Τμήμα Μηχ. Η/Υ & Πληροφορικής **Εαρινό 2020**
Πανεπιστήμιο Ιωαννίνων **Διδάσκων: Σ. Αναστασιάδης**

ΜΥΥ601 Λειτουργικά Συστήματα
Ανακοίνωση: Τρίτη 7 Απριλίου, Παράδοση: Παρασκευή 8 Μαΐου 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 δείξετε με κατάλληλα πειράματα ότι ο χρονοδρομολογητής σας δουλεύει σωστά.

1.1 Εκκίνηση του Minix στο VMware

Κατεβάστε το αρχείο Minix3.2.0.zip (~419MB) και αποσυμπιέστε το στον εσωτερικό δίσκο ή εξωτερική USB μνήμη flash. Μπορείτε να κατεβάσετε και να εγκαταστήσετε τον δωρεάν VMware Workstation Player 15.5 στον προσωπικό σας υπολογιστή (ή 12.0 για παλιότερα μηχανήματα). Εκκινήστε την εικονική μηχανή του Minix κάνοντας κλικ στο αρχείο **Minix3.2.0/Minix3.2.0.vmx**. Μπορείτε ανά πάσα στιγμή να απελευθερώσετε το ποντίκι από το VMware πιέζοντας Ctr-Alt.

Για το υπόλοιπο της άσκησης κάνετε login ως χρήστης **root** (με κωδικό πρόσβασης 'root'). Ένα θέλετε να παραμείνετε στο περιβάλλον της κονσόλας μπορείτε να χρησιμοποιήσετε επιπλέον τερματικά με Alt-F2, κλπ. Διαφορετικά εκκινήσετε το γραφικό περιβάλλον X Windows εισαγάγοντας στο τερματικό την εντολή **startx**. Επιλέξτε πλήρη οθόνη με το εικονίδιο που εμφανίζεται πάνω αριστερά στο VMware. Για αλλαγή μεγέθους παραθύρου μέσα στο Minix κάνετε κλικ στην πάνω δεξιά γωνία του παραθύρου. Για άλλες λειτουργίες κάνετε κλικ οπουδήποτε στην οθόνη και επιλέξτε στο μενού την λειτουργία. Μετακινηθείτε πάνω/κάτω στο περιεχόμενο ενός τερματικού πιέζοντας Shift+PgUp/Down. Πιέστε Alt+Ctrl+Backspace αν θέλετε να βγείτε από το σύστημα X Windows και να επιστρέψετε πίσω στην κονσόλα του συστήματος (ή Ctrl-D στο πρώτο τερματικό αριστερά).

Σε αυτό το σημείο εξασφαλίστε ότι είστε εξοικειωμένοι με το συντάκτη **vi**. Προκειμένου να δοκιμάσετε την εγκατάσταση, εκτελέστε την εντολή **cd /usr/src/test; make; ./run**. Κανονικά, οι 63 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.

Τερματίστε το Minix εισάγοντας **halt** στη γραμμή εντολής (και **Cancel** στην ειδοποίηση του VMware για σφάλμα). Αν σταματήσετε τη μηχανή χωρίς halt το σύστημα θα βρεθεί σε ασυνεπή κατάσταση στην επόμενη επανεκκίνηση.

1.2 Δομή του πηγαίου κώδικα του Minix

Μπορείτε να βρείτε τον πηγαίο κώδικα (επιλογή **Files**) του Minix 3.2.0 διαθέσιμο στην σελίδα του μαθήματος. Ακολουθεί σύντομη παρουσίαση του κώδικα που εμφανίζεται στον κατάλογο **/usr/src** του συστήματος:

- 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 κατάλογος **kernel** υλοποιεί μηνύματα, διεργασίες, και βασικούς οδηγούς στον πυρήνα του συστήματος, ο **servers/pm** περιέχει το διαχειριστή διεργασιών, ο **servers/vm** τον διαχειριστή μνήμης, ο **servers/vfs** το σύστημα αρχείων, και ο **servers/sched** περιέχει τον χρονοδρομολογητή.
- Ο κατάλογος **lib** περιέχει τον πηγαίο κώδικα των βιβλιοθηκών του συστήματος. Ο υποκατάλογος **lib/libsys** υλοποιεί την επικοινωνία των δεργασιών διακομιστή (π.χ., διεργασιών, αρχείων και

χρονοδρομολογητή) με τον πυρήνα του συστήματος.

- Ο κατάλογος **tools** παρέχει εργαλεία που διευκολύνουν την μεταγλώττιση (build) και εγκατάσταση του πυρήνα. Μπορείτε να εκτελέσετε **make** μέσα στον κατάλογο για να δείτε τις διαθέσιμες επιλογές.

1.3 Τροποποίηση του Minix

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

- Τα images (εικόνες 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 μεταγλωττίσετε τον πηγαίο κώδικα τους συστήματος. Το νέο image θα εμφανιστεί κάτω από τον κατάλογο **/boot/minix** ως υποκατάλογος με τον αντίστοιχο αριθμό διάθεσης (release number). Όταν κάνετε επανεκκίνηση, ο **boot monitor** επιλέγει το πιο πρόσφατα εγκατεστημένο image με την προκαθορισμένη ρύθμιση **2**. Εναλλακτικά μπορείτε να εκκινήσετε το σύστημα με άλλο image 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 με τις επόμενες εντολές:

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

  Στο παραπάνω παράδειγμα χρησιμοποιείται το image που βρίσκεται στον κατάλογο **/boot/minix/3.2.0r7**.

- Στον πηγαίο κώδικα των υποκαταλόγων **kernel/, servers/pm**, **servers/fs**, και **servers/sched** του καταλόγου **/usr/src** μπορείτε να εμφανίσετε στην οθόνη μηνύματα εκσφαλμάτωσης εισάγοντας την εντολή **printf**.

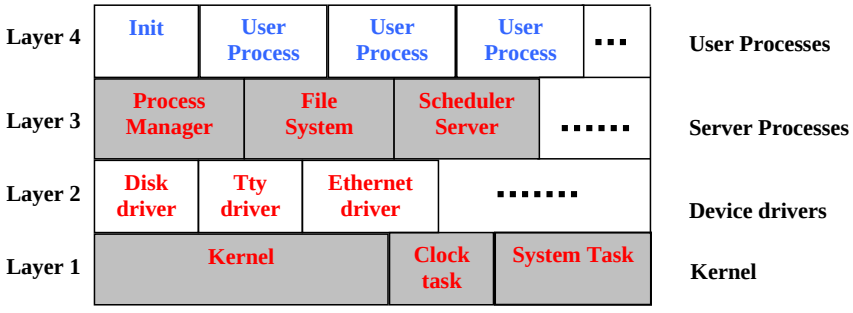
- Μπορείτε να συνδεθείτε με το σύστημα Minix από το μηχάνημα στο οποίο τρέχει το VMware με κάποια εφαρμογή ssh (π.χ., FileZilla για την μεταφορά αρχείων) αφού λάβετε την διεύθυνση IP του συστήματος Minix στην οποία θα συνδεθείτε με την εντολή **ifconfig**.

- 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, καθώς και το σύγγραμμα Operating Systems: Design and Implementation των A. Tanenbaum και A. S. Woodhull.

1.4 Εσωτερική δομή του Minix

Το Minix 3 είναι δομημένο σε τέσσερα επίπεδα, καθένα από τα οποία εκτελεί μια συγκεκριμένη κατηγορία λειτουργιών.



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 το **εργασία συστήματος** (**system task**) που διαβάζει/γράφει θύρες εισόδου/εξόδου και αντιγράφει δεδομένα μεταξύ των χώρων διευθύνσεων. Το σύστημα αρχείων και ο διαχειριστής διεργασιών επικοινωνούν με την εργασία συστήματος και λαμβάνουν απαντήσεις μέσω μηνυμάτων (**lib/libsys**). 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 (device drivers)**, επειδή έχουν το πρόνομιο να ζητούν από την εργασία συστήματος να προσπελάσει συσκευές εισόδου/εξόδου.

Το **επίπεδο 3** περιέχει **διεργασίες διακομιστή** (**server processes**) που παρέχουν χρήσιμες υπηρεσίες στις διεργασίες χρήστη. Τρέχουν σε προνομιακό επίπεδο αλλά 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 3.2.0 αποτελείται από διάφορα ανεξάρτητα προγράμματα τα οποία επικοινωνούν μεταξύ τους μόνο με μηνύματα.

Οι πληροφορίες ελέγχου των διεργασιών είναι διάσπαρτες σε πολλαπλούς πίνακες διεργασιών. Οι εγγραφές του πίνακα διεργασιών πυρήνα έχουν τύπο **struct proc** και ορίζονται στο αρχείο **kernel/proc.h**. Παρομοίως οι εγγραφές του πίνακα διεργασιών του διαχειριστή διεργασιών έχουν τύπο **struct mproc** και ορίζονται στο αρχείο **servers/pm/mproc.h**. Το αναγνωριστικά χρήστη, ομάδας χρηστών, διεργασίας, γονικής διεργασίας και ομάδας διεργασιών σημάτων συντηρούνται από το διαχειριστή διεργασιών στην δομή **mproc**.

1.5 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 το περιμένει, 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.

Το Minix υλοποιεί τα μηνύματα ως δομές που περιέχουν το union 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**).

1	m_source	m_type	m1_i1	m1_i2	m1_i3	m1_p1	m1_p2	m1_p3
2	m_source	m_type	m2_i1	m2_i2	m2_i3	m2_l1	m2_l2	m2_p1

.....

Θεωρώντας το μήνυμα **x**, μπορείτε να προσπελάσετε το union του μηνύματος με **x.m_u**, τον πρώτο

εναλλακτικό τύπο μηνύματος με **x.m_u.m_m1** και τον πρώτο ακέραιο με **x.m_u.m_m1.m1i1**. Επιπλέον, μπορείτε να χρησιμοποιήσετε το όνομα **x.m1_i1** ως βολική συντομογραφία του **x.m_u.m_m1.m1i1**.

Ειδικότερα, οι κλήσεις συστήματος του Minix υλοποιούνται με αποστολή μηνύματος από μια διεργασία στον κατάλληλο διακομιστή. Όπως μπορείτε να διαπιστώσετε από τα περιεχόμενα του καταλόγου **lib/libc/sys-minix**, για κάθε κλήση συστήματος υπάρχει ένα αρχείο που υλοποιεί την αποστολή μηνύματος μέσω της συνάρτησης **_syscall()** του αρχείου **syscall.c**.

1.6 Χρονοδρομολόγηση διεργασιών στο Minix

Ο πυρήνας διατηρεί δεκαέξι (**NR_SCHED_QUEUES** του **include/minix/config.h**) ουρές διεργασιών. Η υψηλότερη προτεραιότητα αντιστοιχεί στην ουρά με τιμή 0 και η χαμηλότερη προτεραιότητα στην ουρά με τιμή 15. Οι διεργασίες των διακομιστών χρονοδρομολογούνται με ουρές που έχουν 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.

Στο Minix 3.2.0 η πολιτική χρονοδρομολόγησης υλοποιείται από την διεργασία διακομιστή χρονοδρομολόγησης (**sched**). Μόνο στην περίπτωση που καταρρεύσει ο διακομιστής αυτός, αναλαμβάνει να υλοποιήσει την πολιτική χρονοδρομολόγησης ο ίδιος ο πυρήνας. Τότε ο πυρήνας εφαρμόζει μια απλή πολιτική σύμφωνα με την οποία μια διεργασία διακόπτεται όταν τελειώνει το κβάντο της και τοποθετείται στο τέλος της τρέχουσας ουράς με νέο κβάντο (**p_quantum_size_ms**).

Όταν μια διεργασία καλεί **fork()** για να δημιουργήσει μια καινούρια διεργασία, τότε ο διακομιστής διεργασιών (**pm**) καλεί την συνάρτηση **do_fork()** στο αρχείο **forkexit.c**. Η συνάρτηση αυτή καλεί **vm_fork()** για να επικοινωνήσει με τον διακομιστή μνήμης (**vm**) που αναλαμβάνει να δεσμεύσει την μνήμη και μετά να ενημερώσει τον πυρήνα για την νέα διεργασία. Στην συνέχεια ο **pm** καλεί **tell_vfs()** για να επικοινωνήσει με τον διακομιστή συστήματος αρχείων (**vfs**). Όταν ολοκληρωθεί η κλήση του **vfs** και λάβει ασύγχρονη απάντηση ο **pm**, τότε ζητάει από τον διακομιστή **sched** να χρονοδρομολογήσει τη διεργασία στον πυρήνα.

Ο διακομιστής **sched** διατηρεί έναν πίνακα διεργασιών που ονομάζεται **schedproc** στο αρχείο **src/servers/sched/schedproc.h**. Κάθε φορά που τελειώνει το κβάντο μιας διεργασίας, ο διακομιστής **sched** λαμβάνει ένα μήνυμα (**SCHEDULING_NO_QUANUM** στο **sched/main.c**) από τον πυρήνα για το γεγονός αυτό. Τότε ο διακομιστής **sched** αυξάνει το πεδίο προτεραιότητας της διεργασίας κατά 1 και ειδοποιεί τον πυρήνα. Περιοδικά, ο **sched** περνάει από τις διεργασίες και μειώνει το πεδίο προτεραιότητας κατά 1 για να τις βάλει σε ουρά υψηλότερης προτεραιότητας. Ο **sched** χρησιμοποιεί την κλήση **sys_schedule()** για να τροποποιήσει την προτεραιότητα και το κβάντο μιας εκτελούμενης διεργασίας στον πυρήνα.

Παρόλο που η πολιτική χρονοδρομολόγησης εκτελείται από τον **sched**, ο πυρήνας κάνει την διεκπεραίωση, δηλαδή βάζει την επόμενη διεργασία να τρέξει. Ο πυρήνας διατηρεί ελάχιστη πληροφορία χρονοδρομολόγησης για κάθε διεργασία στον πίνακα διεργασιών. Η πληροφορία αυτή περιλαμβάνει την ουρά προτεραιότητας της διεργασίας, τον εναπομένοντα χρονο του κβάνρο και την διεργασία του διακομιστή χρονοδρομολόγησης. Επίσης αποθηκεύει το αρχικό κβάντο που δόθηκε στην διεργασία.

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

1.7 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, έστω 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 A1 λαμβάνει 50% του συνολικού χρόνου και οι B1, B2 λαμβάνουν από 25% η καθεμία.

Ένας τρόπος για να υλοποιήσετε δίκαιη χρονοδρομολόγηση είναι να εισαγάγετε 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 A1 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 B1 και B2 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 B1 ή την B2 ως επόμενη. Ας υποθέσουμε ότι η B1 είναι η επόμενη. Στο τέλος του κβάντο, η B1 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 B2 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 A1 αποκτά προτεραιότητα **fss_priority=50/2+50/2+0=50**. Στη συνέχεια, ο χρονοδρομολογητής επιλέγει τις διεργασίες με τη σειρά A1, B2, A1, B1 κλπ.

1.8 Ομάδες Διεργασιών

Στο Minix οι διεργασίες είναι οργανωμένες σε ομάδες (**process groups**). Οι διεργασίες μιας ομάδας έχουν κοινό αναγνωριστικό ομάδας (**procgrp**) ίσο με το αναγνωριστικό (**pid**) της διεργασίας οδηγού (**leader)**. Για παράδειγμα αν δημιουργήσετε πολλαπλά τερματικά στο σύστημα X Windows με την εντολή **xterm**, τότε οι διεργασίες που προέρχονται από το ίδιο τερματικό ανήκουν στην ίδια ομάδα με αναγνωριστικό ομάδας αυτό του τερματικού (εμφανίζεται ως **sh** στην εντολή **ps al).** Για τις ανάγκες της άσκησης, μπορείτε απλώς να χρησιμοποιήσετε ως ομάδα τις διεργασίες που προέρχονται από ένα τερματικό και να δημιουργήσετε πολλαπλές ομάδες διεργασιών από διαφορετικά τερματικά.

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**
servers/pm	**fork_exit.c, main.c**, **mproc.h**, **schedule.c**
servers/sched	**schedule.h**,**schedule.c**
servers/vfs	**main.c**
lib	**libsys/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 Υλοποίηση

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. Προσδιορίστε πού αποθηκεύεται το αναγνωριστικό ομάδας (**procgrp**) μιας διεργασίας στον διακομιστή **pm**. Στην συνέχεια βρείτε τρόπο να μεταφέρετε το αναγνωριστικό ομάδας από τον διακομιστή **pm** στον διακομιστή **sched** κατά την κλήση συστήματος **fork()**. Ένας τρόπος είναι να συμπεριλάβετε το πεδίο **rmc->mp_procgrp** στο μήνυμα που στέλνει ο **pm** στον **sched**. Επιβεβαιώστε ότι η επικοινωνία δουλεύει εμφανίζοντας στην οθόνη το αναγνωριστικό ομάδας του δημιουργούμενου παιδιού. Για τον σκοπό αυτό καλέστε την **printf** στη συνάρτηση **do_start_scheduling()** του αρχείου **servers/sched/schedule.c**.
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. Θα χρειαστεί να υπολογίσετε δυναμικά το πλήθος των διεργασιών που ανήκουν στην ίδια ομάδα, για να ισομοιράσετε στην συνέχεια τον χρόνο στις ομάδες σύμφωνα με τον αλγόριθμο της δίκαιης χρονοδρομολόγησης.
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. Αυτό απαιτεί κατάλληλη επικοινωνία της επιλογής δίκαιης χρονοδρομολόγησης του διακομιστή **sched** προς τον πυρήνα που τελικά κάνει την διεκπεραίωση.
4. Για να δείξετε ότι ο κώδικάς σας δουλεύει, θα χρειαστεί να δημιουργήσετε ομάδες διεργασιών και να μετρήσετε το χρόνο που λαμβάνει προσεγγιστικά η κάθε διεργασία και ομάδα. Για παράδειγμα εκτελέστε από διαφορετικά τερματικά το ίδιο εκτελέσιμο και μετρήστε τον χρόνο που παίρνει το καθένα. Πειραματιστείτε με διαφορετικούς αριθμούς εκτελέσιμων στα τερματικά για να δείξετε ότι η δίκαιη χρονοδρομολόγηση δουλεύει μεταξύ ομάδων που η καθεμία έχει διαφορετικό πλήθος διεργασιών. Ενδεικτικά μπορείτε να παρακολουθήστε το συνολικό χρόνο επεξεργαστή κάθε διεργασίας μέσω του εργαλείου **ps**,

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. Σε περίπτωση που η ομάδα σας είναι ίδια με αυτή που είχατε στην πρώτη άσκηση, ας κάνει το turnin το μέλος της ομάδας που έκανε και στην πρώτη άσκηση για να προγραμματίσουμε την εξέταση. 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_20@myy601 README.pdf image.tar file1 …

Το αρχείο κειμένου **README.pdf** περιλαμβάνει τα ονόματα των μελών 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 του image της λύσης 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 στην εικονική μηχανή που σας δίνεται.