλεισμό μέχρι ο παραλήπτης να καλέσει **receive**. Έτσι το Minix χρησιμοποιεί τη μέθοδο ραντεβού (**rendezvous**) για να αποφύγει την ενδιάμεση αποθήκευση των μηνυμάτων που έχουν σταλεί και δεν έχουν ληφθεί.

Σημειώστε ότι το Minix υλοποιεί τα μηνύματα ως δομές που περιέχουν την ένωση έξι διαφορετικών τύπων μηνυμάτων (**src/include/minix/ipc.h**). Έτσι, ένα μήνυμα είναι μια δομή που περιέχει το πεδίο **m_source** το οποίο καθορίζει ποιος έστειλε το μήνυμα, το πεδίο **m_type** που καθορίζει τον τύπο του μηνύματος, και τα πεδία δεδομένων. Για παράδειγμα, το πρώτος τύπος μηνύματος περιέχει τρεις ακέραιους (**m1i1**, **m1i2**, **m1i3**) και τρεις δείκτες (**m1p1**, **m1p2**, **m1p3**). Ο δεύτερος τύπος μηνύματος περιέχει τρεις ακέραιους (**m2i1**, **m2i2**, **m2i3**), δύο ακέραιους long (**m2l1**, **m2l2**) και έναν δείκτη (**m2p1**).

Θεωρώντας το μήνυμα **x**, μπορείτε να προσπελάσετε το πεδίο ένωση του μηνύματος με **x.m_u**, την πρώτη περίπτωση της ένωσης με **x.m_u.m_m1** και τον πρώτο ακέραιο με **x.m_u.m_m1.m1i1**. Επιπλέον, μπορείτε να χρησιμοποιήσετε το **x.m1_i1** ως βολική συντομογραφία του **x.m_u.m_m1.m1i1**.

Χρονοδρομολόγηση διεργασιών στο Minix

Ο πυρήνας συντηρεί ελάχιστες πληροφορίες χρονοδρομολόγησης για κάθε διεργασία στον πίνακα διεργασιών. Αυτές περιλαμβάνουν την ουρά προτεραιότητας της διεργασίας, τον εναπομείναντα χρόνο και το διακομιστή χρονοδρομολόγησης. Ο πυρήνας επιπλέον καταγράφει το αρχικό κβάντο που δόθηκε στη διεργασία. Αυτό θα το χρειαστεί ο πυρήνας μόνο όταν πρέπει να κάνει τη χρονοδρομολόγηση ο ίδιος, επειδή έχει καταρρεύσει ο διακομιστής χρονοδρομολόγησης. Η προκαθορισμένη πολιτική του πυρήνα είναι σχετικά απλή και εφαρμόζεται μόνο όταν το πεδίο **p_scheduler** της διεργασίας είναι **NULL**. Σε αυτή την περίπτωση, όταν τελειώνει το κβάντο μιας διεργασίας, η διεργασία διακόπτεται και τοποθετείται στο τέλος της τρέχουσας ουράς προτεραιότητας με νέο κβάντο (**p_quantum_size_ms**).

Ο πυρήνας διατηρεί δεκαέξι (**NR_SCHED_QUEUES** του **include/minix/config.h**) ουρές εκτελέσιμων διεργασιών. Σημειώστε ότι η υψηλότερη προτεραιότητα αντιστοιχεί στην τιμή 0 και η χαμηλότερη προτεραιότητα στην τιμή 15. Οι διακομιστές κανονικά χρονοδρομολογούνται σε ουρές με προτεραιότητες υψηλότερες από αυτές που επιτρέπονται για τις διεργασίες χρήστη, οι οδηγοί σε ουρές με προτεραιότητες υψηλότερες από εκείνες των διακομιστών, και το ρολόι μαζί με την εργασία συστήματος στην ουρά ύψιστης προτεραιότητας.

Ο βασικός κώδικας χρονοδρομολόγησης του πυρήνα εμφανίζεται στα αρχεία **src/kernel/proc.c** και **src/kernel/proc.h**. Η συνάρτηση **pick_proc()** στο **src/kernel/proc.c** είναι υπεύθυνη για την επιλογή της διεργασίας που τρέχει στη συνέχεια. Η καλούσα συνάρτηση **switch_to_user()** ενημερώνει τη σφαιρική μεταβλητή **proc_ptr** και κάνει την αλλαγή επιπέδου από πυρήνα σε χρήστη της επιλεγμένης διεργασίας. Οι αποφάσεις χρονοδρομολόγησης γίνονται όταν η διεργασία μπαίνει ή βγαίνει από τον αποκλεισμό, ή όταν ο χειριστής ρολογιού διαπιστώνει ότι μια δεργασία έχει ξεπεράσει το κβάντο.

Το Minix διαθέτει μια διεργασία διακομιστή χρονοδρομολόγησης (**scheduler**). Ο χρονοδρομολογητής διατηρεί το δικό του πίνακα διεργασιών που ορίζεται ως **schedproc** στο αρχείο **src/servers/sched/schedproc.h**. Όταν δημιουργείται μια διεργασία με την κλήση συστήματος **fork()**, η **do_fork()** του **forkexit.c** του **pm** επικοινωνεί με τους διακομιστές **vm** και **vfs**. Στη συνέχεια, ο **pm** αιτείται από το **sched** να χρονοδρομολογήσει τη διεργασία στον πυρήνα.

Κάθε φορά που τελειώνει το κβάντο μιας διεργασίας, ο χρονοδρομολογητής λαμβάνει ένα μήνυμα (**SCHEDULING_NO_QUANUM** στο **sched/main.c**) από τον πυρήνα για το γεγονός αυτό. Στη συνέχεια, ο χρονοδρομολογητής αυξάνει το πεδίο προτεραιότητας της διεργασίας κατά 1 και ειδοποιεί τον πυρήνα. Περιοδικά, ο χρονοδρομολογητής περνάει από τις διεργασίες και μειώνει το πεδίο προτεραιότητας κατά 1 για να τις βάλει σε ουρά υψηλότερης προτεραιότητας. Ο χρονοδρομολογητής χρησιμοποιεί την κλήση **sys_schedule()** για να τροποποιήσει την προτεραιότητα και το κβάντο μιας εκτελέσιμης διεργασίας στον πυρήνα.

Δίκαιη χρονοδρομολόγηση

Αρχή της δίκαιης χρονοδρομολόγησης είναι να διαιρέσει τις διεργασίες χρήστη σε ομάδες και να ισομοιράσει το χρόνο επεξεργαστή μεταξύ των ομάδων. Επιπλέον ισομοιράζει το χρόνο επεξεργαστή της κάθε ομάδας μεταξύ των διεργασιών της ομάδας. Για παράδειγμα, θεωρούμε

δύο ομάδες διεργασιών Α και Β. Θεωρούμε ότι η Α περιέχει μόνο μία διεργασία Α1, ενώ η Β περιέχει δύο διεργασίες Β1 και Β2. Τότε ο δίκαιος χρονοδρομολογητής αναθέτει από 50% του χρόνου επεξεργαστή σε κάθε ομάδα. Επομένως η Α1 λαμβάνει 50% του συνολικού χρόνου και οι Β1, Β2 λαμβάνουν από 25% η καθεμία.

Ένας τρόπος για να υλοποιήσετε δίκαιη χρονοδρομολόγηση είναι να εισάγετε τρία πεδία σε κάθε διεργασία: **fss_priority**, **process usage** και **group usage**. Το **group usage** θα έχει την ίδια τιμή για όλες τις διεργασίες στην ομάδα, δηλαδή τις διεργασίες που έχουν τον ίδιο **οδηγό ομάδας**. Ας υποθέσουμε ότι αρχικά όλες οι διεργασίες έχουν προτεραιότητα 0. Στο τέλος του κβάντο, ο χρονοδρομολογητής ενημερώνει την πληροφορία ελέγχου όλων των διεργασιών χρήστη ως εξής:

process_usage = process_usage/2;
group_usage = group_usage/2;
fss_priority = process_usage/2 + group_usage*number_of_groups/4 + base;

όπου **base = 0**. Τελικά, ο χρονοδρομολογής ειδοποιεί κατάλληλα τον πυρήνα να επιλέξει τη διεργασία με την χαμηλότερη τιμή **fss_priority**.

Συνεχίζοντας το παραπάνω παράδειγμα, υποθέτουμε ότι η διεργασία Α1 αρχίζει να εκτελείται με **quantum = 200ms**. Στο τέλος του κβάντο θα έχει: **process_usage = 200** και **group_usage = 200**. Μετά την ενημέρωση κατά την αλλαγή διεργασίας, οι παράμετροι γίνονται **process_usage=200/2=100**, **group_usage=200/2=100**, ενώ η προτεραιότητα γίνεται **fss_priority=100/2+100/(4/2)+0=100**.

Θεωρώντας ότι οι Β1 και Β2 δεν έλαβαν τον επεξεργαστή μέχρι στιγμής, θα έχουν και οι δύο **fss_priority=0**. Επειδή χαμηλότερη τιμή προτεραιότητας σημαίνει περισσότερα προνόμια, ο χρονοδρομολογητής επιλέγει την Β1 ή την Β2 ως επόμενη. Ας υποθέσουμε ότι η Β1 είναι η επόμενη. Στο τέλος του κβάντο, η Β1 θα έχει **process_usage=200**, **group_usage=200**. Μετά την ενημέρωση, η προτεραιότητά της γίνεται **fss_priority=100/2+100/2+0=100**. Παρομοίως, η Β2 θα έχει **process_usage=0**, **group_usage=200** και μετά την ενημέρωση η προτεραιότητά της γίνεται **fss_priority=0/2+100/2+0=50**. Τελικά, μετά την ενημέρωση η Α1 αποκτά προτεραιότητα **fss_priority=50/2+50/2+0=50**. Στη συνέχεια, ο χρονοδρομολογητής επιλέγει τις διεργασίες με τη σειρά Α1, Β2, Α1, Β1 κλπ.

2 Προετοιμασία

Μπείτε στο Minix και τρέξτε τις δοκιμές του **/usr/test** για να βεβαιωθείτε ότι όλα τρέχουν εντάξει. Εξοικειωθείτε με τη δομή του πηγαίου κώδικα, ειδικότερα με τα ακόλουθα αρχεία:

include	**minix/com.h, minix/config.h**, **minix/type.h**, **minix/ipc.h**,
kernel	**main.c**, **proc.h**, **proc.c**, **system.c**, **system/do_fork.c**, **system/do_schedule.c**
pm	**fork_exit.c, main.c**, **mproc.h**, **schedule.c**
sched	**schedule.h¸schedule.c**
vfs	**main.c**
sys/lib	**syslib/sched_start.c**

Στο Minix, πειραματιστείτε με το script **search <string> <directory>**. Παίρνει ως πρώτη παράμετρο μια συμβολοσειρά και ψάχνει αναδρομικά ξεκινώντας από τον καθορισμένο κατάλογο για όλα τα αρχεία με επέκταση **.c**, **.h**, **.s** που περιέχουν τη συμβολοσειρά. Π.χ. μπορείτε να καλέσετε **search do_schedule /usr/src | more** από τον κατάλογο **/usr/src** για να βρείτε όλα τα αρχεία που περιέχουν τη συμβολοσειρά **sys_schedule** στα περιεχόμενά τους.

3 Άσκηση

Η εργασία σας είναι να εισάγετε δίκαιη χρονοδρομολόγηση στο Minix 3.2.0. Τροποποιήστε τον πηγαίο κώδικα του Minix βαθμιαία με βάση τα επόμενα βήματα:

1. Η ταυτότητα ομάδας μιας διεργασίας είναι η ταυτότητα της πρώτης διεργασίας της ομάδας (**οδηγός ομάδας**). Βρείτε τρόπο για να μεταφέρετε την ταυτότητα του οδηγού ομάδας από

τη διεργασία **pm** στο **sched** κατά την κλήση **fork()**. Ένας τρόπος είναι να συμπεριλάβετε το πεδίο **rmc->mp_procgrp** στο μήνυμα που στέλνεται από τον **pm** στο **sched**. Επιβεβαιώστε ότι τη μέθοδός σας δουλεύει εμφανίζοντας στην οθόνη την ταυτότητα ομάδας του δημιουργούμενου παιδιού στη συνάρτηση **do_start_scheduling** του αρχείου **servers/sched/schedule.c** χρησιμοποιώντας **printf**.

2. Τροποποιήστε τη δομή **schedproc** στο **schedproc.h** για να συμπεριλάβετε τα πεδία **procgrp**, **proc_usage**, **grp_usage** και **fss_priority**. Όταν μια διεργασία ολοκληρώνει ένα κβάντο, ενημερώστε τα πεδία **procgrp**, **proc_usage**, **grp_usage** και **fss_priority** για όλες τις διεργασίες που περιμένουν στην ουρά χρήστη.
3. Στο **kernel/proc.h** μειώστε το συνολικό πλήθος ουρών έτσι ώστε όλες οι διεργασίες χρήστη να βρίσκονται σε μία μόνο ουρά. Τροποποιήστε τον πυρήνα για να επιλέγει τη διεργασία χρήστη με την χαμηλότερη τιμή **fss_priority** σύμφωνα με τη δίκαιη χρονοδρομολόγηση.
4. Για να δείξετε ότι ο κώδικάς σας δουλεύει, θα χρειαστεί να δημιουργήσετε ομάδες από διαφορετικές διεργασίες και να μετρήσετε το χρόνο που λαμβάνει προσεγγιστικά η κάθε διεργασία και ομάδα. Για παράδειγμα εκτελέστε από διαφορετικά τερματικά ομάδες από πολλαπλά scripts του τύπου:

```
#!/bin/sh
j=0
while [ 1 ] ; do
        i=0
        while [ $i -lt 10 ] ; do
                i=`expr $i + 1`
        done
        j=`expr $j + 1`
        echo $j
        sleep 1
done
```

Παρακολουθήστε το συνολικό χρόνο επεξεργαστή κάθε διεργασίας μέσω του εργαλείου **ps**, ή μέσω του ρυθμού εκτέλεσης της εντολής **echo** του script.

4 Τι θα παραδώσετε

Μπορείτε να ετοιμάσετε τη λύση ατομικά ή σε ομάδες των δύο. Ακόμη και αν η ομάδα σας αποτελείται από δύο μέλη, υποβάλετε την εργασία από το λογαριασμό του ενός. Υποβολή μετά την προθεσμία μειώνει το βαθμό 10% κάθε ημέρα μέχρι 50%. Για παράδειγμα, εάν υποβάλετε 1 ώρα μετά την προθεσμία ο μέγιστος βαθμός σας γίνεται 9 στους 10. Αν υποβάλετε μια εβδομάδα μετά την προθεσμία, ο μέγιστος βαθμός πέφτει στο 5 από 10. Υποβάλετε τη λύση σας με την εντολή

turnin lab2_19@myy601 README.pdf image.tar file1 …

Το αρχείο κειμένου **README.pdf** περιλαμβάνει τα ονόματα των φοιτητών της ομάδας. Επίσης περιέχει μια περιγραφή της λειτουργίας του χρονοδρομολογητή σας, τις βασικές δομές και συναρτήσεις που χρησιμοποιήσατε και τα αρχεία πηγαίου κώδικα στα οποία περιέχονται. Επιπλέον περιγράφει κανονικές και εξαιρετικές περιπτώσεις που χρησιμοποιήσατε για την εκσφαλμάτωση του κώδικα. Μαζί συμπεριλάβετε όλα τα αρχεία πηγαίου κώδικα που προσθέσατε ή τροποποιήσατε, τα αντίστοιχα **Makefile** που αλλάξατε καθώς και ένα tar αρχείο με τα δυαδικά αρχεία της εικόνας του πυρήνα σας στον κατάλογο **/boot/minix/<release id>**. Μη συμπεριλαμβάνετε **.o** ή εκτελέσιμα αρχεία εκτός του **image**, αλλά μπορείτε να συμπεριλάβετε scripts για τη δοκιμή του κώδικα. Ο κώδικάς σας πρέπει να μεταγλωττίζεται, διασυνδέεται και να τρέχει σε περιβάλλον VMware σε Ubuntu μηχανές του Τμήματος.