Πτυχιακη εργασια

Βελτιστοποίηση έξυπνων αποθηκευτικών συστημάτων: χρονοπρογραμματισμός εργασιών και προανάκτηση δεδομένων

Επιβλέποντες καθηγητές:

Β. Δημακόπουλος και Ε. Πιτουρά

Εξεταστής: Σ. Πάσχος

Κίναλης Αθανάσιος

Οκτώβριος 2001

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Στις μέρες μας βιώνουμε τη μετάβαση προς την εποχή της πληροφορίας. Ολοένα και περισσότεροι μεμονωμένοι άνθρωποι αλλά και οργανισμοί χρησιμοποιούν το διαδίκτυο για να ικανοποιήσουν βασικές τους ανάγκες, όπως η ενημέρωση και η ψυχαγωγία, η συλλογή και η ανταλλαγή πληροφοριών. Παράλληλα με την αύξηση της χρήσης του διαδικτύου αλλάζει και η μορφή των δεδομένων που διακινούνται, με την εισαγωγή νέων τύπων δεδομένων όπως ήχος, video, εικόνες από δορυφόρους. Τα παραπάνω οδηγούν σε μία αυξημένη ζήτηση για αποθηκευτικές ικανότητες και γρήγορη διαμεταγωγή δεδομένων.

Για την ικανοποίηση της ζήτησης αυτής έχουν επιστρατευθεί διάφορες τεχνολογικές λύσεις, όπως η αύξηση της αποθηκευτικής χωρητικότητας των σκληρών δίσκων και η αύξηση του εύρους ζώνης των δικτυακών συνδέσεων, οι οποίες όμως δεν επαρκούν. Μία άλλη προσέγγιση στην επίλυση αυτού του προβλήματος βασίζεται στην πιο αποτελεσματική χρήση της υπάρχουσας τεχνολογίας. Συγκεκριμένα η ύπαρξη σκληρών δίσκων με πολλά MB cache μνήμης, όπως ο Cheetah της Seagate, και το γεγονός ότι οι δίσκοι αυτοί διαθέτουν αρκετή υπολογιστική ισχύ ώστε να υποστηρίζουν πολύπλοκα πρωτόκολλα σύνδεσης με το δίαυλο, όπως το SCSI, οδηγεί στην αντίληψη ότι ένα μέρος της επεξεργασίας των δεδομένων μπορεί να μεταφερθεί στο δίσκο πριν την ανάκτηση και αποστολή τους στο δίκτυο από τον υπολογιστή. Η ιδέα αυτή εκφράζεται από την επινόηση των active disks [1, 3, 5] ή intelligent disks [2], συσκευών που αποτελούν βελτίωση των κοινών σκληρών δίσκων. Οι active disks έχουν προταθεί σαν αντικαταστάτες των ακριβών σκληρών δίσκων. Συνδέονται στο δίαυλο του συστήματος και διαθέτουν δικό τους επεξεργαστή (πολλές φορές αναλόγων δυνατοτήτων με έναν Intel Pentium) ο οποίος μπορεί να εκτελέσει μέρος της επεξεργασίας δεδομένων. Η επεξεργασία γίνεται με τη μεταφορά ενός μέρους της εφαρμογής στον active disk όπου και εκτελείται.

Η προτάσεις των active disks ως τώρα έχουν περιοριστεί στη αντικατάσταση των κοινών δίσκων που συνδέονται σε ένα εξυπηρέτη [4]. Πρακτικά όμως αυτή η χρήση προϋποθέτει την ανάπτυξη νέων αρχιτεκτονικών για τη σύνδεση πολλαπλών active disks σε ένα δίαυλο και την τροποποίηση των υπαρχόντων συστημάτων, κάτι το οποίο δεν είναι εφικτό. Μία βελτίωση αυτής της ιδέας είναι ένας δίσκος που χρησιμοποιεί μέρος της υπολογιστικής του δύναμης για να υλοποιήσει το πρωτόκολλο TCP/IP και να συνδεθεί κατευθείαν σαν αυτόνομη συσκευή στο δίκτυο. Την ιδέα αυτή υλοποιεί το σύστημα SmAS [6, 7] (SMart Autonomous Storage) που αναπτύσσεται στο Πανεπιστήμιο Ιωαννίνων[1]. Οι συσκευές SmAS είναι αυτόνομοι δίσκοι με ικανότητα σύνδεσης σε δίκτυο και διαθέτουν σημαντική υπολογιστική ισχύ, όπως οι active disks. Οι δίσκοι SmAS μειώνουν την ποσότητα δεδομένων που διακινείται στο δίκτυο με την τοπική εκτέλεση στο δίσκο εφαρμογών που επεξεργάζονται τα δεδομένα, ενώ έτσι κατανέμεται και η επεξεργασία των δεδομένων ανάμεσα στον client[2] και στο SmAS δίσκο, που αργότερα θα αναφέρεται και σαν εξυπηρέτης.

Παρόλο που στη συνέχεια εξετάζουμε αναλυτικά το σύστημα SmAS χρήσιμο είναι να αναφέρουμε επιγραμματικά μερικά από τα χαρακτηριστικά του εδώ. Το σύστημα SmAS εκτός από το υλικό, αποτελείται και από λογισμικό. Ένα τμήμα του λογισμικού αυτού βρίσκεται στους client, ενώ το μεγαλύτερο μέρος του βρίσκεται στον εξυπηρέτη. Το τμήμα του client περιλαμβάνει ένα API (Application Programming Interface) για επικοινωνία με τον εξυπηρέτη και μικρά προγράμματα, τα φίλτρα, που εκτελούνται στον εξυπηρέτη. Το τμήμα του εξυπηρέτη περιλαμβάνει ένα λειτουργικό σύστημα για τη διαχείριση του δίσκου, της μνήμης και του δικτύου ενώ η εξυπηρέτηση των client, συμπεριλαμβανομένης και της εκτέλεσης των φίλτρων, γίνεται με πολλαπλά νήματα, τον χρονοπρογραμματισμό των οποίων πρέπει να διαχειριστεί το λειτουργικό σύστημα του εξυπηρέτη.

Σκοπός της παρούσας εργασίας είναι η βελτίωση του ήδη υλοποιημένου συστήματος SmAS. Το σύστημα SmAS εκτελεί συγκεκριμένες και περιορισμένες λειτουργίες οπότε ένας τομέας τον οποίο μπορούμε να εξετάσουμε είναι η κατάλληλη επιλογή ενός λειτουργικού συστήματος για το SmAS. Στην περίπτωσή μας κατάλληλη επιλογή ισοδυναμεί με ένα "ελαφρύ", αποδοτικό και παραμετροποιήσιμο λειτουργικό σύστημα. Ακόμη η αρχιτεκτονική των πολλαπλών νημάτων και το γεγονός ότι αυτά εκτελούν φίλτρα, εγείρει συγκεκριμένες απαιτήσεις στο χρονο-

[1]Υποστηρίζεται μερικώς από τη Γενική Γραμματεία Έρευνας και Τεχνολογίας μέσω του ερευνητικού έργου ΠΕΝΕΔ-99/495

[2]Με την λέξη client εννοούμε μία εφαρμογή που εκτελείται σε έναν απομακρυσμένο από το δίσκο υπολογιστή, γενικά ένα χρήστη του συστήματος SmAS

προγραμματισμό τους και μία κατάλληλη πολιτική χρονοπρογραμματισμού μπορεί να βελτιώσει την απόδοση. Ακόμα μία άλλη βελτίωση είναι η μείωση του χρόνου ανάγνωσης και επεξεργασίας των δεδομένων από το δίσκο που μπορεί να επιτευχθεί με την υλοποίηση τεχνικών προανάκτησης δεδομένων (prefetching).

Στα πλαίσια αυτής της εργασίας, αρχικά ασχολούμαστε με την επιλογή ενός λειτουργικού συστήματος για το SmAS, στη συνέχεια τροποποιούμε την υλοποίηση των νημάτων (thread) αυτού του λειτουργικού και παραμετροποιούμε τον χρονοπρογραμματισμό τους προκειμένου να βελτιώσουμε την απόδοση του εξυπηρέτη. Τέλος προσθέτουμε τη δυνατότητα προανάκτησης δεδομένωνστο σύστημα προκειμένου να βελτιώσουμε την ταχύτητα επεξεργασίας των δεδομένων.

Η δομή της εργασίας έχει ως εξής. Αρχικά (Κεφάλαιο 2) περιγράφουμε την αρχική υλοποίηση και το προγραμματιστικό μοντέλο του SmAS και εξηγούμε τα κριτήρια επιλογής λειτουργικού συστήματος. Συνεχίζουμε (Κεφάλαιο 3) με την περιληπτική περιγραφή της αρχιτεκτονικής και των δυνατοτήτων του επιλεγμένου λειτουργικού συστήματος. Έπειτα (Κεφάλαιο 4) αναλύουμε την υλοποίηση του χρονοπρογραμματισμού και περιγράφουμε με βάση πειραματικά δεδομένα τα αποτελέσματα στην επίδοση του εξυπηρέτη. Ύστερα (Κεφάλαιο 5) αναλύουμε την υλοποίηση της προανάκτησης δεδομένων και περιγράφουμε με βάση πειραματικά δεδομένα τα αποτελέσματα στην επίδοση του εξυπηρέτη. Τέλος (Κεφάλαιο 6) κάνουμε ένα απολογισμό του αποτελέσματος της όλης προσπάθειας και προτείνουμε περαιτέρω βελτιώσεις στο σύστημα SmAS.

Κεφάλαιο 2

Περιγραφή του SmAS

Στο κεφάλαιο αυτό θα εξετάσουμε την αρχιτεκτονική (Ενότητα 2.1) και τις υπηρεσίες που παρέχει το σύστημα SmAS (Ενότητα 2.2), τον τρόπο υλοποίησης και προγραμματισμού του (Ενότητα 2.3) καθώς και τις ελάχιστες προϋποθέσεις που πρέπει να πληροί ένα λειτουργικό σύστημα μίας συσκευής SmAS και από ποιο θα μπορούσαν αυτές να ικανοποιηθούν (Ενότητα 2.4).

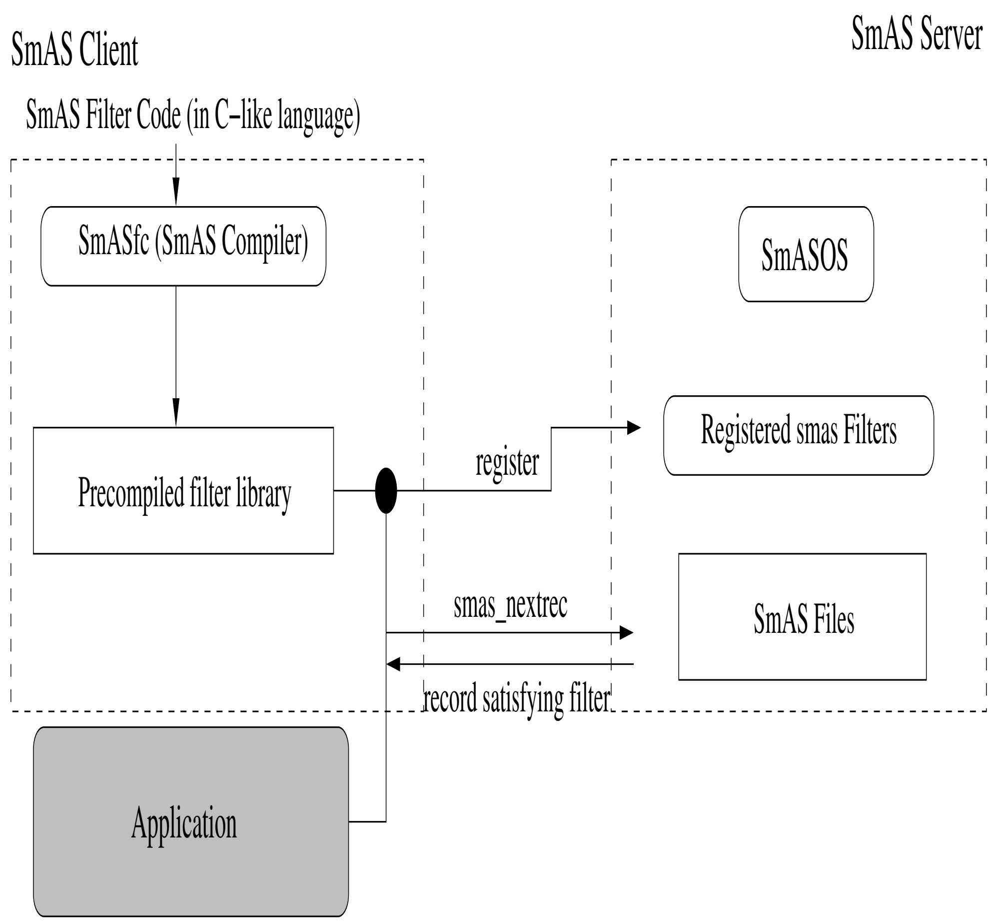
2.1 Αρχιτεκτονική SmAS

Η αρχιτεκτονική μίας SmAS συσκευής, σε επίπεδο υλικού, είναι αρκετά απλή. Ένας σκληρός δίσκος με κύρια μνήμη και ένα απλό επεξεργαστή, κλάσης Pentium για παράδειγμα, ώστε να εκτελεί λογισμικό και μία κάρτα δικτύου αποτελούν τις κύριες δομικές μονάδες της συσκευής SmAS.

Το σύστημα SmAS [6, 7] σε επίπεδο λογισμικού αρχικά μπορεί να διαχωριστεί σε δύο διαφορετικά τμήματα (Σχήμα 2.1), το τμήμα του εξυπηρέτη, δηλαδή το λογισμικό που εκτελείται στη SmAS συσκευή, και το τμήμα του client που εκτελείται σε κάθε μηχάνημα που προσπελαύνει δεδομένα στον εξυπηρέτη. Ειδική περίπτωση αποτελούν τα φίλτρα που είναι εκτελέσιμα κομμάτια λογισμικού που υπάρχουν τόσο στον εξυπηρέτη όσο και στον client.

Στον εξυπηρέτη το κυρίαρχο κομμάτι λογισμικού είναι το λειτουργικό σύστημα (SmASOS) το οποίο είναι επιφορτισμένο με την διαχείριση των δεδομένων του σκληρού δίσκου, την δικτυακή επικοινωνία με τον client και την εξυπηρέτηση των αιτήσεών του και την εκτέλεση των φίλτρων.

Στο client υπάρχει το SmAS API, το κομμάτι του λογισμικού που αναλαμβάνει την επικοινωνία με τον εξυπηρέτη, τα φίλτρα που δημιουργούνται από το χρήστη και ο SmASfc (SmAS filter compiler) ο μεταφραστής των φίλτρων.



Σχήμα 2.1: Η αρχιτεκτονική του SmAS

Τα φίλτρα είναι κομμάτια λογισμικού που δημιουργούνται στο client και στη συνέχεια μεταφέρονται και εκτελούνται μόνο στον εξυπηρέτη. Σκοπός τους είναι να επεξεργαστούν τα δεδομένα πριν αυτά σταλούν στο client. Έτσι όταν τα δεδομένα διαβαστούν από το δίσκο αποστέλλονται στο φίλτρο, το οποίο επιστρέφει τα δεδομένα που πρέπει να σταλούν στον client. Το φίλτρο μπορεί να αλλάξει τη μορφή των δεδομένων ή να στείλει μόνο συγκεκριμένα δεδομένα, οι επιλογές περιορίζονται από την φαντασία του προγραμματιστή και το τρόπο επικοινωνίας του φίλτρου με τον εξυπηρέτη, που περιγράφεται αργότερα. Τα φίλτρα γράφονται σε μία γλώσσα προγραμματισμού που μοιάζει με τη C και μεταφράζονται σε κατάλληλη μορφή για εκτέλεση στον εξυπηρέτη από το SmASfc. Το SmAS API αναλαμβάνει τη μεταφορά τους στον εξυπηρέτη όπου και το SmASOS τα εκτελεί μετά από αίτηση του client.

2.2 Υπηρεσίες SmAS

Προκειμένου να εξασφαλιστεί η χρήση του συστήματος SmAS με τη μεγαλύτερη δυνατή ευκολία υποστηρίζονται όλες οι λειτουργίες πάνω σε αρχεία που υπάρχουν στα συστήματα *UNIX* (`open(), read(), write(), ...`) με διαφάνεια προς το χρήστη ο οποίος χρησιμοποιεί το SmAS API. Έτσι το σύστημα μπορεί να αντικαταστήσει άνετα λειτουργίες που ως τώρα γίνονταν με συμβατικό τρόπο, για παράδειγμα με χρήση του NFS, ενώ προστέθηκαν νέες λειτουργίες που προσφέρουν ένα ξεκάθαρο και άνετο μέσο στο χρήστη για τη διαχείριση των φίλτρων.

Επιπλέον παρέχονται και υπηρεσίες για τη συγγραφή φίλτρων. Ειδικότερα κατασκευάστηκε μία γλώσσα προγραμματισμού που μοιάζει στη σύνταξη με τη **C** αλλά ορίζονται κάποιοι περιορισμοί όπως η αφαίρεση της δυναμικής διαχείρισης μνήμης και της χρήσης δεικτών, προκειμένου να διασφαλιστεί η απρόσκοπτη λειτουργία του εξυπηρέτη και η προστασία του από ελαττωματικά φίλτρα. Ακόμα ορίζεται συγκεκριμένος τρόπος εισαγωγής δεδομένων στο φίλτρο και επιστροφής των αποτελεσμάτων στον εξυπηρέτη για να τα μεταβιβάσει στο client. Ο SmASfc, ο μεταφραστής των φίλτρων, διασφαλίζει τους περιορισμούς που τέθηκαν και παράγει κώδικα έτοιμο προς εκτέλεση στον εξυπηρέτηr, ενώ το SmAS API διαχειρίζεται την αποστολή τους στον εξυπηρέτη και την εφαρμογή τους σε συγκεκριμένα δεδομένα.

Τα φίλτρα αποτελούν μια από τις πιο χρήσιμες υπηρεσίες του SmAS και είναι το μέσο με το οποίο μεταφέρεται τμήμα της εκτέλεσης της εφαρμογής του client στον εξυπηρέτη. Η μεγαλύτερη χρησιμότητα των φίλτρων είναι ότι μπορούν να μειώσουν τον όγκο των δεδομένων που αποστέλλονται στο client, όπως θα δούμε στη συνέχεια, αυξάνοντας την αποτελεσματικότητα της διαμεταγωγής δεδομένων ανάμεσα στο client και τον εξυπηρέτη.

2.3 Υλοποίηση SmAS

Επειδή η υλοποίηση του υλικού της συσκευής SmAS δεν υπάρχει, οπότε αρχικά χρησιμοποιήθηκε ένας υπολογιστής *Pentium* στα `166 MHz` με `32 MB RAM`, το υλικό αυτό είναι αντίστοιχο με αυτό που συναντάται σε σκληρούς δίσκους υψηλών επιδόσεων αλλά πιο οικονομικό. Για το SmASOS χρησιμοποιήθηκε ο πυρήνας *LINUX* με απενεργοποιημένες τις περισσότερες υπηρεσίες του λειτουργικού συστήματος *GNU/LINUX*, όπως ο *Xserver*, *web server* και διάφοροι άλλοι daemons.

Το κομμάτι του SmASOS που εξυπηρετεί τους clients υλοποιήθηκε σαν ένας daemon που "ακούει" σε μία συγκεκριμένη θύρα (port) για συνδέσεις client. Ο daemon είναι μια εφαρμογή εξυπηρέτη υλοποιημένη σε C. Η επικοινωνία γίνεται μέσω του κλασικού socket interface του *UNIX*. Με την σύνδεση ενός client στον εξυπηρέτη δημιουργείται ένα καινούργιο νήμα που διαχειρίζεται την επικοινωνία με το client. Όλες οι αιτήσεις του client εξυπηρετούνται από αυτό το νήμα, συμπεριλαμβανομένης και της εκτέλεσης των φίλτρων. Έτσι επιτυγχάνεται η ταυτόχρονη εξυπηρέτηση πολλών client. Πρόκειται δηλαδή για μία πολυνηματική υλοποίηση.

Ο client εισάγει αιτήσεις με κλήση των συναρτήσεων που είναι υλοποιημένες στο SmAS API. Το SmAS API έχει υλοποιηθεί σαν μία στατική βιβλιοθήκη. Οι συναρτήσεις είναι οι εξής [8]:

- `smas_open()` - Άνοιγμα ή δημιουργία ενός αρχείου
- `smas_close()` - Κλείσιμο ενός αρχείου
- `smas_read()` - Ανάγνωση από ένα αρχείου
- `smas_write()` - Εγγραφή σε ένα αρχείο
- `smas_lseek()` - Μετακίνηση της θέσης ανάγνωσης/εγγραφής σε ένα αρχείο
- `smas_registerfilter()` - Καταγραφή ενός φίλτρου στο server
- `smas_applyfilter()` - Εφαρμογή ενός φίλτρου σε ένα αρχείο
- `smas_nextrec()` - Ανάκτηση της εξόδου του φίλτρου

Οι τέσσερις πρώτες βρίσκονται σε πλήρη αντιστοιχία με τις κλήσεις συστήματος του *UNIX* `open()`, `close()`, `read()`, `write()` και `lseek()`.

Τα φίλτρα υλοποιούνται σαν shared libraries του *UNIX* οι οποίες αποστέλλονται στον εξυπηρέτη όπου φορτώνωνται δυναμικά και εκτελούνται. Ο εκτελέσιμος κώδικας περιλαμβάνει την υλοποίηση τριών συναρτήσεων μία για την αρχικοποίηση του φίλτρου, δηλαδή των τοπικών μεταβλητών του, μία για την αποπεράτωση και από μία συνάρτηση που επιτελεί την επεξεργασία των δεδομένων. Η συνάρτηση αυτή καλείται από το νήμα που εξυπηρετεί τον client με όρισμα τα δεδομένα που διαβάζονται από το δίσκο ενώ επιστρέφει τα δεδομένα που πρέπει να σταλούν στο client. Η μνήμη που χρησιμοποιεί ένα φίλτρο ορίζεται αποκλειστικά μέσω καθολικών μεταβλητών. Δεν δόθηκε δυνατότητα δυναμικής διαχείρισης μνήμης καθώς ένα φίλτρο θα μπορούσε να καταναλώσει μεγάλα ποσά μνήμης ανεξέλεγκτα, ανεπιθύμητο χαρακτηριστικό από τη στιγμή που η μνήμη είναι ένας περιορισμένος πόρος του συστήματος. Στην πραγματικότητα απαγορεύεται εντελώς η χρήση δεικτών για την αποτροπή παραβιάσεων μνήμης από τα φίλτρα.

Τέλος ο SmASfc μεταφράζει τον κώδικα σε καθαρή C, ελέγχοντας τους περιορισμούς που αναφέραμε, και χρησιμοποιεί τον `gcc` για τη δημιουργία της shared library. Στο σχήμα 2.2 φαίνεται ο κώδικας μίας απλής εφαρμογής που χρησιμοποιεί το φίλτρο του σχήματος 2.3 για να επεξεργαστεί ένα αρχείο δομημένο σε εγγραφές των `100 bytes`. Αρχικά η εφαρμογή δημιουργεί το αρχείο, γράφει 1000 εγγραφές, με τις γνωστές λειτουργιές αρχείων , και το κλείνει. Έπειτα ανοίγει πάλι το αρχείο αποστέλλει ένα φίλτρο στον εξυπηρέτη και το εφαρμόζει στα περιεχόμενα του αρχείου . Με επαναλαμβανόμενες κλήσεις της `smas_nextrec()` λαμβάνει τις εγγραφές που επιλέγονται από το φίλτρο του σχήματος 2.3. Το φίλτρο δηλώνει τη μνήμη που χρησιμοποιεί μέσω της δομής `myrec` η οποία είναι και η είσοδός του αφού είναι το όρισμα της `body()`. Η `body()` επιλέγει να περάσει στο client τις εγγραφές με πεδίο `key` μικρότερο ίσο του 15. Στο σχήμα 2.4 φαίνεται η ανταλλαγή μηνυμάτων ανάμεσα στον εξυπηρέτη και στον client και η αλληλεπίδραση του εξυπηρέτη με το φίλτρο σε μία τυπική σύνοδο ενός client με τον εξυπηρέτη.

2.4 Λειτουργικό σύστημα SmAS

Είναι φανερό από όσα έχουμε αναφέρει ως τώρα ότι το λειτουργικό σύστημα για το SmAS θα πρέπει να είναι ιδιαίτερα απλό. Συγκεκριμένα μπορούμε να απαριθμήσουμε τις λειτουργιές που πρέπει να επιτελεί το SmASOS στις ακόλουθες:

- **Διαχείριση δίσκου** - Το λειτουργικό σύστημα ασφαλώς πρέπει να μπορεί να προσπελαύνει τα αρχεία στο δίσκο και να υλοποιεί ένα σύστημα αρχείων.

```
#define FILENAME "smas.cs.uoi.gr:testfiles/test1"

typedef struct {
        int key;
        char data[96];
} MyRecord;

main()
{
        int fd,k,filtid,rsize;
        MyRecord r;

/*Part 1: create a sample file*/
        fd=smas_open(FILENAME, SMAS_O_CREAT|SMAS_O_RDWR);
        for (k=0;k<1000;k++) {
                r=randomrecord(); /*Creates a record*/
                smas_write(fd,&r,sizeof(r));
        }
        smas_close(fd);

/*Part 2: filter the records*/
        fd=smas_open(FILENAME, SMAS_O_RDONLY);
        filtid=smas_registerfilter(fd,"simpleselect");
        smas_applyfilter(fd, filtid);
        while (smas_nextrec(fd, &r, &rsize) > 0)
                process_rec(r);
}
```

Σχήμα 2.2: Ένας απλός client

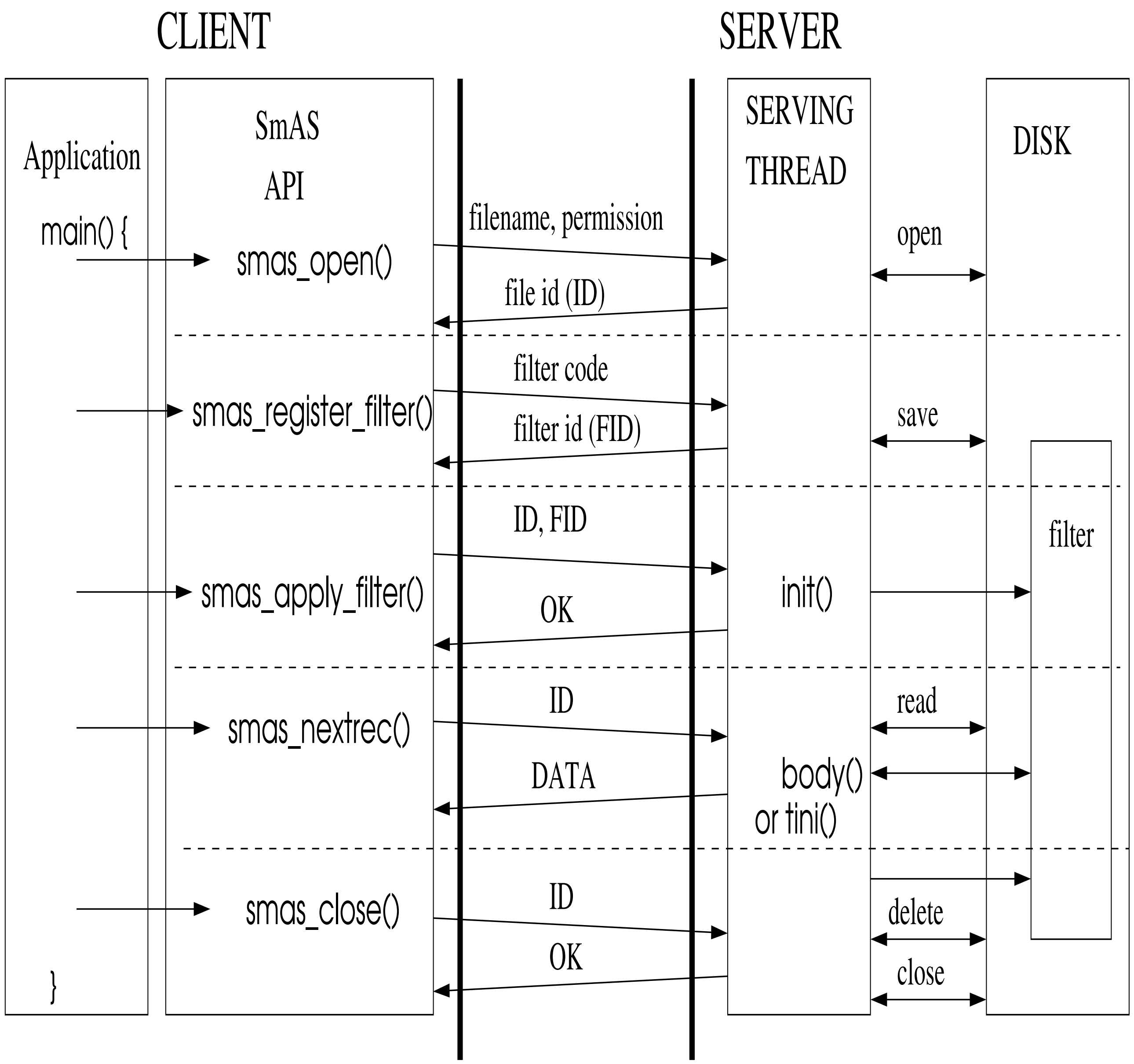
```
typedef struct {
        int key;
        char data[96];
} myrec;

init() { }

body(myrec r)
{
       if (r.key <= 15)
               pass(r, sizeof(r));
       else
               nopass();
}

tini() { }
```

Σχήμα 2.3: Ένα απλό φίλτρο



Σχήμα 2.4: Επικοινωνία server και client

- **Διαχείριση μνήμης** - Βασική λειτουργία για κάθε λειτουργικό σύστημα είναι η διαχείριση μνήμης, που χρειάζεται για τη μνήμη που χρησιμοποιούν τα φίλτρα και ο εξυπηρέτης.

- **Χρονοπρογραμματισμός** - Ο χρονοπρογραμματισμός των φίλτρων είναι μία ιδιότητα του συστήματος που αποδεικνύεται πολύ σημαντική όταν εξυπηρετούμε πολλούς client ταυτόχρονα.

- **Διαδικτύωση** - Η υλοποίηση του πρωτοκόλλου TCP/IP είναι βασική για το σύστημά μας.

Με βάση τα παραπάνω επιλέξαμε να μεταφέρουμε το σύστημα SmAS σε ένα νέο λειτουργικό το οποίο θα μπορεί να παραμετροποιηθεί εύκολα ώστε να υποστηρίζει τις παραπάνω λειτουργίες, ενώ θα αναστέλλονται οι υπόλοιπες αποφεύγοντας έτσι άσκοπη χρήση του επεξεργαστή. Η επιλογή μας ήταν ο υπό ανάπτυξη πυρήνας του *GNU* λειτουργικού συστήματος, το *Hurd*. Το *Hurd* έχει μία πρωτοποριακή αρχιτεκτονική που ξεφεύγει από την παραδοσιακή μονολιθική αρχιτεκτονική των υπόλοιπων κλώνων του *UNIX* και επιτρέπει την εύκολη παραμετροποίησή του. Το γεγονός ότι πρόκειται για ένα νέο πυρήνα ισοδυναμεί με σχετικά μικρό μέγεθος κώδικά, που βολεύει τους σκοπούς αυτής της εργασίας και την τροποποίηση του συστήματος. Ακόμη, οι μελλοντικές πολλά υποσχόμενες δυνατότητές του το καθιστούν ένα πολύ ενδιαφέρον σύστημα για μελέτη.

Παρόλο που εξετάσαμε και άλλες επιλογές το *Hurd* φάνηκε σαν η καλύτερη λύση. Συγκεκριμένα η δεύτερη επιλογή μας ήταν ο πυρήνας *LINUX* [23]. Παρόλο που αυτή η λύση έχει το πλεονέκτημα ότι αποτελεί ένα πολύ δημοφιλή πυρήνα, ο οποίος είναι αρκετά παραμετροποιήσιμος και βελτιστοποιημένος, τελικά απορρίφθηκε. Ο κυριότερος λόγος είναι το τεράστιο μέγεθος του κώδικά του και η μονολιθική αρχιτεκτονική του. Πραγματικά η μονολιθική αρχιτεκτονική του *LINUX* οδηγεί από τη μία σε ένα αποδοτικό σύστημα αλλά από την άλλη δημιουργεί πολλές εξαρτήσεις ανάμεσα στα διαφορετικά τμήματα του κώδικά του, γεγονός που καθιστά οποιαδήποτε ριζική αλλαγή πρακτικά αδύνατη.

Στο επόμενο κεφάλαιο παρουσιάζεται ο πυρήνας *Hurd* και οι δυνατότητές του ώστε να γίνουν ξεκάθαροι οι λόγοι που οδήγησαν στην επιλογή του και οι αλλαγές που έγιναν.

Κεφάλαιο 3

Ο πυρήνας Hurd

Το 1984 ο Richard Stallman ξεκίνησε μία προσπάθεια για την ανάπτυξη ενός ελεύθερου, από κάθε περιορισμό διάθεσης και διακίνησης του κώδικα, λειτουργικού συστήματος. Η προσπάθεια αυτή συνεχίζεται μέχρι σήμερα και ονομάζεται *GNU*[1] *project* [16] και ένα σημαντικό τμήμα της είναι η δημιουργία ενός νέου πυρήνα, για αυτό το σύστημα, του *Hurd*[2] [17]. Η ανάπτυξη του *Hurd* ουσιαστικά άρχισε το 1991 όταν το *Free Software Foundation*[3] *(FSF)* πήρε άδεια να χρησιμοποιήσει τον μικροπυρήνα *Mach* που είχε αναπτυχθεί από το *Carnegie Mellon University.*

Η αρχιτεκτονική του *Hurd* περιλαμβάνει το μικροπυρήνα *GNU/Mach*, σαν βάση για τη χαμηλού επιπέδου διαχείριση και προσπέλαση του υλικού, και ένα σύνολο από ειδικές εφαρμογές, τους *servers*, που αναλαμβάνουν να υλοποιήσουν τη λειτουργικότητα που βρίσκουμε σε συστήματα *UNIX*. Ακολουθεί μία σύντομη εισαγωγή (Ενότητα 3.1) στο μικροπυρήνα *Mach* και έπειτα παρουσιάζουμε την αρχιτεκτονική του *Hurd* (Ενότητα 3.2).

3.1 Εισαγωγή στο Mach

Η ανάπτυξη του μικροπυρήνα *Mach* [19], άρχισε το 1985 στο *Carnegie Mellon University (CMU).* Στόχος ήταν η δημιουργία ενός μικροπυρήνα που υποστηρίζει:

- Μία αρχιτεκτονική μικροπυρήνα που παρέχει περιορισμένες λειτουργίες και επιτρέπει σε επιπέδου χρήστη εφαρμογές, τους servers, να παρέχουν διάφορα συστήματα υποστήριξης εφαρμογών.

[1]'Οπου GNU σημαίνει GNU's not UNIX.

[2]'Οπου *Hurd* σημαίνει Hird of UNIX-Replacing Daemons και Hird σημαίνει Hurd of Interfaces Representing Depth.

[3]Οργάνωση που ιδρύθηκε το 1985 με στόχο την υποστήριξη του GNU project

- Ένα εξελιγμένο σύστημα διαδιεργασιακής επικοινωνίας στον πυρήνα που χρησιμεύει για την υλοποίηση του υπόλοιπου συστήματος.

- Αποδοτικά νήματα.

- Παροχή εικονικής μνήμης μέσω του πυρήνα και των server.

- Αρχιτεκτονικές με πολλαπλούς επεξεργαστές.

Μέχρι το 1994 οι άνθρωποι του *CMU* είχαν επιτύχει αυτούς τους στόχους με τη δημιουργία του *Mach* ενός μικροπυρήνα που έχει επηρεάσει σημαντικά αυτό τον τομέα των λειτουργικών συστημάτων από τότε. Η τελευταία έκδοση του *Mach* από το *CMU* είχε αριθμό τρία. Έπειτα το Πανεπιστήμιο της Utah συνέχισε, μέχρι το 1996, την εργασία του *CMU* με την προσθήκη νέων λειτουργιών και τη δημιουργία μίας τέταρτης έκδοσης του *Mach* [20]. Σήμερα αυτή η έκδοση χρησιμοποιείται στο *xMach* [21] μία προσπάθεια δημιουργίας ενός *BSD* [22] συστήματος. Παράλληλα το *FSF* ανέπτυξε τη δική του έκδοση του *Mach* για χρήση στο *GNU project*.

Αρχικά παρουσιάζουμε εν συντομία μερικά βασικά χαρακτηριστικά του *Mach* ενώ έπειτα εξετάζουμε το σύστημα διαχείρισης μνήμης και διεργασιών ώστε να γίνει κατανοητό πως επιτυγχάνεται η δημιουργία διεργασιών σε ένα τέτοιο σύστημα. Ο *Mach* αναλαμβάνει πλήρως τη διαχείριση του υλικού και παρέχει τη δυνατότητα δημιουργίας υψηλού επιπέδου server για τη παροχή περαιτέρω λειτουργικότητας. Η δυνατότητα αυτή δίνεται με την υποστήριξη από το *Mach* βασικών εννοιών. Οι κυριότερες από αυτές είναι [10]:

- **Task** - Αποτελεί τη μονάδα εκχώρησης πόρων, έχει ένα χώρο διευθύνσεων στη μνήμη και περιέχει ένα ή περισσότερα thread. Ουσιαστικά είναι μία συλλογή πόρων του συστήματος, οι οποίοι είναι προσπελάσιμοι είτε μέσω ports είτε άμεσα.

- **Thread** - Αποτελεί τη μονάδα αξιοποίησης της CPU, είναι ένα σημείο ελέγχου ροής μέσα σε ένα task, μπορεί να εκτελείται παράλληλα με άλλα thread και περιέχει ελάχιστη πληροφορία για μικρή κατανάλωση πόρων.

- **Port** - Είναι ένα κανάλι επικοινωνίας, μία ουρά μηνυμάτων που χειρίζεται ο μικροπυρήνας. Η πρόσβαση στην port γίνεται μέσω δικαιωμάτων, μόνο ένα task έχει δικαίωμα λήψης αλλά πολλά μπορεί να έχουν δικαίωμα αποστολής.

- **Message** - Αποτελεί μία τυποποιημένη συλλογή δεδομένων που χρησιμοποιείται στην επικοινωνία ανάμεσα σε tasks, δηλαδή στα περιεχόμενά τους threads.

- **Memory object** - Αποτελεί τη μονάδα διαχείρισης μνήμης του πυρήνα.

Τα παραπάνω αντικείμενα είναι ότι χρειάζεται για την υλοποίηση υψηλού επιπέδου εφαρμογών. Τα task και τα thread υλοποιούν την έννοια της διεργασίας ενώ τα ports και τα messages δημιουργούν ένα εξελιγμένο σύστημα διαδιεργασιακής επικοινωνίας, που χρησιμοποιείται στην επικοινωνία των task μεταξύ τους, στην επικοινωνία με τον μικροπυρήνα και στην προσπέλαση του υλικού. Το σύστημα επικοινωνίας είναι αρκετά εξελιγμένο ώστε να υποστηρίζει έλεγχο πρόσβασης στα port μέσω δικαιωμάτων, και αποστολή αυτών των δικαιωμάτων σε ένα server προκειμένου να επικοινωνήσει με κάποιον άλλο.

3.1.1 Μοντέλο διαχείρισης μνήμης Mach

Ο *Mach* χρησιμοποιεί ένα εξελιγμένο μοντέλο διαχείρισης μνήμης που χωρίζεται σε ένα τμήμα εξαρτούμενο από το υλικό και ένα τμήμα ανεξάρτητο από το υλικό που προσφέρει ένα μέσο για την υλοποίηση διαχείρισης μνήμης υψηλού επιπέδου. Ο *Mach* παρέχει ένα σύνολο από εικονικές διευθύνσεις (virtual addresses) [11, 12] στο χώρο διευθύνσεων ενός task που αντιστοιχούν σε φυσικές διευθύνσεις της μνήμης του συστήματος. Όπως σε όλα τα συστήματα που χρησιμοποιούν σελιδοποίηση (memory paging) [11, 12] για την υλοποίηση εικονικής μνήμης, έτσι και στο Mach ο χώρος διευθύνσεων ενός task δεν βρίσκεται ολόκληρος στη φυσική μνήμη. Την μεταφορά σελίδων από και πρός το χώρου διευθύνσεων ενός task αναλαμβάνει ένας εξωτερικός, του πυρήνα, server, ο memory manager. Ο *Mach* χρησιμοποιεί τη φυσική μνήμη σαν μία cache για τις σελίδες μνήμης που διαχειρίζονται διάφοροι memory manager. Ο *Mach* αναλαμβάνει όλα τα θέματα που αφορούν την πρόσβαση των σελίδων μνήμης από τα task, όσο αυτές βρίσκονται στη φυσική μνήμη, και παράλληλα αποφασίζει ποιές σελίδες θα παραμένουν στη φυσική μνήμη. Οι memory manager αναλαμβάνουν την αποθήκευση σελίδων εκτός της μνήμης και την ανάκτησή τους όταν αυτές χρειαστούν.

Η διαχείριση της μνήμης γίνεται μέσω των memory object. Ο Mach χρησιμοποιεί δύο ειδών memory object:

- **Abstract memory object** - Παρέχει το μέσο με το οποίο ένας memory manager μπορεί να χειριστεί τις σελίδες του και να τροποποιήσει τις ιδιότητες τους, όταν αυτές βρίσκονται εκτός της φυσικής μνήμης.

- **Memory cache object** - Το διαχειρίζεται ο πυρήνας και περιέχει την κατάσταση των σελίδων ενός abstract memory object, που βρίσκονται στη φυσική μνήμη.

Ένας memory manager είναι ένας server που δημιουργεί και αρχικοποιεί ένα abstract memory object και συνεπώς τη μνήμη που θα διαχειριστεί. Η διαχείριση μνήμης επιτυγχάνεται με τη συνεχή επικοινωνία του μικροπυρήνα με το memory manager. Όταν κάποιο task δημιουργήσει κάποιο σφάλμα σελίδας σε κάποιο memory cache object και δεν μπορεί να το διαχειριστεί ο μικροπυρήνας, τότε ο *Mach* μεταφέρει την αίτηση στο κατάλληλο memory manager. Ο memory manager αναλαμβάνει τη διαχείριση των σελίδων, με την εύρεση της ζητούμενης σελίδας στο abstract memory object του και την αποστολή της στο *Mach*. Γενικά το πρωτόκολλο επικοινωνίας του *Mach* με τους memory manager είναι πολύ περίπλοκο αφού πρέπει να διαχειρίζεται όλες τις περιπτώσεις σφαλμάτων μνήμης.

Ο default pager είναι ένας memory manager με ειδική σημασία για το *Mach*. Όταν κάποιος memory manager καθυστερήσει ή αποτύχει να διαχειριστεί μία σελίδα, τότε ο *Mach* καλεί το default pager ο οποίος δεν πρέπει να αποτυγχάνει ποτέ. Στο *Hurd* ο default pager είναι και ο μόνος memory manager που υπάρχει.

3.1.2 Tasks και threads στο Mach

Στο Mach δεν υπάρχει η έννοια της διεργασίας, τουλάχιστον όχι όπως την ξέρουμε στο UNIX. Αντίθετα ο Mach παρέχει τις δομές εκείνες που επιτρέπουν την υλοποίηση διεργασιών σε πιο υψηλό επίπεδο. Οι δομές αυτές είναι τα threads και τα tasks. Επομένως ο Mach παρέχει δυνατότητα διαχείρισης μόνο των task και των thread.

Τα thread αποτελούν τις βασικές οντότητες υπολογισμού. Κάθε thread ανήκει σε ένα και μόνο task που καθορίζει το χώρο διευθύνσεών του, τον οποίο και μοιράζεται με τα υπόλοιπα thread του task. Ο χρονοπρογραμματισμός των thread γίνεται από το *Mach*. Κάθε thread σχετίζεται με μία προτεραιότητα ανάλογα με την οποία αποφασίζεται πότε θα χρησιμοποιήσει τη CPU, ενώ η αλλαγή ανάμεσα στα thread γίνεται προεκχωρητικά. Υπάρχουν ειδικές κλήσεις με τις οποίες ένα thread μπορεί να παραδώσει τη CPU ή να αποκτήσει ειδικά προνόμια χρήσης της CPU και της μνήμης, όπως γίνεται με τα thread του default pager. Αξιοπρόσεκτο είναι ότι ο *Mach* είναι ειδικά σχεδιασμένος για να εκμεταλλεύεται στο έπακρο τις δυνατότητες παραλληλίας των μηχανημάτων με πολλαπλούς επεξεργαστές, ενώ η προηγμένη διαχείριση των thread του, του επιτρέπει την αποστολή σε άλλο μηχάνημα με μικροπυρήνα *Mach* ενός thread για να συνεχίσει εκεί την εκτέλεσή του, αν και αυτή τη δυνατότητα δεν την εκμεταλλεύεται ακόμα ο *Hurd*.

Τα tasks αποτελούν το μέσο εκχώρησης πόρων από το σύστημα στο περιβάλλον το οποίο εκτελούνται τα thread. Η δημιουργία ενός task γίνεται μόνο από κάποιο άλλο task από το οποίο και κληρονομεί διάφορες παραμέτρους και θέτει το χώρο εικονικών διευθύνσεών του. Παρόλο που ένα task δεν χρονοπρογραμματίζεται άμεσα, υπάρχει τρόπος να επηρεάσει το χρονοπρογραμματισμό των thread που περιέχει θέτοντας κάποιες παραμέτρους, ενώ μπορεί να ελέγξει το τερματισμό και την εκκίνηση των thread του.

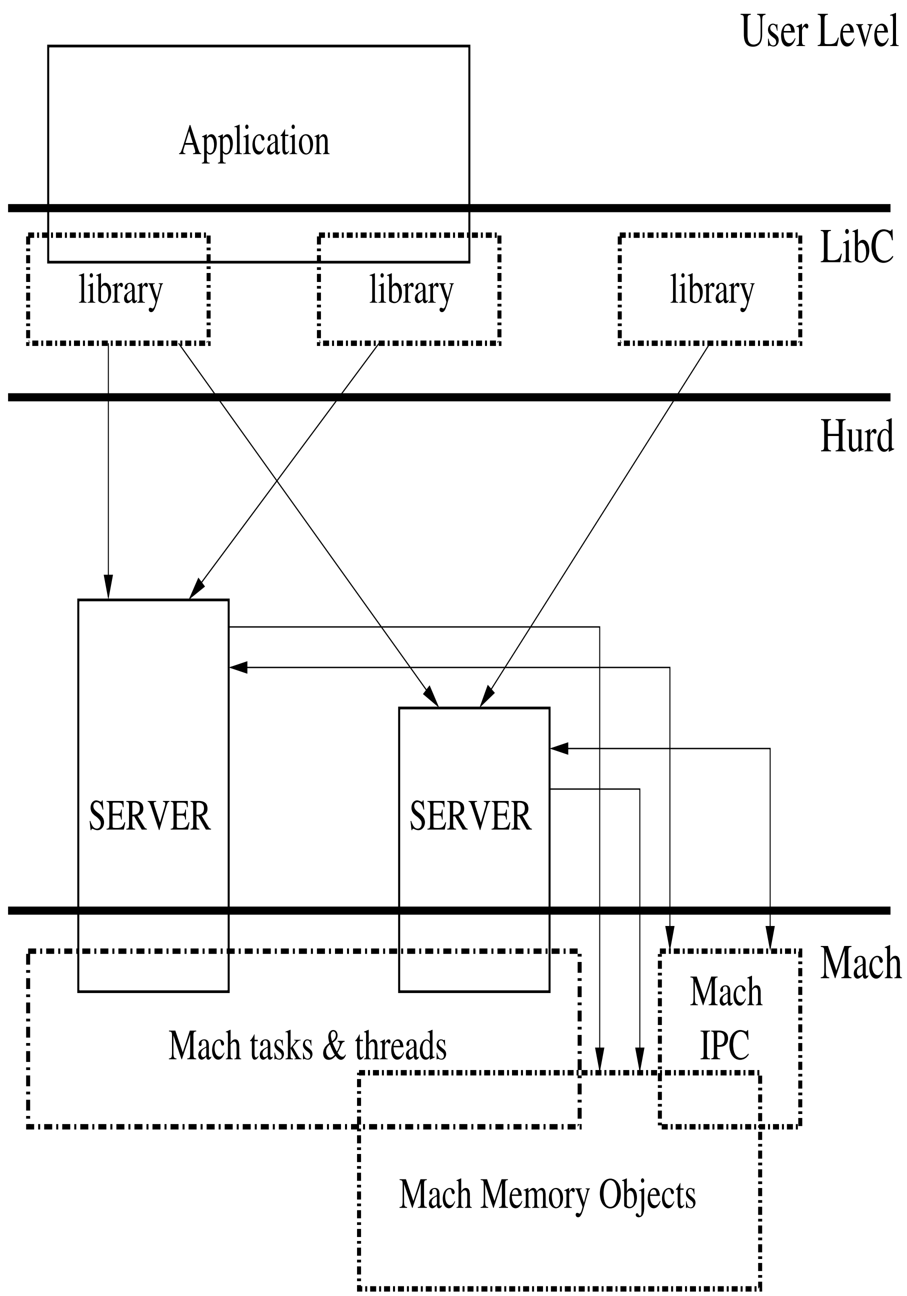
Τέλος για τη συγγραφή server και εφαρμογών χρήστη μαζί με το *Mach* έρχεται και η βιβλιοθήκη CThread που παρέχει μία υλοποίηση νημάτων σε επίπεδο χρήστη και λειτουργίες για τη διαχείριση και τον αμοιβαίο αποκλεισμό τους. Η βιβλιοθήκη αυτή εξετάζεται λεπτομερώς στο επόμενο κεφάλαιο. Περισσότερες πληροφορίες για το *Mach* μπορεί να βρεί κανείς στα εξής βιβλία: [13, 14, 15].

3.2 Εισαγωγή στο Hurd

Το *Hurd* είναι ένα σύνολο από server που χρησιμοποιεί τις δυνατότητες του *Mach* για να παρέχει τη λειτουργικότητα ενός POSIX συστήματος. Επιπλέον εισάγει καινούργιες δυνατότητες και μία εντελώς διαφορετική αρχιτεκτονική από τους παραδοσιακούς μονολιθικούς *UNIX* πυρήνες. Εδώ παρουσιάζεται ο τρόπος με τον οποίο οι server του *Hurd* υλοποιούν ένα λειτουργικό σύστημα.

Είδαμε ότι ένα *Mach* task διαθέτει αρκετή λειτουργικότητα ώστε να μπορεί να υλοποιήσει μία διεργασία. Οι server είναι εφαρμογές χρήστη και ισοδυναμούν με *Mach* task τα οποία εκτελούν μία συγκεκριμένη λειτουργία, ανεξάρτητα το ένα από το άλλο, και μαζί υλοποιούν ένα *UNIX* σύστημα. Η συνεργασία των server γίνεται δυνατή μέσω του συστήματος διαδιεργασιακής επικοινωνίας (IPC) του *Mach*. Η ολοκλήρωση μίας λειτουργίας μπορεί να απαιτεί την επικοινωνία αρκετών server (Σχήμα 3.1). Η μέθοδος επικοινωνίας περιλαμβάνει την αποστολή ενός μηνύματος από ένα server στο port κάποιου άλλου. Το πρόβλημα που εμφανίζεται είναι πως κάποιος server αποκτά δικαίωμα αποστολής στην port κάποιου άλλου με τον οποίο πρέπει να επικοινωνήσει ώστε να ολοκληρώσει τη λειτουργία. Η λύση σε αυτό το πρόβλημα βασίζεται μερικά σε ένα ειδικευμένο server που περιέχει τα ονόματα ευρέως χρησιμοποιούμενων server και στο ριζοσπαστικό τρόπο που χρησιμοποιεί το *Hurd* το σύστημα αρχείων. Κάθε server έχει δικαίωμα αποστολής στην port ενός συγκεκριμένου server-ευρετηρίου. Οι server κατά την έναρξή τους καταγράφουν τα στοιχεία τους, μία port για επικοινωνία και το όνομά τους, σε αυτό τον server-ευρετήριο, ενώ ανακτούν τα στοιχεία άλλων server εκτελώντας ερωτήσεις σε αυτόν.

Το σύστημα αρχείων του *Hurd* είναι το `ext2fs`, ένα ευρέως χρησιμοποιούμενο σύστημα αρχείων, που χρησιμοποιεί κόμβους (i-nodes) για να αποθηκεύσει τα αρχεία και πληροφορίες που τα αφορούν [11, 12]. Στο *Hurd* όμως έχει ένα



Σχήμα 3.1: Επικοινωνία τμημάτων του Hurd. Τα τμήματα χωρίζονται σε επίπεδα όπου στο τελευταίο βρίσκεται ο Mach, ύστερα ο Hurd και πιό πάνω πριν από τις κοινές εφαρμογές η βιβλιοθήκη της C. Διακεκομμένα πλαίσια δείχνουν τις υπηρεσίες κάθε επιπέδου. Συμπαγή πλαίσια τις εφαρμογές δηλαδή τους server και του χρήστη. Επικαλυπτόμενα πλαίσια δείχνουν επικοινωνία με μορφή κλήσης συναρτήσεων, ενώ τα βέλη υποδεικνύουν επικοινωνία μέσω ανταλλαγής μηνυμάτων.

πλεονέκτημα καθώς μέσα στους κόμβους του συστήματος αρχείων αποθηκεύονται πληροφορίες για τους ίδιους τους server του συστήματος.

Συγκεκριμένα κάθε κόμβος μπορεί να αναπαριστά ένα απλό αρχείο ή να περιέχει το όνομα ενός server που είναι υπεύθυνος για την διαχείριση ή αλλιώς μετάφραση των περιεχομένων του, γι'αυτό οι server λέγονται και translator. Για κάθε λειτουργία του συστήματος υπάρχει και ένα αντίστοιχο αρχείο στο σύστημα αρχείων,ενώ στον κόμβο του αρχείου αυτού υπάρχει αποθηκευμένο το όνομα του server που πρέπει να κληθεί με όποιες παραμέτρους χρειάζεται ο server για να λειτουργήσει σωστά. Για παράδειγμα για το TCP/IP υπάρχει το αρχείο `/servers/socket/inet` ενώ στον κόμβο του γράφεται το όνομα του server, `/hurd/pfinet` με παραμέτρους την IP διεύθυνση του μηχανήματος, τη μάσκα υποδικτύου κ.ά.

Η προσπέλαση του συστήματος αρχείων γίνεται μέσω ενός server που διαχειρίζεται την επικοινωνία με το *Mach* και συνεπώς το σκληρό δίσκο. Όταν προσπελαύνεται ένας κόμβος του συστήματος αρχείων ο server που διαχειρίζεται το σύστημα αρχείων, αν αυτό είναι κανονικό αρχείο επιστρέφει μία port που αντιπροσωπεύει το αρχείο, αλλιώς εκτελεί το server που αντιστοιχεί στον κόμβο και επιστρέφει μία port για επικοινωνία με αυτό το server. Τη διαδικασία αυτή ενεργοποιούν και διαχειρίζονται εξ ολοκλήρου οι κλήσεις συστήματος που υλοποιούνται από τη βιβλιοθήκη της C, έτσι ο προγραμματιστής δε χρειάζεται να γνωρίζει το παραμικρό για τις λεπτομέρειες αυτής της επικοινωνίας. Παρακάτω παραθέτουμε μερικούς από τους κυριότερους server του πυρήνα *Hurd* οι οποίοι εκτελούνται αυτόματα από το σύστημα.

- **ext2fs** - Ο server που αναλαμβάνει τη διαχείριση του συστήματος αρχείων.
- **exec** - Ο server που δημιουργεί μία νέα διεργασία από ένα εκτελέσιμο αρχείο.
- **proc** - Ο server που αναλαμβάνει θέματα διαχείρισης διεργασιών, π.χ. ανάθεση PID.
- **init** - Ο server που αρχικοποιεί το σύστημα.
- **mach-defpager** - Ο server που αναλαμβάνει τη διαχείριση μνήμης.

Φυσικά εξακολουθούν να ισχύουν οι άδειες πρόσβασης του κλασικού συστήματος αρχείων του *UNIX* παρέχοντας έτσι ελεγχόμενη πρόσβαση στους server.

Είναι φανερό ότι αυτό το σύστημα είναι πολύ ευέλικτο και έχει πολλά πλεονεκτήματα όσον αφορά την παραμετροποίησή του. Συγκεκριμένα επιτρέπει στους χρήστες να χρησιμοποιούν τους δικούς τους server παραμετροποιώντας ή και αντικαταστώντας αυτούς του συστήματος, μέσα στα πλαίσια των αδειών τους φυσικά. Ακόμα είναι προφανές ότι εκτός από μερικούς βασικούς server οι υπόλοιποι εκτελούνται αυτόματα από το σύστημα μόνο όταν χρειάζεται γλιτώνοντας έτσι πόρους. Αυτά τα χαρακτηριστικά πληρούν τις προϋποθέσεις που θέσαμε στο προηγούμενο κεφάλαιο, δηλάδή την επεκτασιμότητα και την ευελιξία, και καθιστούν το *Hurd* ένα πολύ ελκυστικό σύστημα για μελέτη, δικαιολογώντας την επιλογή μας. Στο επόμενο κεφάλαιο εξετάζουμε αναλυτικά τις αλλαγές που κάναμε στο *Hurd* προκειμένου να επιτύχουμε βελτίωση της απόδοσης του SmAS.

Κεφάλαιο 4

Χρονοπρογραμματισμός φίλτρων

Έχουμε ήδη επισημάνει ότι ένα μεγάλο κομμάτι του SmAS αφορά την μεταφορά ενός τμήματος της επεξεργασίας των δεδομένων από τον client στον εξυπηρέτη. Αυτή η μεταφορά υλοποιείται με τη δημιουργία απλών προγραμμάτων από το χρήστη, που τα ονομάζουμε φίλτρα, τα οποία εκτελούνται στο server και ελέγχουν την ροή των δεδομένων προς τον client. Επομένως τα φίλτρα αποτελούν ένα σημαντικό παράγοντα που επηρεάζει την απόδοση του server.

Ειδικότερα τα φίλτρα αποτελούν το σημαντικότερο παράγοντα καθορισμού του ρυθμού διαμεταγωγής δεδομένων που πετυχαίνει ο εξυπηρέτης στην περίπτωση που εξυπηρετεί πολλούς clients ταυτόχρονα. Ταυτόχρονη εξυπηρέτηση σημαίνει την ύπαρξη πολλών νημάτων στον εξυπηρέτη που το καθένα αναλαμβάνει τις αιτήσεις ενός client, συμπεριλαμβανομένης και της εκτέλεσης του φίλτρου.

Με την εκτέλεση πολλών φίλτρων ταυτόχρονα αυξάνεται η συνδρομικότητα (concurrency) στον εξυπηρέτη και εγείρονται θέματα όπως το χρονικό διάστημα που ένα φίλτρο θα χρησιμοποιεί τον επεξεργαστή και αν κάποιος client έχει προτεραιότητα έναντι κάποιου άλλου. Προκειμένου να διερευνήσουμε τα θέματα αυτά και να βελτιστοποιήσουμε την απόδοση του εξυπηρέτη σχεδιάσαμε και υλοποιήσαμε διαφορετικές πολιτικές χρονοπρογραμματισμού των φίλτρων.

Στη συνέχεια (Ενότητα 4.1) εξετάζουμε τις πολιτικές χρονοπρογραμματισμού που θα υλοποιήσουμε, έπειτα (Ενότητα 4.2) αναλύουμε το τρόπο υλοποίησης αυτών των πολιτικών και (Ενότητα 4.3) τις επιπτώσεις τους στην απόδοση του εξυπηρέτη.

4.1 Πολιτικές χρονοπρογραμματισμού

Προκειμένου να βελτιώσουμε τις επιδόσεις του εξυπηρέτη, όταν εξυπηρετούνται πολλοί client που εκτελούν ερωτήσεις μέσω των φίλτρων τους, πρέπει να εξετάσουμε τον τρόπο με τον οποίο τα φίλτρα αυτά χρησιμοποιούν τον επεξεργαστή. Προκειμένου να μελετήσουμε τη βελτιστοποιήση του εξυπηρέτη δοκιμάσαμε δύο πολιτικές χρονοπρογραμματισμού των νημάτων που συναντώνται αρκετά συχνά στη βιβλιογραφία των λειτουργικών συστημάτων [11, 12].

Roundrobin Ο αλγόριθμος αυτός εναλλάσσει ανά τακτικά χρονικά διαστήματα το νήμα που χρησιμοποιεί τον επεξεργαστή με ένα άλλο που βρίσκεται υπό αναστολή, δίνοντας έτσι ένα κβάντο χρόνου σε κάθε νήμα. Τα νήματα εναλλάσσονται το ένα μετά το άλλο, δηλαδή επαναπρογραμματίζεται το ίδιο νήμα μόνο αφού όλα τα υπόλοιπα έχουν χρησιμοποιήσει τον επεξεργαστή. Έτσι επιτυγχάνεται ομοιόμορφη κατανομή του επεξεργαστή στα νήματα.

First Come First Served Αυτός ο αλγόριθμος προγραμματίζει το νήμα που δημιουργείται πρώτο, συνεχώς και μέχρι αυτό να τελειώσει την εκτέλεσή του. Έτσι δεν υπάρχει παράλληλη εκτέλεση αλλά το κάθε νήμα περιμένει να τελειώσει το προηγούμενό του για να χρησιμοποιήσει τον επεξεργαστή.

Η πρώτη πολιτική αυξάνει τη συνδρομικότητα, αφού όταν ένα νήμα έχει μπλοκάρει (π.χ. περιμένοντας δεδομένα από το δίσκο) ένα άλλο μπορεί να εκτελείται, ενώ είναι δίκαιη για όλους τους client, αφού όλοι χρησιμοποιούν τον επεξεργαστή παράλληλα. Στην πραγματικότητα όμως η συνδρομικότητα δεν αυξάνεται πολύ αφού διαθέτουμε μόνο ένα επεξεργαστή για να εκτελεί τα νήματα. Επιπλέον η συνεχής διαδικασία της εναλλαγής των νημάτων, χρησιμοποιεί χρόνο του επεξεργαστή ελαττώνοντας το διαθέσιμο χρόνο για τα φίλτρα.

Η δεύτερη πολιτική ουσιαστικά καταργεί τη συνδρομικότητα αφού μόνο ένα φίλτρο εξυπηρετείται χωρίς διακοπή. Στο μονοεπεξεργαστικό μοντέλο γλιτώνουμε το χρόνο εναλλαγής των thread και αυτό θα περίμενε κανείς ότι είναι το βέλτιστο. Όμως ένα φίλτρο πολύ συχνά χρειάζεται να περιμένει την ανάγνωση δεδομένων από το δίσκο και την αποστολή τους στο δίκτυο. Έτσι πάλι υπάρχει χρόνος στον οποίο ο επεξεργαστής δε χρησιμοποιείται για την εκτέλεση φίλτρων. Επιπλέον ορισμένοι client θα αργήσουν πάρα πολύ να λάβουν τα πρώτα αποτελέσματα από τον εξυπηρέτη, αφού μπορεί να περιμένουν να τελειώσει κάποιος ή κάποιοι άλλοι που συνδέθηκαν πριν από αυτούς, χαρακτηριστικό όχι και τόσο επιθυμητό.

Ακόμη μία πολιτική που είναι ενδιαφέρον να εξεταστεί είναι η συντομότερη εργασία πρώτα (shortest job first - SJF). Στην πολιτική αυτή διαθέτουμε πληροφορία για το χρόνο που χρειάζεται κάθε εφαρμογή για να ολοκληρωθεί και επιλέγουμε την πιο σύντομη. Στην περίπτωσή μας κάθε φίλτρο έχει διαφορετικό χρόνο εκτέλεσης που εξαρτάται από το ίδιο το φίλτρο και από το αρχείο στο οποίο εφαρμόζεται. Το πρόβλημα είναι ότι δεν διαθέτουμε την απαραίτητη πληροφορία για το χρόνο εκτέλεσης των φίλτρων και πρέπει να τον μετρήσουμε δυναμικά κατά την εκτέλεσή τους.

Στόχος μας είναι η σύγκριση των πολιτικών Roundrobin και First Come - First Served που θα βοηθήσει στο να βελτιστοποιήσουμε την απόδοση του εξυπηρέτη. Στη συνέχεια αναλύουμε την υλοποίησή τους.

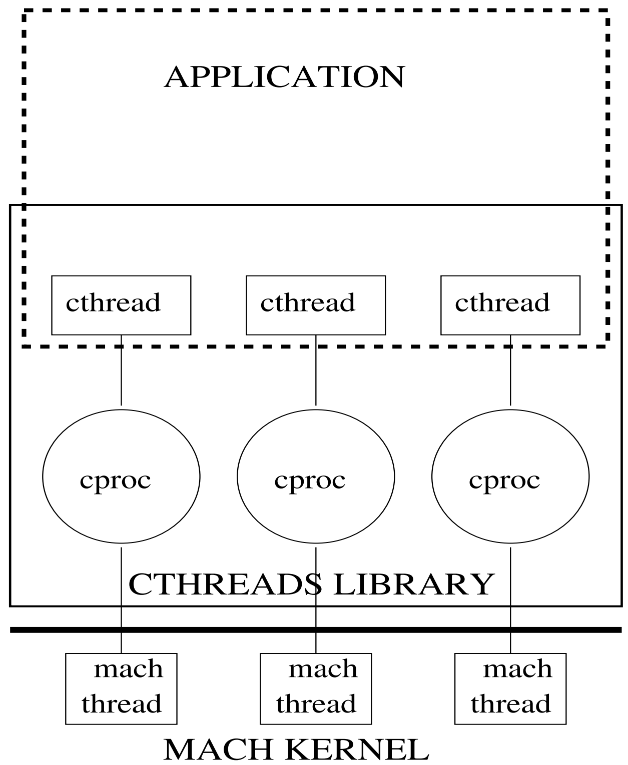
4.2 Υλοποίηση των πολιτικών χρονοπρογραμματισμού

Η υλοποίηση γίνεται με βάση τις παρεχόμενες δυνατότητες του *Hurd* και του *Mach* για αυτό το λόγο ξεκινάμε την ανάλυση της υλοποίησης από τη βιβλιοθήκη δημιουργίας νημάτων του *Hurd*, τη CThread [9].

Έχουμε ήδη αναφέρει τις δυνατότητες που παρέχει ο *Mach* για τη δημιουργία νημάτων αλλά στην υλοποίησή μας ασχολούμαστε με την βιβλιοθήκη CThreads. Η υλοποίηση με βάση τα *Mach* thread θα απαιτούσε αλλαγές στη δομή του μικροπυρήνα, ένα πολύ δύσκολο εγχείρημα αφού τα thread χρησιμοποιούνται σε κάθε λειτουργία του *Mach*, ενώ και τα διάφορα τμήματα του *Mach* δεν είναι ανεξάρτητα ώστε αλλαγές σε ένα να μην επηρεάζουν το άλλο. Εξάλλου αρχικά θέλουμε να χρονοπρογραμματίζουμε μόνο τα φίλτρα οπότε αντιμετωπίσαμε το *Mach* σαν ένα *"μαύρο κουτί"* και χρησιμοποιήσαμε τα cthreads ως βάση για τα δικά μας χρονοπρογραμματιζόμενα νήματα.

Αρχίζουμε την ανάλυση της υλοποίησης εξηγώντας τη λειτουργία της βιβλιοθήκης CThread. Η βιβλιοθήκη CThread παρέχει τη δυνατότητα δημιουργίας νημάτων στις εφαρμογές του *Hurd* και η τωρινή υλοποίηση της βασίζεται στην αρχική βιβλιοθήκη CThread που διδόταν σαν βοηθητικό εργαλείο μαζί με το *Mach* από το Carnegie Mellon University. Εκτός από τη δημιουργία των νημάτων παρέχεται και η δυνατότητα αμοιβαίου αποκλεισμού τους με την υλοποίηση αντικειμένων mutex και condition.

Τα cthread υλοποιούνται με τη βοήθεια των lightweight processes [12], των cproc. Στην πραγματικότητα ένα cthread είναι μία δομή με την οποία ο χρήστης αλληλεπιδρά με τις cproc η οποία περιέχει απλές πληροφορίες, όπως τη συνάρτηση που θέλει να εκτελέσει ο χρήστης. Οι cproc δημιουργούνται έτσι ώστε να περιέχουν ότι χρειάζεται ένα καινούργιο νήμα. Η δημιουργία μίας cproc αντιστοιχεί



Σχήμα 4.1: Σχέση cthread, cproc και *Mach* thread

στη δημιουργία ενός *Mach* thread και την εκχώρηση μίας νέας περιοχής μνήμης που χρησιμοποιείται σαν στοίβα από το νέο thread. Έπειτα η cproc αναλαμβάνει την εκτέλεση ενός cthread. Στον αρχικό κώδικα του CMU οι cproc μπορούσαν να διαχειρίζονται πολλαπλά cthread, στην υλοποίηση του *Hurd* όμως μία cproc αντιστοιχεί σε ένα cthread ενώ μετά την εκτέλεση και τερματισμό του cthread η cproc μπορεί να αναλάβει ένα νέο cthread. Επομένως για κάθε cthread υπάρχει μία cproc και για κάθε cproc ένα *Mach* thread (Σχήμα 4.1). Αναστολή της εκτέλεσης ενός cthread σημαίνει την αναστολή της cproc και του *Mach* thread που αντιστοιχούν σε αυτό.

Τα cthreads δεν χρησιμοποιούν κάποιο αλγόριθμο χρονοπρογραμματισμού. Για την εναλλαγή των νημάτων βασίζονται αποκλειστικά στο *Mach*. Ο χρονοπρογραμματιστής του *Mach* προσπαθεί να κρατά τον επεξεργαστή συνεχώς απασχολημένο γι'αυτό και όταν ένα *Mach* thread μπλοκάρει σε μία κλήση συστήματος, περιμένοντας δεδομένα από το δίσκο για παράδειγμα, εναλλάσσεται με όποιο άλλο *Mach* thread είναι έτοιμο να εκτελεστεί. Η εναλλαγή αυτή των *Mach* thread αντιστοιχεί όπως είπαμε σε εναλλαγή των cproc και cthread.

Όταν μία εφαρμογή χρησιμοποιεί τη βιβλιοθήκη CThread πρέπει αρχικά να καλέσει τη συνάρτηση `cthread_init()` ενώ έπειτα τα cthread δημιουργούνται και καταστρέφονται με τις `cthread_fork()` και `cthread_exit()`. Στην πραγματικότητα η κλήση της `cthread_init()` δεν είναι απαραίτητη στο *Hurd* αφού η βιβλιοθήκη αρχικοποιείται όταν αρχίζει να εκτελείται η διεργασία, με τη δημιουργία του κυρίως νήματος και ενός που αναλαμβάνει την διαχείριση των μηνυμάτων που στέλνονται στη διεργασία από το *Mach*.

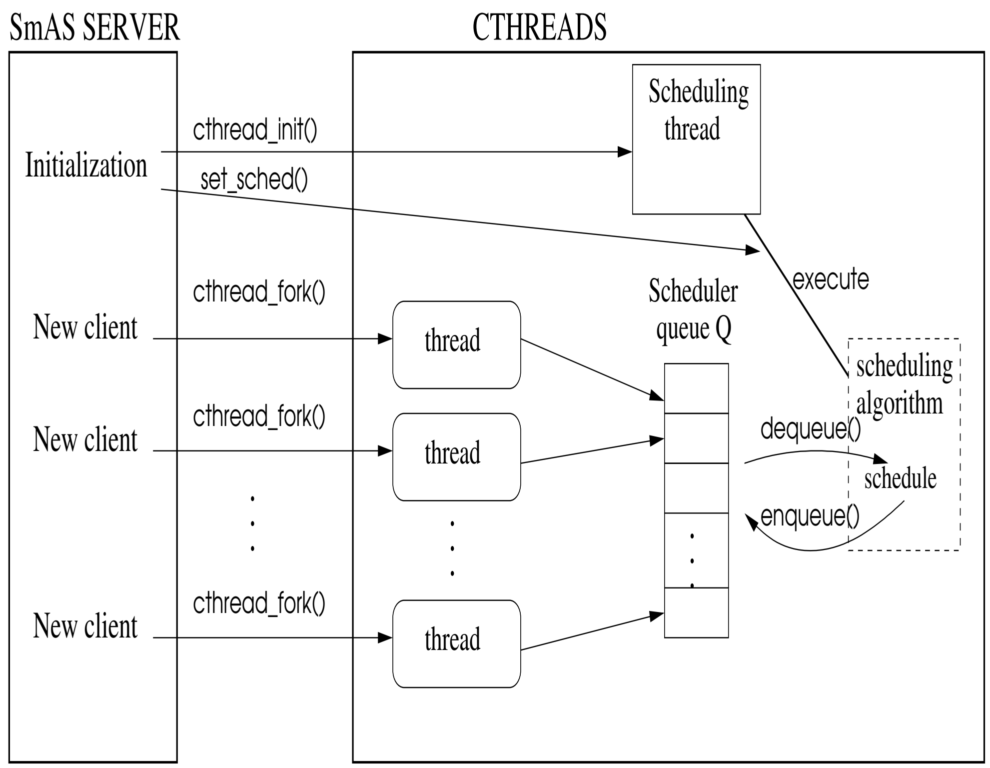
Κατά την δική μας υλοποίηση διατηρήθηκε η ένα προς ένα αντιστοιχία των cthread με τις cproc και των cproc με τα *Mach* thread. Η κλήση της `cthread_fork()` από το προγραμματιστή πλέον έχει σαν αποτέλεσμα τη δημιουργία μίας cproc η οποία βρίσκεται σε κατάσταση αναστολής και δεν χρονοπρογραμματίζεται. Η cproc αυτή εισάγεται σε μία ουρά Q η οποία περιέχει όλες τις cproc που πρέπει να εκτελεστούν. Κατά την αρχικοποίηση του προγράμματος όμως δημιουργούμε ένα επιπλέον νήμα το *scheduling thread* το οποίο εκτελείται καθ'όλη τη διάρκεια ζωής του server. Το *scheduling thread* εκτελεί έναν από τους αλγορίθμους που αναφέραμε παραπάνω. Ο προγραμματιστής επιλέγει τον αλγόριθμο με κλήση της `set_sched()`, η οποία πρέπει να γίνει πριν από τη δημιουργία άλλων νημάτων, η αρχιτεκτονική της υλοποίησης των νημάτων φαίνεται στο σχήμα 4.2.

Οι αλγόριθμοι χρονοπρογραμματισμού είναι πολύ απλοί. Ο αλγόριθμος round-robin αφαιρεί την κεφαλή της προαναφερθείσας ουράς Q των cproc και αναθέτει τον επεξεργαστή σε αυτή τη cproc για το ανάλογο κβάντο χρόνου, το οποίο καθορίζεται από τον προγραμματιστή όταν καλείται η `set_sched()`. Στο τέλος του κβάντου χρόνου ο αλγόριθμος χρονοπρογραμματισμού αναστέλλει την cproc που είχε τον επεξεργαστή και την εισάγει στο τέλος της ουράς, αν δεν έχει τερματίσει την εργασία της, ώστε να χρονοπρογραμματιστεί ξανά και επιλέγει την τρέχουσα κεφαλή της ουράς για εκτέλεση.

Αξίζει να σημειωθεί ότι ο αλγόριθμος που εκτελεί το *scheduling thread* μπορεί να δρομολογήσει ένα cthread που μπλοκάρει, για παράδειγμα λόγω ανάγνωσης δεδομένω από το δίσκο. Σε αυτή την περίπτωση ο *Mach* δρομολογεί ξανά το *scheduling thread* αφού είναι το μόνο διαθέσιμο προς εκτέλεση *Mach* thread. Έτσι δεν μπορούμε να βασιστούμε στο ότι ο *Mach* θα χρονοπρογραμματίσει ένα νήμα για ένα ολόκληρο κβάντο χρόνου.

Χρειάστηκε να ελέγχουμε το χρόνο που μεσολαβεί ανάμεσα στις εκτελέσεις του *scheduling thread* οπότε αν βρεθεί ότι έχει περάσει χρονικό διάστημα μικρότερο από ένα κβάντο χρόνου, χρονοπρογραμματίζουμε το ίδιο cthread. Έτσι διασφαλίσουμε ότι κάθε νήμα θα έχει στη διάθεσή του τον επεξεργαστή για ένα ολόκληρο κβάντο χρόνου.

Ο αλγόριθμος first come first served αφαιρεί την cproc που βρίσκεται στην



Σχήμα 4.2: Αρχιτεκτονική scheduler. Φαίνονται η δημιουργία του scheduling thread, η διαδικασία εισαγωγής νέων νημάτων και η εκτέλεση του αλγορίθμου χρονοπρογραμματισμού.

κεφαλή της προαναφερθείσας ουράς Q και αναθέτει τον επεξεργαστή σε αυτή τη cproc. Ανά τακτά χρονικά διαστήματα, ίσα με ένα κβάντο χρόνου όπως αυτό ορίζεται στο roundrobin, ο αλγόριθμος ελέγχει αν τερμάτισε η cproc οπότε και επιλέγει τη τρέχουσα κεφαλή της ουράς για εκτέλεση, αλλιώς μεταφέρει ξανά τον επεξεργαστή σε αυτή. Στην επόμενη παράγραφο εξετάζουμε τα αποτελέσματα αυτών των πολιτικών στην απόδοση του server.

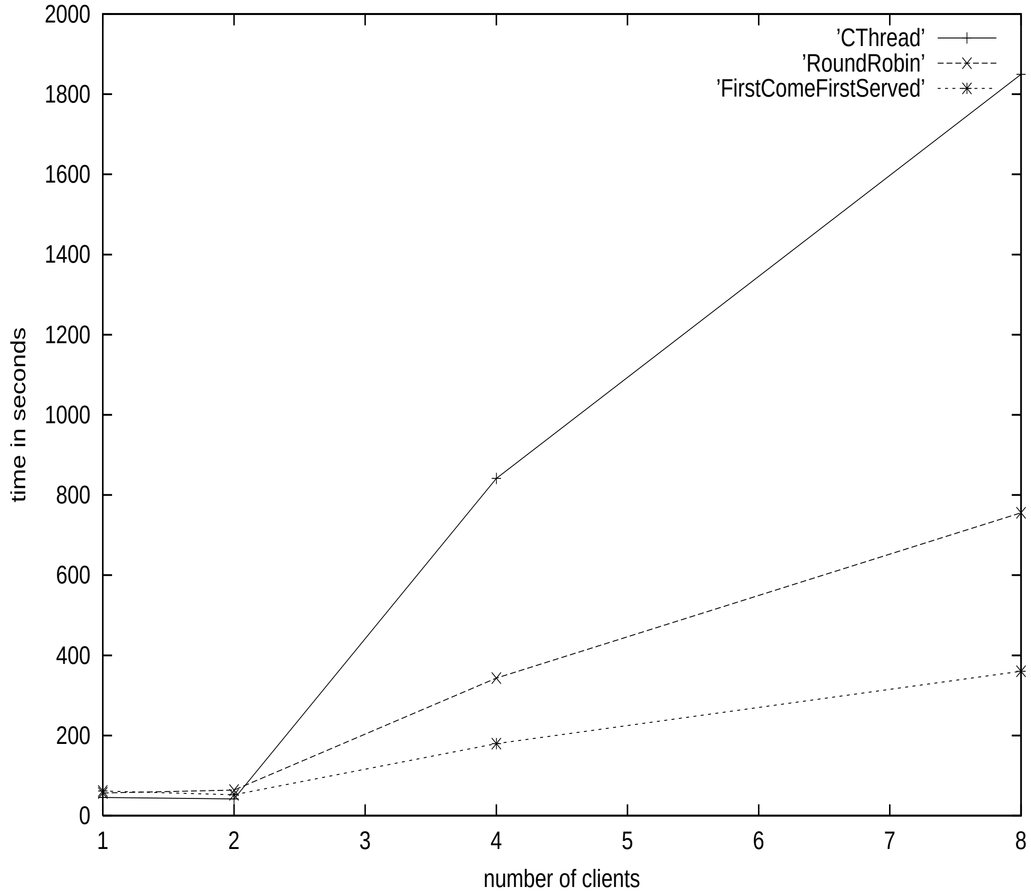
4.3 Πειραματικές μετρήσεις

Η υλοποίηση των δύο πολιτικών χρονοπρογραμματισμού έδωσε τη δυνατότητα της σύγκρισης της απόδοσης του server με χρήση καθεμίας από αυτές. Το πρόβλημα είναι πως θα μετρήσουμε την απόδοση του server από τη στιγμή που εμπλέκονται τόσοι παράγοντες, όπως ο χρόνος απόκρισης του client, ο χρόνος επεξεργασίας του φίλτρου και του client. Σαν κριτήριο για τη μέτρηση της απόδοσης του εξυπηρέτη επιλέξαμε το μέσο χρόνο που χρειάζεται ένας client για να ολοκληρώσει τη συναλλαγή του με τον εξυπηρέτη, που μας δίνει μία καλή εκτίμηση του ποσού δεδομένων που μπορεί να επεξεργαστεί και να αποστείλει ο εξυπηρέτης. Συγκεκριμένα μείωση του χρόνου εκτέλεσης σημαίνει αύξηση της απόδοσης, ενώ αύξηση του χρόνου εκτέλεσης σημαίνει μείωση της απόδοσης.

Τα μηχανήματα που χρησιμοποιήθηκαν για την εκτέλεση των πειραμάτων ήταν ένας `Pentium III` στα `733 MHz` με `128 MB RAM` και ένα μικρό σκληρό δίσκο των `512 MB` με λειτουργικό σύστημα *Hurd*, ο οποίος χρησιμοποιήθηκε σαν η SmAS συσκευή. Οι client έτρεχαν σε ένα μηχάνημα που έτρεχε *Linux*.

Οι client εκτελούσαν ένα απλό φίλτρο select, σαν αυτό του σχήματος 2.3, στο server ενώ δεν επεξεργάζοταν καθόλου το αποτέλεσμα ώστε να μειωθεί ο χρόνος επεξεργασίας στο client. Τα select εφαρμόστηκαν σε αρχεία μεγέθους `40 MB` ενώ κάθε εγγραφή που επιλέγοταν από το select, και αποστέλοταν στο client, είχε μέγεθος `10000 byte`. Οι εγγραφές ήταν της ίδιας μορφής με τη δομή `MyRecord` του παραδείγματος 2.2 με το πεδίο `key` να παίρνει ισοπίθανα τιμές από ένα εως εκατό.

Μετρήσαμε την απόδοση του server με 1, 2, 4 και 8 clients που εκτελούσαν select με διαφορετικό selectivity. Το selectivity ορίζει τον αριθμό των εγραφών που επιστρέφει ένα φίλτρο και συγκεκριμένα στο παράδειγμά μας τα φίλτρα select επιστρέφουν τις εγγραφές με πεδίο `key` μικρότερο ή ίσο του selectivity τους. Εκτός από τη σύγκριση των δικών μας αλγορίθμων συγκρίναμε και την εγγενή βιβλιοθήκη thread του *Hurd*, όπου όπως εξηγήσαμε δεν υπήρχε καθόλου χρονοπρογραμματισμός. Τα αποτελέσματα φαίνονται στο σχήμα 4.3. Στην πολιτική

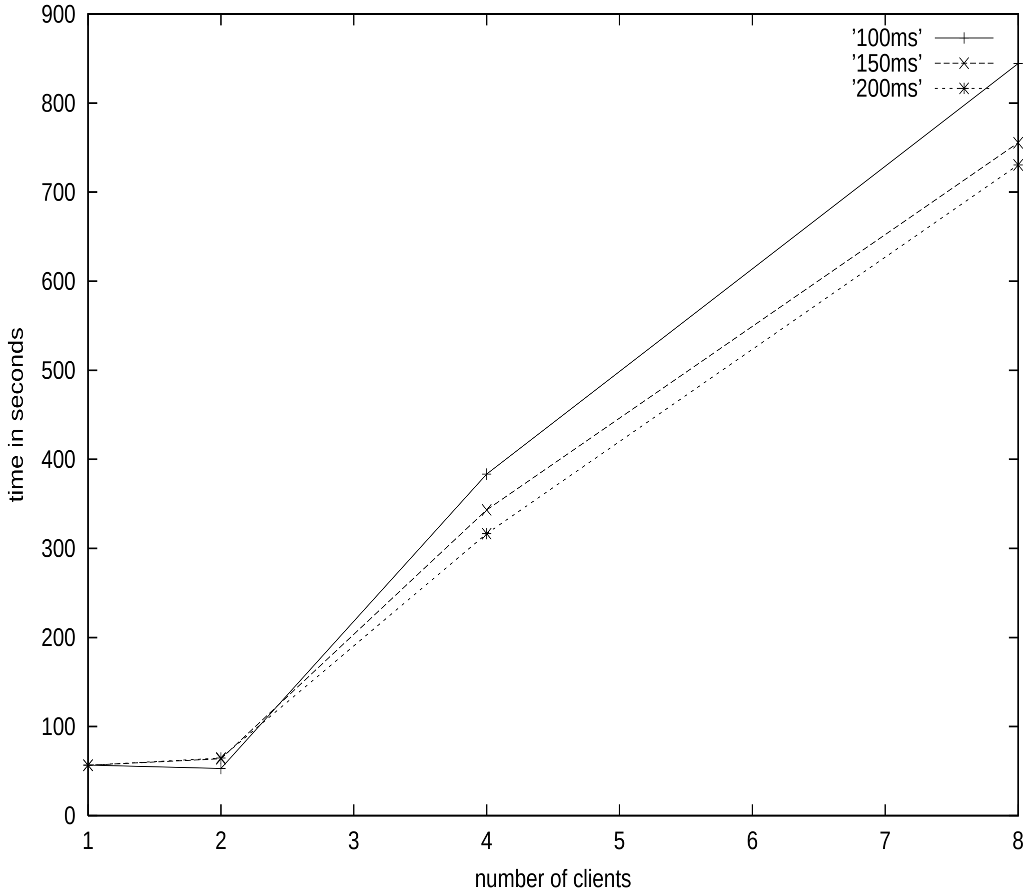


Σχήμα 4.3: Σύγκριση πολιτικών χρονοπρογραμματισμού. Στον οριζόντιο άξονα εμφανίζεται ο αριθμός των client σε κάθε πείραμα. Στον κατακόρυφο άξονα εμφανίζεται ο μέσος όρος των χρόνων εκτέλεσης των client για κάθε πείραμα, όπως προέκυψε από την εκτέλεση κάθε πειράματος 4 φορές

roundrobin το κβάντο χρόνου είναι `150 msec`.

Μπορούμε να παρατηρήσουμε ότι η βιβλιοθήκη CThread έχει την χειρότερη απόδοση ειδικά για πολλούς client. Αυτό μπορεί να εξηγηθεί αν θυμηθούμε τον τρόπο με τον οποίο γίνεται η δρομολόγηση των cthreads. Ο *Mach* εναλλάσσει συνεχώς τα thread οδηγώντας έτσι στην αύξηση της συνδρομικότητας. Όμως κάθε φίλτρο προσπελαύνει διαφορετικό αρχείο και στην περίπτωσή μας αυτό έχει σαν αποτέλεσμα την συχνή αλλαγή προσπελαυνόμενων αρχείων, που συνεπάγεται την αύξηση των αναζητήσεων της κεφαλής του δίσκου. Μία αναζήτηση μπορεί να διαρκέσει αρκετά msec, γεγονός που αυξάνει δραματικά το χρόνο ανάγνωσης δεδομένων.

Οι δικές μας πολιτικές εμφανίζονται πολύ γρηγορότερες. Ωστόσο υπάρχει μία



Σχήμα 4.4: Σύγκριση roundrobin με κβάντα χρόνου 100, 150 και 200 msec

μεγάλη διαφορά ανάμεσα στο roundrobin και τον first come first served αφού ο δεύτερος εμφανίζεται γρηγορότερος από τον πρώτο ειδικά για περισσότερους των δύο client. Αυτό μπορεί να εξηγηθεί από αυτό που επισημάναμε νωρίτερα: λόγω του ενός επεξεργαστή του συστήματος μας, η αύξηση του αριθμού των thread δεν συμβαδίζει με αύξηση της συνδρομικότητας. Επιπλέον, η ύπαρξη διαφορετικών client που επεξεργάζονται διαφορετικά αρχεία συνεπάγεται ότι ο σκληρός δίσκος πρέπει συνεχώς να εκτελεί μετακινήσεις της κεφαλής από αρχείο σε αρχείο που καθυστερεί αρκετά την ανάγνωση και αυξάνει την αναμονή των νημάτων. Αντίθετα, στο first come first served δεν έχουμε συνεχείς αναζητήσεις στο δίσκο και έτσι ο χρόνος που χάνουμε περιμένοντας την ανάγνωση των δεδομένων γίνεται σημαντικά μικρότερος. Παρόλα αυτά, ο first come first served δεν είναι ο καταλληλότερος αλγόριθμος αφού έχει μεγάλο χρόνο απόκρισης στην αρχική αίτηση του client.

Ειδικότερα για τον roundrobin εκτελέστηκαν τα ίδια πειράματα με διαφορετικό κβάντο χρόνου. Τα αποτελέσματα φαίνονται στο σχήμα 4.4. Παρατηρούμε ότι για μικρό κβάντο χρόνου (100 ms) η απόδοση αυξάνεται για λίγους client και μειώνεται για πολλούς. Αυτό συμβαίνει λόγω της αυξημένης συνδρομικότητας οπότε και γίνεται καλύτερη διαχείριση του επεξεργαστή. Για πολλούς client όμως η αυξημένη συνδρομικότητα έχει σαν αποτέλεσμα την αύξηση των αναζητήσεων της κεφαλής του δίσκου επιβραδύνοντας έτσι το χρόνο εκτέλεσης. Αντίθετα όσο μεγαλώνει το κβάντο χρόνου αυξάνει και η απόδοση όταν έχουμε πολλούς client ενώ μειώνεται ελάχιστα για λίγους client. Αυτό οφείλεται στη μείωση του αριθμού των αναζητήσεων που εκτελεί η κεφαλή του δίσκου, αφού λόγω του μεγάλου κβάντου χρόνου περισσότερες αναγνώσεις γίνονται από το ίδιο νήμα χωρίς να χρειάζεται μετακίνηση της κεφαλής του δίσκου.

Είδαμε ότι η απόδοση του server εξαρτάται πολύ από τον τρόπο που χρονοπρογραμματίζονται τα φίλτρα και ότι σημαντικό ρόλο παίζει ο σκληρός δίσκος που φαίνεται ότι είναι το πιο αργό μέρος του συστήματος. Το επόμενο βήμα για την βελτίωση της απόδοσης είναι η μείωση του χρόνου ανάγνωσης δεδομένων από το δίσκο, στόχος που μπορεί να επιτευχθεί με την υλοποίηση τεχνικών προανάκτησης. Με το θέμα αυτό θα ασχοληθούμε στη συνέχεια.

Κεφάλαιο 5

Προανάκτηση δεδομένων

Είδαμε στο προηγούμενο κεφάλαιο ότι σημαντικός παράγοντας περιορισμού του ρυθμού διαμεταγωγής δεδομένων από τον εξυπηρέτη στους clients είναι η καθυστέρηση που προκαλεί η ανάγνωση των δεδομένων από το δίσκο. Η προσπέλαση πολλών αρχείων ταυτόχρονα καταπονεί ιδιαίτερα το σκληρό δίσκο καθώς η κεφαλή χρειάζεται να μετακινείται συνεχώς εκτελώντας αναζητήσεις, όπου μία αναζήτηση μπορεί να διαρκέσει αρκετά msec. Σε αυτό το χρονικό διάστημα το νήμα μπλοκάρει χωρίς να παράγει ωφέλιμο έργο ο επεξεργαστής.

Ο "νεκρός" χρόνος κάθε νήματος μπορεί να μειωθεί αν χρησιμοποιηθούν τεχνικές προανάκτησης. Η προανάκτηση αφορά την προσπέλαση δεδομένων και αποθήκευση των αποτελεσμάτων πριν ακόμα αυτά χρειαστούν. Επομένως μία αίτηση μπορεί να εξυπηρετηθεί αμέσως από τα ήδη επεξεργασμένα και αποθηκευμένα αποτελέσματα. Στόχος μας είναι να μειώσουμε το χρόνο ανάγνωσης από το δίσκο με προανάκτηση τμήματος του αρχείου, ενώ με την προεπεξεργασία αυτού του μέρους των δεδομένων, δηλαδή την εφαρμογή φίλτρου πάνω τους, στον εξυπηρέτη πριν αυτά ζητηθούν από τον client στόχος μας είναι να μειώσουμε τον μέσο χρόνο απόκρισης για κάθε client.

Στη συνέχεια εξετάζουμε τη μέθοδο που θα ακολουθήσουμε για την υλοποίηση της προανάκτησης (Ενότητα 5.1), έπειτα περιγράφουμε το τρόπο υλοποίησης του (Ενότητα 5.2) και αναλύουμε τα αποτελέσματα στις επιδόσεις του εξυπηρέτη (Ενότητα 5.3).

5.1 Μελέτη υλοποίησης prefetching

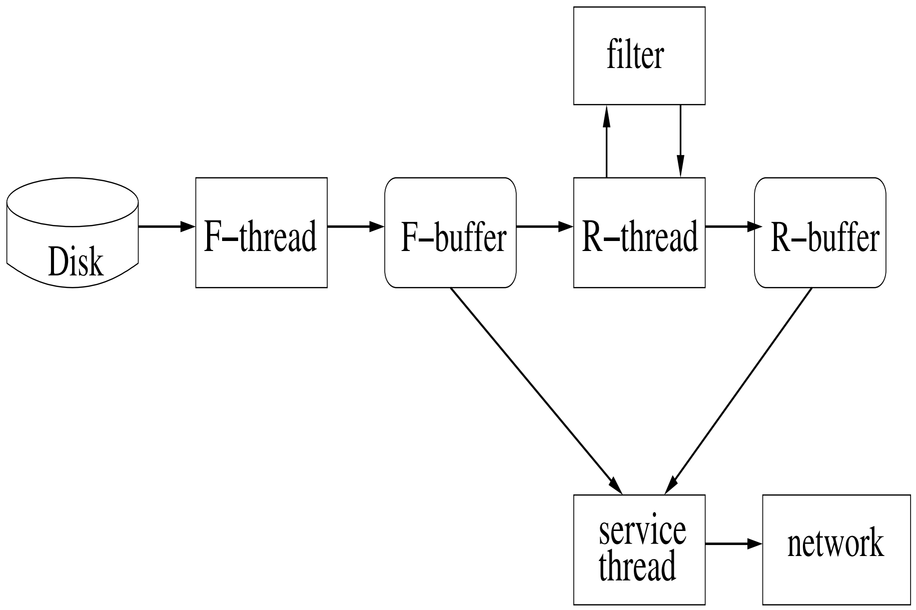
Περιγράφουμε τη μέθοδο που θα ακολουθήσουμε προκειμένου να βελτιώσουμε τις επιδώσεις του εξυπηρέτη. Είδαμε ότι σημαντικός παράγοντας καθυστέρησης είναι οι αναζητήσεις που εκτελεί η κεφαλή του δίσκου. Προκειμένου να ελαττώσουμε το χρόνο αυτό μπορούμε να χρησιμοποιήσουμε ένα buffer στον οποίο θα αποθηκεύουμε τα επόμενα N απροσπέλαστα byte του αρχείου, όπου N το μέγεθος του buffer. Ονομάζουμε αυτόν τον buffer file buffer ή F-buffer. Η επόμενη ανάγνωση του αρχείου γίνεται μέσα από τα δεδομένα του F-buffer και διαρκεί πολύ λιγότερο χρόνο. Η εισαγωγή των δεδομένων στο F-buffer πρέπει να γίνεται παράλληλα με την εξαγωγή τους ώστε να μην αδειάζει κατά το δυνατό, αφού κενός F-buffer σημαίνει ότι η επεξεργασία δεν μπορεί να προχωρήσει μέχρι να εισαχθούν σε αυτόν νέα δεδομένα. Αυτό μπορεί να επιτευχθεί με τη χρήση δύο ανεξάρτητων νημάτων: ενός για ανάγνωση από το δίσκο και ενός για την επεξεργασία.

Ένα θέμα που προκύπτει είναι η συνέπεια του F-buffer σε σχέση με τα δεδομένα στο σκληρό δίσκο. Συγκεκριμένα μία εγγραφή ή αναζήτηση στο αρχείο αφήνει το F-buffer σε μία ασυνεπή κατάσταση αφού τα δεδομένα που περιέχει δεν αντιστοιχούν σε αυτά που θα λαμβάναμε με μία κανονική ανάγνωση από το αρχείο. Σ'αυτή την περίπτωση πρέπει να ακυρωθούν τα περιεχόμενα του F-buffer και να εισαχθούν τα νέα δεδομένα από το αρχείο.

Επιπλέον μπορούμε να μειώσουμε ακόμα περισσότερο το χρόνο απόκρισης του εξυπηρέτη στην αίτηση ενός client εξουδετερώνοντας το χρόνο αναμονής εξαιτίας της επεξεργασίας των δεδομένων από ένα φίλτρο. Εφαρμόζουμε το φίλτρο στα δεδομένα πριν λάβουμε αιτήσεις next_rec και αποθηκεύουμε τα αποτελέσματα σε έναν δεύτερο buffer, τον οποίο ονομάζουμε record buffer ή R-buffer. Οι επόμενες αιτήσεις next_rec μπορούν να εξυπηρετηθούν κατευθείαν από τα περιεχόμενα του R-buffer. Φυσικά ο R-buffer πρέπει πάντα να διαθέτει δεδομένα και αυτό γίνεται με τη χρήση ενός νήματος που εφαρμόζει το φίλτρο και γεμίζει το R-buffer και ενός νήματος που εξυπηρετεί αιτήσεις next_rec και αδειάζει το R-buffer. Παρόμοια με τον F-buffer και εδώ μία ανάγνωση, εγγραφή ή αναζήτηση στο αρχείο αφήνει το R-buffer σε μία ασυνεπή κατάσταση αφού τα δεδομένα που περιέχει δεν αντιστοιχούν σε αυτά που θα λαβαίναμε από μία κανονική next_rec. Ομοίως πρέπει να ακυρωθούν τα περιεχόμενα και να αρχίσει πάλι η εφαρμογή του φίλτρου στα νέα δεδομένα.

Βέλτιστη επίδοση στις εφαρμογές που χρησιμοποιούν αποκλειστικά την έξοδο φίλτρου επιτυγχάνουμε με το συνδυασμό των δύο buffers, δηλαδή έχοντας ένα νήμα το R-thread, που γεμίζει το R-buffer με τα δεδομένα του φίλτρου και έχει σαν είσοδο τον F-buffer που περιέχει τα δεδομένα του αρχείου, ο οποίος γεμίζει από ένα άλλο νήμα το F-thread που προσπελαύνει το αρχείο. Οι αιτήσεις των client εξυπηρετούνται από ένα τρίτο νήμα το service thread. Η νέα αρχιτεκτονική φαίνεται στο σχήμα 5.1.

Βέβαια χρειάζεται ιδιαίτερη προσοχή στην υλοποίηση καθώς ο αμοιβαίος απο-



Σχήμα 5.1: Ροή δεδομένων από το δίσκο στο δίκτυο μέσω των δύο buffers και των τριών νημάτων

κλεισμός των νημάτων απαιτεί αρκετά πολύπλοκους χειρισμούς, ενώ και η διατήρηση των buffers σε συνεπή κατάσταση δεν είναι καθόλου εύκολη υπόθεση. Οι λεπτομέρειες της υλοποίησης αυτής της μεθόδου αποτελούν το θέμα της επόμενης παραγράφου.

5.2 Υλοποίηση προανάκτησης

Η υλοποίηση της προανάκτησης στον εξυπηρέτη SmAS έγινε με την υλοποίηση μιας νέας δομής, του buffer, ενώ για την δημιουργία των νημάτων που ενεργούν στους buffer διακρίθηκαν και δύο περιπτώσεις:

- η περίπτωση που χρησιμοποιείται η βιβλιοθήκη CThread του *Mach*
- η περίπτωση που χρησιμοποιείται η δικιά μας παραλλαγή των CThread

Αρχικά εξετάζουμε το, ανεξάρτητο από βιβλιοθήκη νημάτων, τμήμα της υλοποίησης των buffer. Ορίσαμε δύο νέες δομές δεδομένων τους file buffers και τους record buffers καθώς και έναν αριθμό από λειτουργίες πάνω σε αυτούς οι κυριότερες από τις οποίες είναι :

- `buf_read` για ανάγνωση των δεδομένων του buffer

- `buf_validate` για επικύρωση των αναγνώσεων που έγιναν από το buffer
- `buf_invalidate` για ακύρωση των αναγνώσεων και των δεδομένων του buffer

Οι λειτουργίες αυτές είναι εσωτερικές με την έννοια ότι ενεργοποιούνται, από τα F-thread, R-thread και το service thread, ως αντίδραση στις εξωτερικές αιτήσεις του client που είναι `read (R)`, `write (W)`, `lseek (S)` και `next_rec (NEXT_REC)`. Οι F-buffers χρησιμοποιούνται για την αποθήκευση ενός αριθμού byte από το αρχείο στη μνήμη. Εκτός από τα δεδομένα περιέχουν πληροφορίες για την εξασφάλιση της συνέπειάς των περιεχομένων τους σε σχέση με τα περιεχόμενα των αρχείων. Συγκεκριμένα ο F-buffer αποθηκεύει πληροφορία για την κανονική θέση του *δείκτη ανάγνωσης του αρχείου*, δηλαδή τη θέση του αρχείου από την οποία πρέπει να ξεκινήσει μία κανονική ανάγνωση, ενώ ένας ακόμη *δείκτης αναγνωσμένων δεδομένων* δείχνει σε ποία θέση του αρχείου θα έπρεπε να βρίσκεται ο δείκτης ανάγνωσης αρχείου αν οι αναγνώσεις που έγιναν από το F-buffer ήταν κανονικές. Όταν οι δύο αυτοί δείκτες δείχνουν στην ίδια θέση τότε ο F-buffer βρίσκεται σε έγκυρη κατάσταση, δηλαδή μπορούν να εφαρμοστούν αμέσως στο αρχείο όλες οι λειτουργίες ανάγνωση, εγγραφή ή αναζήτηση. Ακόμα περιέχει πληροφορία για τη θέση στο αρχείο από την οποία διαβάζονται τα δεδομένα που αποθηκεύονται στο F-buffer, το μέγεθος της αποθηκευμένης πληροφορίας στο F-buffer καθώς και αντικείμενα mutex για την ασφαλή πρόσβαση στο F-buffer.

Ένας F-buffer διατηρεί τη συνέπεια με τα δεδομένα του αρχείου ως εξής: με κάθε ανάγνωση δεδομένων από το F-buffer (`buf_read`) αυξάνεται ο δείκτης αναγνωσμένων δεδομένων αλλά όχι και ο δείκτης ανάγνωσης αρχείου. Ο δείκτης ανάγνωσης αρχείου γίνεται ίσος με τον δείκτη αναγνωσμένων δεδομένων όταν ο F-buffer επικυρώνεται (`buf_validate`) οπότε και βρίσκεται σε έγκυρη κατάσταση. Αν γίνει μία αίτηση για εγγραφή :

1. ο F-buffer αδειάζει
2. τα δεδομένα γράφονται στη θέση του δείκτη ανάγνωσης αρχείου
3. ο δείκτης ανάγνωσης αρχείου αυξάνεται και δείχνει στο τέλος αυτών των δεδομένων
4. ο δείκτης αναγνωσμένων δεδομένων γίνεται ίσος με το δείκτη ανάγνωσης αρχείου
5. νέα δεδομένα εισάγονται στο F-buffer από τη θέση του δείκτη ανάγνωσης αρχείου

Αν γίνει μία αίτηση για ανάγνωση από το χρήστη και ο buffer βρίσκεται σε μη συνεπή κατάσταση:

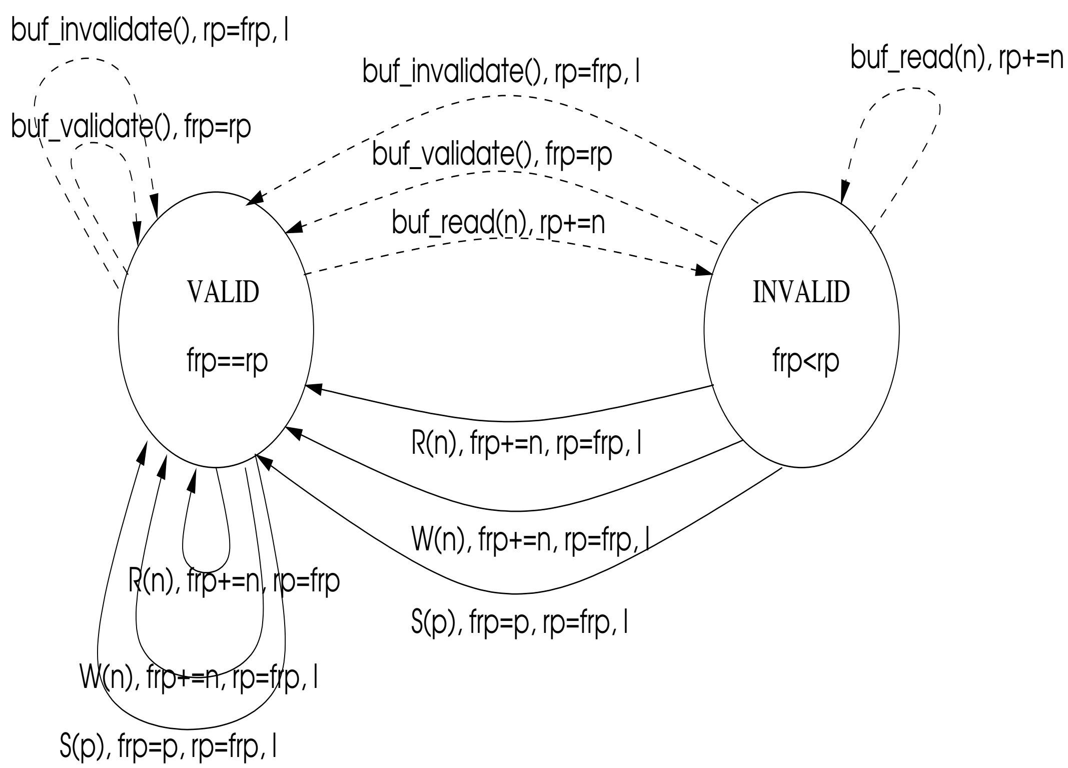
1. ο F-buffer αδειάζει
2. τα δεδομένα διαβάζονται από τη θέση του δείκτη ανάγνωσης αρχείου
3. ο δείκτης ανάγνωσης αρχείου αυξάνεται και δείχνει στο τέλος αυτών των δεδομένων
4. ο δείκτης αναγνωσμένων δεδομένων γίνεται ίσος με το δείκτη ανάγνωσης αρχείου
5. νέα δεδομένα εισάγονται στο F-buffer από τη θέση του δείκτη ανάγνωσης αρχείου

αν όμως βρίσκεται σε συνεπή κατάσταση :

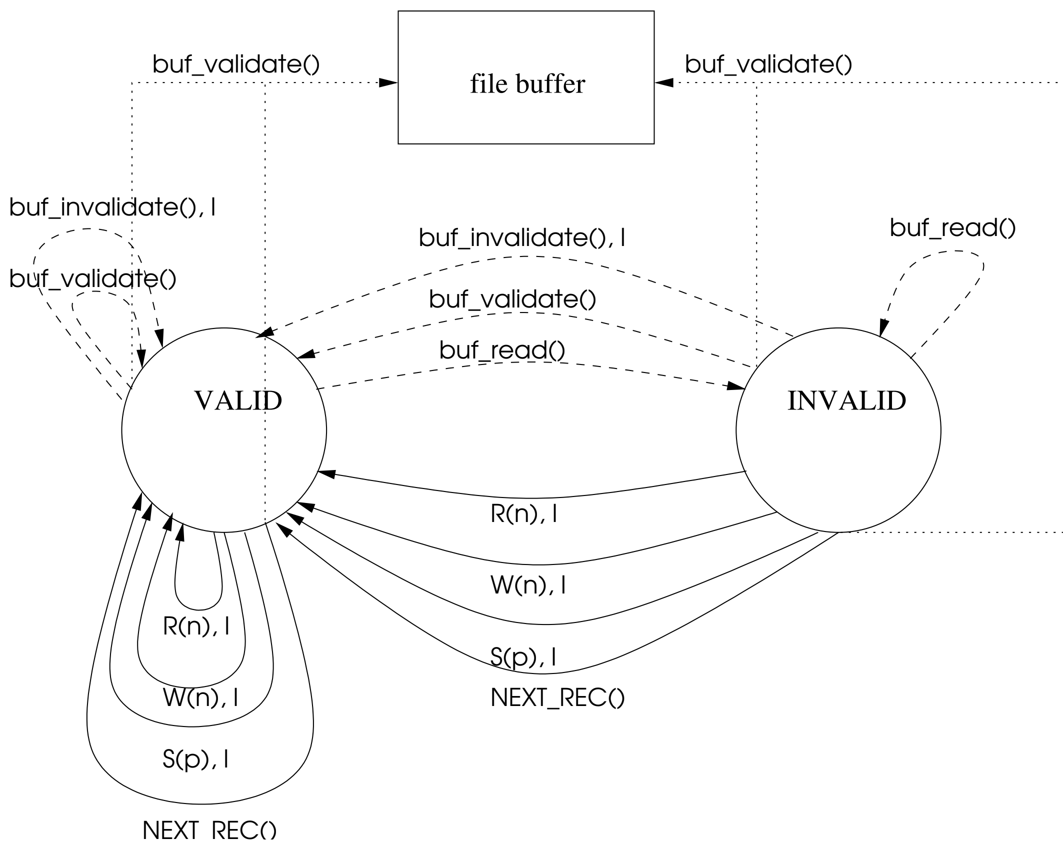
1. τα δεδομένα διαβάζονται από το F-buffer
2. ο δείκτης ανάγνωσης αρχείου αυξάνεται και δείχνει στο τέλος αυτών των δεδομένων
3. ο δείκτης αναγνωσμένων δεδομένων γίνεται ίσος με το δείκτη ανάγνωσης αρχείου
4. νέα δεδομένα εισάγονται στο F-buffer από τη θέση που σταμάτησε η προηγούμενη εισαγωγή

Αν γίνει μία αίτηση για αναζήτηση:

1. ο F-buffer αδειάζει
2. ο δείκτης ανάγνωσης αρχείου δείχνει στη νέα θέση
3. ο δείκτης αναγνωσμένων δεδομένων γίνεται ίσος με το δείκτη ανάγνωσης αρχείου
4. νέα δεδομένα εισάγονται στο F-buffer από τη θέση του δείκτη ανάγνωσης αρχείου



Σχήμα 5.2: Καταστάσεις F-buffer. Με διακεκομμένες γραμμές φαίνονται οι μεταβάσεις που προκαλούν εσωτερικές ενέργειες πάνω στο buffer , το δείκτη ανάγνωσης (frp) και το δείκτη αναγνωσμένων δεδομένων (rp). Ομοίως και για τις συνεχείς γραμμές μόνο που αντιστοιχούν σε μεταβάσεις λόγω αιτήσεων χρήστη με W(n), R(n) και S(p) να αντιστοιχούν σε εγγραφή n bytes, ανάγνωση n bytes και αναζήτηση στη θέση p. Το σύμβολο I δηλώνει ότι τα περιεχόμενα του buffer δεν χρησιμοποιούνται και απορρίπτονται



Σχήμα 5.3: Καταστάσεις R-buffer.

Οι καταστάσεις του F-buffer και οι μεταβάσεις τους φαίνονται στο σχήμα 5.2.

Ένας R-buffer είναι πιο απλός αφού ο συγχρονισμός με τα δεδομένα του αρχείου γίνεται κυρίως από το F-buffer. Ο R-buffer περιέχει ένα διάνυσμα από εγγραφές και μετρητές για το μέγεθος και τον αριθμό των record που επιστρέφει το φίλτρο και αποθηκεύονται στο buffer, και mutex για την ασφαλή πρόσβαση σε αυτόν. Ο R-buffer έχει πιο απλή διαδοχή καταστάσεων καθώς με κάθε αίτηση ανάγνωσης, εγγραφής ή αναζήτησης ακυρώνονται τα περιεχόμενά του ενώ κάθε αίτηση next_rec επικυρώνει την ανάγνωση από αυτόν και αυτός επικυρώνει τον F-buffer από τον οποίο διαβάζει. Στο σχήμα 5.3 φαίνονται οι καταστάσεις ενός R-buffer και αλλαγές τους καθώς και οι αλλαγές καταστάσεων που προκαλούνται στο αντίστοιχο F-buffer από τον οποίο διαβάζει ο R-buffer.

Η εισαγωγή νέων δεδομένων στους buffer πρέπει να γίνεται από ένα ξεχωριστό νήμα έτσι ώστε αυτή η διαδικασία να γίνεται παράλληλα με την ανάγνωση δεδομένων από τους buffer. Θυμίζουμε ότι αρχικά είχαμε ένα νήμα που εξυπηρετούσε τις αιτήσεις του client, διάβαζε από το δίσκο και εφάρμοζε τα φίλτρα.

Τώρα με την προσθήκη των buffer προσθέσαμε για κάθε buffer και από ένα νήμα επιφορτισμένο με το καθήκον να γεμίζει τον buffer. Αυτά τα νήματα τα έχουμε ήδη αναφέρει σαν F-thread και R-thread αλλά θα αναφερόμαστε σε αυτά και σαν fill thread. Εξακολουθεί να υπάρχει και ένα νήμα που εξυπηρετεί τις αιτήσεις του client, το οποίο ονομάζουμε service thread. Όλες οι λειτουργίες των buffer που περιγράψαμε παραπάνω γίνονται από αυτά τα νήματα.

Κατά τη σύνδεση ενός νέου client δημιουργείται το service thread, ενώ όταν ο client ανοίγει ένα αρχείο δημιουργείται ο file buffer και το αντίστοιχο fill thread του. Όταν εφαρμόζεται ένα φίλτρο πάνω στο αρχείο τότε έχουμε τη δημιουργία του R-buffer και του δικού του fill thread. Ένα F-thread εκτελεί συνεχώς αναγνώσεις από το δίσκο προσπαθώντας να γεμίσει το F-buffer του. Ένα R-thread εκτελεί συνεχώς αναγνώσεις από το F-buffer, εφαρμόζει το φίλτρο στα δεδομένα και αποθηκεύει το αποτέλεσμα στο δικό του R-buffer ώσπου να γεμίσει. Αυτό το νήμα ευθύνεται για λειτουργίες `buf_read, buf_validate` και `buf_invalidate` στο F-buffer. To service thread περιμένει αιτήσεις του client και τις εξυπηρετεί διαβάζοντας όποτε χρειάζεται τους buffer. Το νήμα αυτό ευθύνεται για λειτουργίες `buf_read, buf_validate, buf_invalidate` και αλλαγές κατάστασης λόγω αιτήσεων `R, W, S, NEXT_REC` στο R-buffer και για `buf_read, buf_validate` και αλλαγές κατάστασης λόγω αιτήσεων `R, W, S` στο F-buffer.

Στην περίπτωση που χρησιμοποιούνται τα cthreads η δημιουργία των fill thread αντιστοιχεί απλά στη δημιουργία δύο νέων cthread. Η πρόσβαση στους buffer προστατεύεται με το κλείδωμα αντικειμένων mutex. Μία ειδική περίπτωση εμφανίζεται όταν κάποιο νήμα που διαβάζει από κάποιο buffer τον βρει άδειο. Σε αυτή την περίπτωση η `buf_read` μπλοκάρει το νήμα με τη χρήση ενός αντικειμένου condition.

Υπενθυμίζουμε ότι τα condition είναι αντικείμενα στα οποία ένα νήμα μπλοκάρει με μία λειτουργία `condition_wait()` περιμένοντας για κάποια συνθήκη. Ένα άλλο νήμα μπορεί να εκτελέσει μία λειτουργία `condition_signal()` όταν η συνθήκη γίνει αληθής, "ξυπνώντας" το μπλοκαρισμένο, σε αυτό το condition, thread. Τα condition παρέχονται από τη βιβλιοθήκη CThreads.

Έτσι τα fill thread εκτελούν `condition_signal()` στα condition των buffer τους, "ξυπνώντας" τα νήματα που μπορεί να περιμένουν για δεδομένα. Βέβαια η εκτέλεση του `condition_signal()` δε συνεπάγεται και τον άμεσο χρονοπρογραμματισμό του "κοιμισμένου" νήματος οπότε μπορεί να υπάρχουν αρκετές καθυστερήσεις. Σε μία προσπάθεια να μειωθούν αυτές οι καθυστερήσεις μετά το `condition_signal()` τα fill thread εκτελούν `cthread_yield()`, λειτουργία των cthread που εναλλάσσει το cthread που εκτελεί ο επεξεργαστής με ένα άλλο, ώστε να επιταχυνθεί ο χρονοπρογραμματισμός του επόμενου νήματος. Συγκεκριμένα το fill thread του F-buffer εκτελεί το `cthread_yield()` μόνο όταν είναι τελείως γεμάτος, αφού αλλιώς θα είχαμε την ίδια περίπτωση χωρίς buffer, δηλαδή διαδοχικές αναγνώσεις από το δίσκο. Το fill thread του R-buffer εκτελεί το `cthread_yield()` συνεχώς αφού αν δεν υπάρχει αίτηση next_rec και ο F-buffer είναι γεμάτος τότε ξαναεκτελείται.

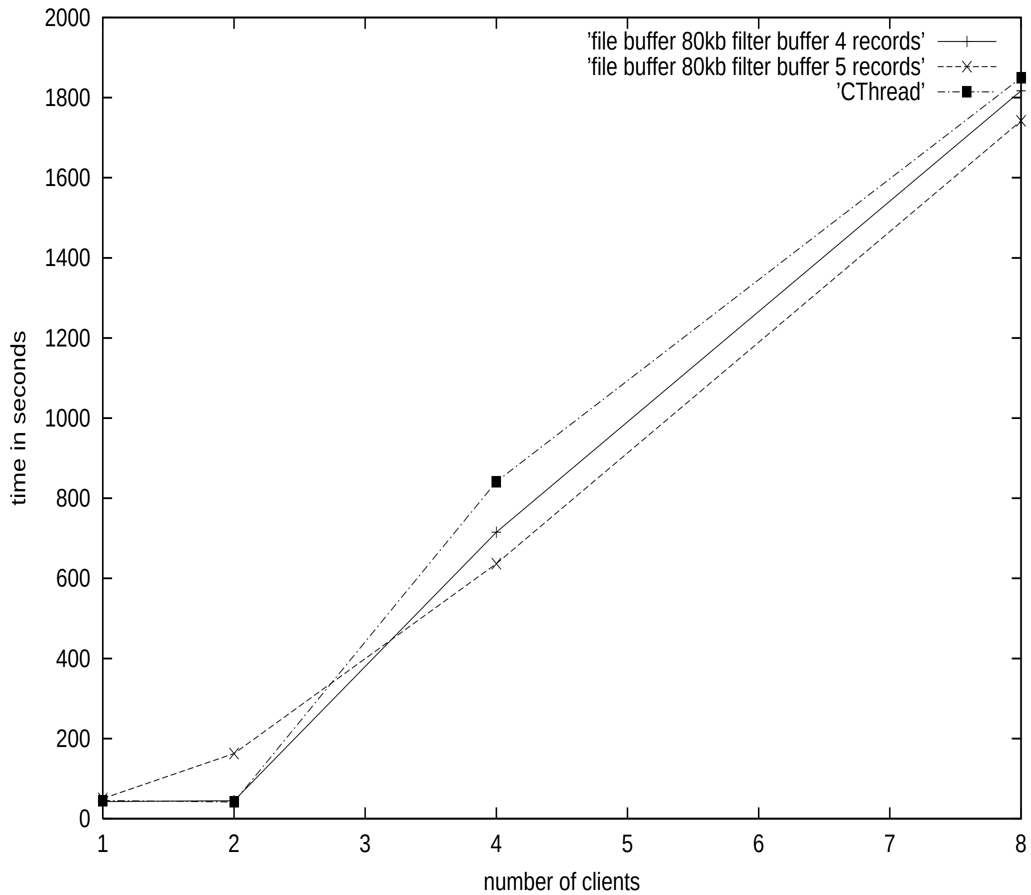
Στην δική μας υλοποίηση των cthread έχει αλλάξει λίγο ο τρόπος που δημιουργούνται τα νήματα. Είδαμε ότι ο χρονοπρογραμματισμός γίνεται με βάση τους client και αφού μέχρι τώρα σε κάθε client αντιστοιχούσε ένα νήμα χρονοπρογραμματίζαμε αυτό το νήμα. Τώρα έχουμε τρία νήματα ανά client οπότε και ο χρονοπρογραμματισμός πρέπει να εφαρμόζεται και στα τρία αυτά νήματα. Αυτό επιτυγχάνεται με τη δημιουργία *ομάδων νημάτων*. Κάθε ομάδα δημιουργείται σαν μία λίστα με κεφαλή το service thread, οι δεσμοί της λίστας φυλάσσονται στη δομή cproc κάθε νήμα όπου και περιέχεται ο δεσμός προς το επόμενο νήμα. Τα fill thread πλέον δημιουργούνται με την `cthread_spawn()` που είναι αντίστοιχη με τη `cthread_fork()` αλλά τα δημιουργούμενα νήματα εισάγονται στο τέλος της λίστας που ορίζει το νήμα που τη `cthread_spawn()`. Το νήμα αυτό είναι πάντα το service thread οπότε τα δύο fill thread και το service thread ανήκουν πάντα στην ίδια ομάδα.

Οι αλγόριθμοι χρονοπρογραμματισμού πλέον εξάγουν μία cproc, από την ουρά των cproc που πρέπει να εκτελεστούν, η οποία είναι ένα service thread, τα fill thread δεν εισάγονται σε αυτή την ουρά από τη `cthread_spawn()`. Ο χρονοπρογραμματισμός γίνεται για τα νήματα όλης της ομάδας, δηλαδή χρονοπρογραμματίζονται και τα fill thread και το service thread ή αναστέλλονται όλα μαζί. Ο περαιτέρω χρονοπρογραμματισμός ανάμεσα στα τρία αυτά νήματα αφήνεται στο *Mach*. Εξαίρεση αποτελούν οι κλήσεις στη `cthread_yield()` που έχουν πλέον αντικατασταθεί από κλήσεις στην κλήση συστήματος του *Mach*, `thread_switch()` που επιτρέπει την ανάθεση του επεξεργαστή σε ένα άλλο thread. Οπότε μπορούμε να αναθέσουμε τον επεξεργαστή κατ'ευθείαν στο νήμα που έχει μπλοκάρει και πρέπει να χρονοπρογραμματιστεί.

5.3 Πειραματικά δεδομένα

Προκειμένου να λάβουμε μία εκτίμηση της επίπτωσης της προανάκτησης στην απόδοση του εξυπηρέτη, εκτελέσαμε ακριβώς τα ίδια πειράματα με αυτά του προηγουμένου κεφαλαίου. Έτσι εξακολουθούμε να έχουμε μετρήσεις με σύνδεση 1, 2, 4 και 8 client στον εξυπηρέτη, που εκτελούν select με διαφορετικό selectivity.

Αμέσως όμως δημιουργείται το ζήτημα της επιλογής μεγέθους buffer. Μία καλή

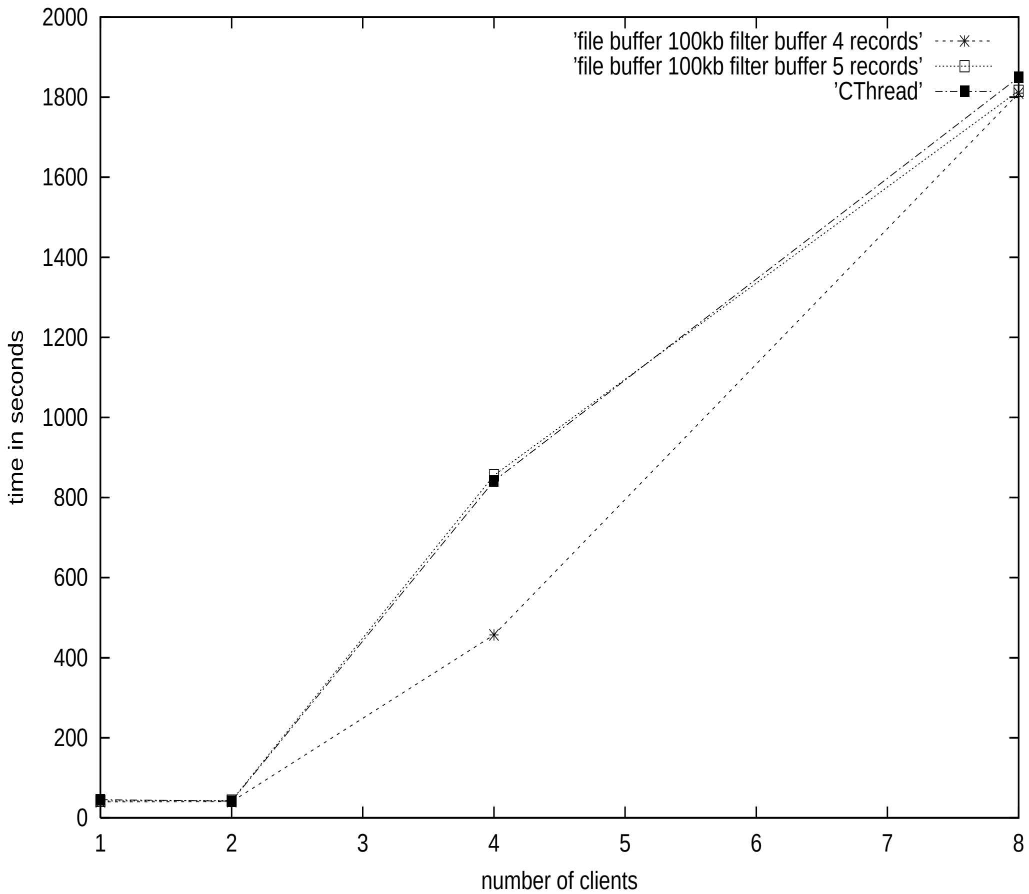


Σχήμα 5.4: Σύγκριση της βιβλιοθήκης CThread με χρήση F-buffer μεγέθους 80000 bytes.

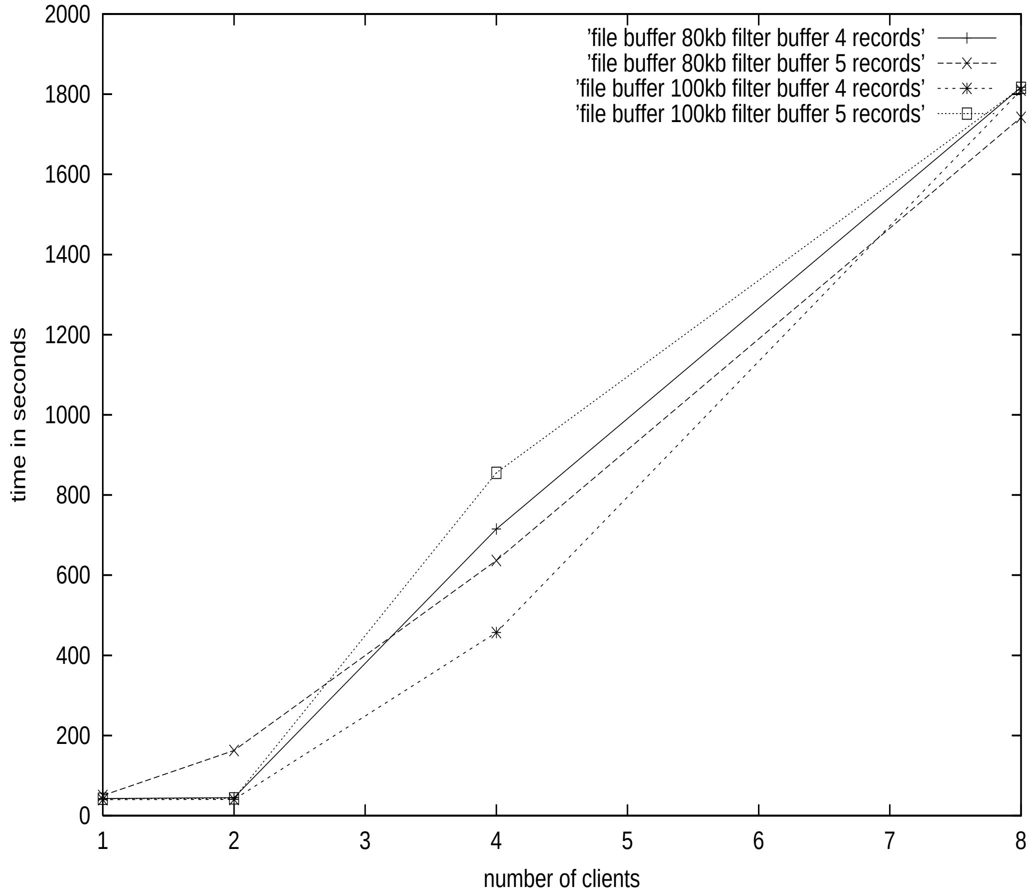
τιμή για το F-buffer φαίνεται να είναι πάνω από 5 εγγραφές καθώς έτσι είναι αρκετά μικρός ώστε να γεμίζει σχετικά γρήγορα αλλά και αρκετά μεγάλος ώστε να μην αδειάζει.΄Οσο για το μέγεθος του R-buffer μεγάλο ρόλο παίζει το selectivity του φίλτρου, καθώς, υποθέτοντας ότι οι εγγραφές έχουν ίση πιθανότητα να επιλεγούν από το φίλτρο, αν έχουμε μικρό selectivity ο buffer θα αργεί να γεμίσει ενώ αν έχουμε μεγάλο selectivity θα γεμίζει πολύ γρήγορα οπότε τα fill thread θα μένουν ανενεργά.

Στο σχήμα 5.4 φαίνεται η σύγκριση της εκτέλεσης των πειραμάτων με χρήση της βιβλιοθήκης CThread του *Hurd* με μέγεθος F-buffer ίσο με 80000 bytes, δηλαδή 8 εγγραφές σύμφωνα με τη δομή των αρχείων μας, και μέγεθος R-buffer ίσο με 4 και 5 εγγραφές. Στο σχήμα φαίνονται και οι μετρήσεις χωρίς χρήση buffer.

Στο σχήμα 5.5 έχουμε τα ίδια πειράματα μόνο που αυτή τη φορά χρησιμοποιείται μέγεθος F-buffer ίσο με 100000 bytes, δηλαδή 10 εγγραφών.



Σχήμα 5.5: Σύγκριση της βιβλιοθήκης CThread με χρήση F-buffer μεγέθους 100000 bytes.



Σχήμα 5.6: Σύγκριση της βιβλιοθήκης CThread με χρήση F-buffer μεγέθους 80000 bytes και 100000 bytes.

Στο σχήμα 5.6 φαίνεται η σύγκριση όλων των πειραμάτων με χρήση buffer.

Γενικά παρατηρούμε ότι έχουμε μία τάξη βελτίωσης του 10–20% με τη χρήση buffers. Παρατηρούμε ότι, όπως περιμέναμε, το μέγεθος F-buffer λίγο πάνω από 5 εγγραφές δίνει και τα καλύτερα αποτελέσματα, μεγαλύτερο μέγεθος buffer σημαίνει μεγαλύτερη καθυστέρηση στο να γεμίσει. Επιπλέον παρατηρούμε ότι για λίγους client αποδίδουν καλύτερα οι περιπτώσεις με μικρό μέγεθος R-buffer, αυτό συμβαίνει επειδή ο R-buffer αργεί να γεμίσει ειδικά για φίλτρα με μικρό selectivity. Με λίγους client όταν αδείαζει ο μικρός R-buffer γεμίζει σχετικά γρήγορα. Για περιπτώσεις όμως με μεγάλο πλήθος client ο μεγαλύτερος R-buffer γεμίζει μεν πιο αργά αλλά αδειάζει και πιο αργά καλύπτοντας έτσι τη διαφορά. Εξάλλου παρατηρούμε μία σαφή μείωση της απόδοσης για μεγάλο πλήθος client όπου και πλησιάζουμε την περίπτωση χωρίς χρήση buffer, οπότε και φαίνεται ότι η αναποτελεσματική εναλλαγή των νημάτων που έχουμε ήδη επισημάνει, ρίχνει την απόδοση.

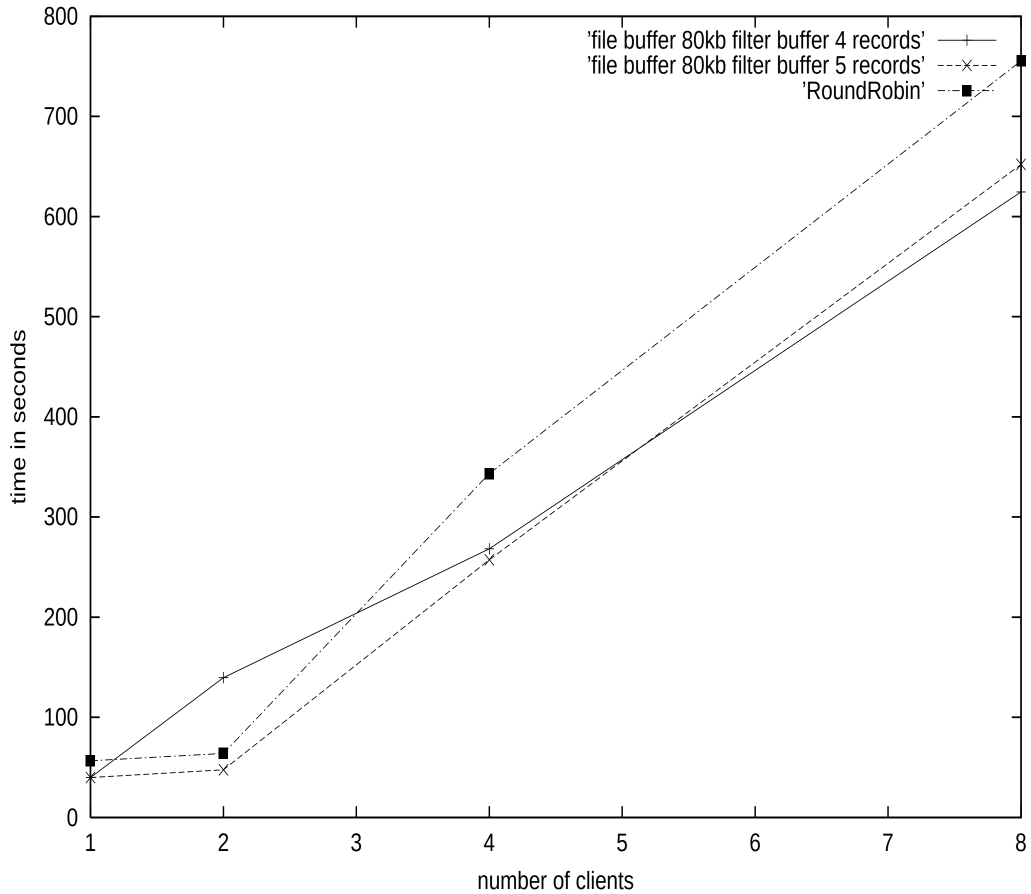
Η επόμενη σειρά πειραματικών μετρήσεων αφορά την δική μας υλοποίηση των CThread. Τα πειράματα είναι πανομοιότυπα με αυτά που περιγράψαμε προηγουμένως, εκτελέστηκαν όμως μόνο για την περίπτωση του αλγόριθμου χρονοπρογραμματισμού roundrobin. Η επιλογή μας αυτή δικαιολογείται από τα συμπεράσματα του προηγούμενου κεφαλαίου όπου τα αποτελέσματα του first come first served είναι το άθροισμα των χρόνων των client όταν τρέχει ο καθένας μόνος του. Η βελτίωση που μπορούμε να περιμένουμε με τη χρήση του first come first server θα είναι αντίστοιχη της βελτίωσης με τη χρήση roundrobin με ένα μόνο client. Επίσης επιλέξαμε τα πειράματα να εκτελεστούν μόνο για κβάντο χρόνου 150 ms αφού μπορούμε και εδώ να γενικεύσουμε με βάση τα συμπεράσματα του προηγούμενου κεφαλαίου, οπότε και αναμένουμε αύξηση της απόδοσης με την αύξηση του κβάντου χρόνου.

Στο σχήμα 5.7 φαίνεται η σύγκριση της εκτέλεσης των πειραμάτων με χρήση πολιτικής χρονοπρογραμματισμού roundrobin, με μέγεθος F-buffer ίσο με 80000 bytes και μέγεθος R-buffer ίσο με 4 και 5 εγγραφές. Στο σχήμα φαίνονται και οι μετρήσεις χωρίς χρήση buffer.

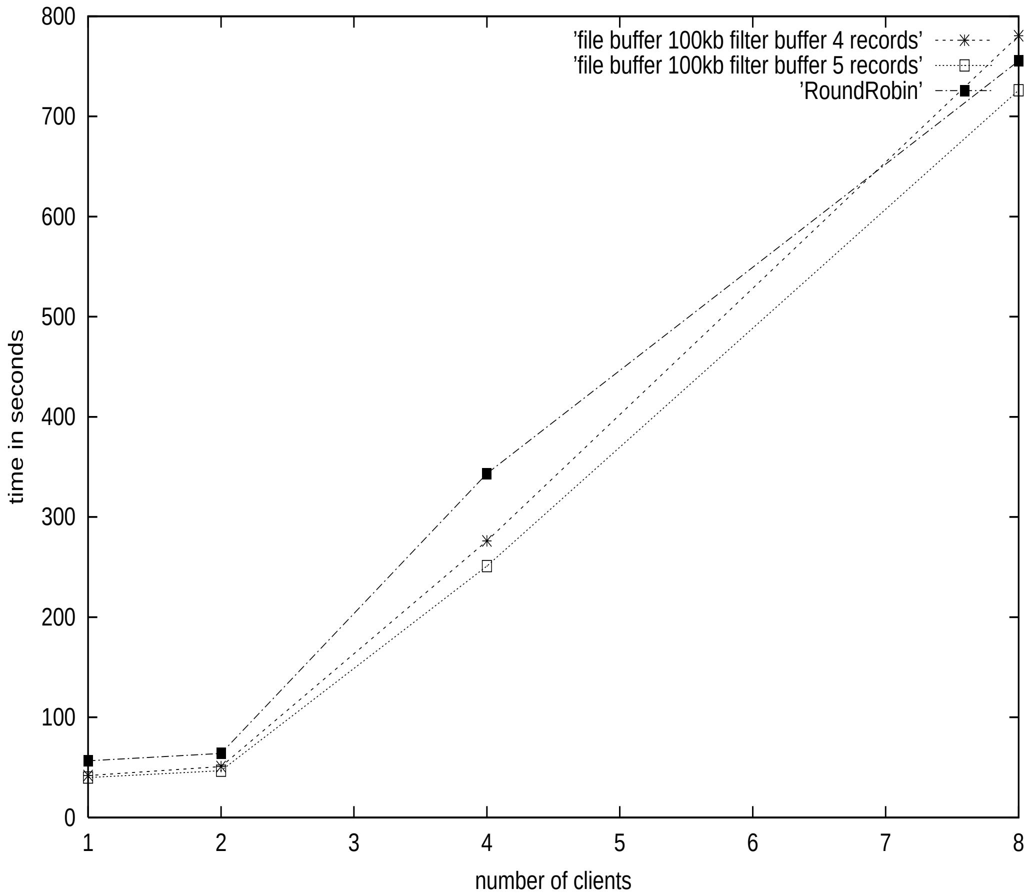
Στο σχήμα 5.8 έχουμε τα ίδια πειράματα μόνο που αυτή τη φορά χρησιμοποιείται μέγεθος F-buffer ίσο με 100000 bytes.

Στο σχήμα 5.9 φαίνεται η σύγκριση όλων των πειραμάτων με χρήση buffer.

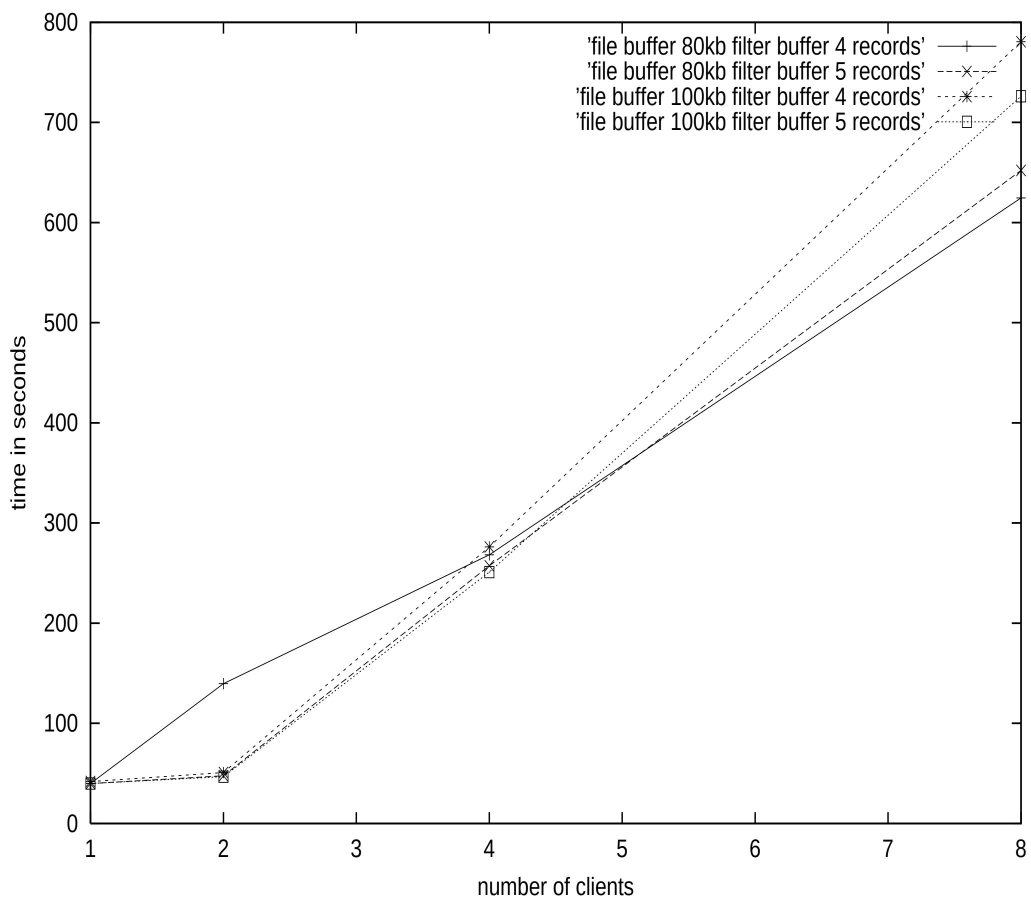
Παρατηρούμε ότι και τώρα έχουμε μία βελτίωση της τάξης του 10–25% ενώ για τα μεγέθη των buffer ισχύουν παρόμοια αποτελέσματα με αυτά των πειραμάτων με χρήση της βιβλιοθήκης CThread του *Hurd*.



Σχήμα 5.7: Σύγκριση αλγορίθμου χρονοπρογραμματισμού roundrobin με χρήση F-buffer μεγέθους 80000 bytes.



Σχήμα 5.8: Σύγκριση αλγορίθμου χρονοπρογραμματισμού roundrobin με χρήση F-buffer μεγέθους 100000 bytes.



Σχήμα 5.9: Σύγκριση αλγορίθμου χρονοπρογραμματισμού roundrobin με χρήση F-buffer μεγέθους 80000 bytes και 100000 bytes.

Κεφάλαιο 6

Σύνοψη και μελλοντική εργασία

6.1 Σύνοψη

Είδαμε ότι προκειμένου να αντιμετωπιστεί η συνεχώς αυξανόμενη ζήτηση για υψηλό ρυθμό μεταφοράς δεδομένων, δεν είναι απαραίτητη η χρήση ακριβών υπολογιστών και συνδέσεων υψηλών ταχυτήτων. Αρκεί ένα σύστημα που θα χρησιμοποιεί πιο αποτελεσματικά τις υπάρχουσες τεχνολογίες. Το σύστημα SmAS υλοποιεί ένα αυτόνομο δίσκο με δυνατότητα σύνδεσης σε δίκτυο, που παρέχει τη δυνατότητα μεταφοράς της επεξεργασίας των δεδομένων στο δίσκο βελτιώνοντας το ρυθμό μεταφοράς δεδομένων.

Στην εργασία αυτή βελτιώσαμε ορισμένα τμήματα του SmAS με την επιλογή ενός λειτουργικού συστήματος που μπορεί να παραμετροποιηθεί εύκολα ώστε να ικανοποιεί τις ανάγκες του συστήματος. Ακόμα τροποποιήσαμε την υλοποίηση των thread προσθέτοντας τη δυνατότητα χρονοπρογραμματισμού με την υλοποίηση των αλγορίθμων round-robin και first-come-first-served. Τα πειράματα με τον αλγόριθμο χρονοπρογραμματισμού round-robin έδειξαν ότι υπήρξε μία σαφής βελτίωση, σε σχέση με την υπάρχουσα υλοποίηση των thread, και μία εξάρτηση της απόδοσης από το κβάντο χρόνου που διατίθεται σε κάθε thread. Αυτό εξηγείται από το γεγονός ότι με μεγάλο κβάντο χρόνου μειώσαμε τον μεγάλο αριθμό αναζητήσεων που εκτελούσε η κεφαλή του σκληρού δίσκου, που οφείλονταν στη συχνή εναλλαγή των thread. Όπως είναι φανερό με τον αλγόριθμο first come first served είχαμε ακόμη μεγαλύτερη βελτίωση των επιδόσεων αφού η κεφαλή του δίσκου διάβαζε σειριακά από ένα αρχείο, αλλά αυτός ο αλγόριθμος είναι κατάλληλος μόνο για περιπτώσεις που δεν μας ενδιαφέρει η δίκαια εξυπηρέτηση των χρηστών.

Έπειτα ασχοληθήκαμε με την υλοποίηση ενός συστήματος προανάκτησης για την ανάγνωση και επεξεργασία δεδομένων πριν αυτά ζητηθούν. Στόχος μας είναι η ελαχιστοποίηση καθυστερήσεων στην ανάγνωση και την επεξεργασία των δεδομένων. Υλοποιήσαμε ένα σύστημα buffer και πολλαπλών νημάτων για την ανάγνωση και επεξεργασία των δεδομένων.

Οι πειραματικές μετρήσεις έδειξαν ότι η χρήση των buffer βελτίωσε κατά ένα σημαντικό ποσοστό την απόδοση του εξυπηρέτη. Ένα άλλο συμπέρασμα που βγήκε από τις πειραματικές μετρήσεις είναι ότι το μέγεθος των buffer παίζει αρκετά σημαντικό ρόλο στη βελτίωση της απόδοσης καθώς για συγκεκριμένο συνδυασμό μπορεί να επιτευχθεί αρκετά μεγάλη αύξηση της απόδοσης.

6.2 Μελλοντικές επεκτάσεις

Παρόλο που η εργασία αυτή πρόσθεσε αρκετές νέες δυνατότητες στο σύστημα SmAS ακόμα υπάρχουν πολλοί τομείς στους οποίους μπορούμε να επεκτάθούμε. Στη παράγραφο αυτή εξετάζουμε μερικές προτάσεις που μπορούν να βελτιώσουν την απόδοση του SmAS, να προσθέσουν νέες λειτουργίες, να αλλάξουν την δομή και τη χρήση του συστήματος SmAS.

6.2.1 Λειτουργικό σύστημα

Η τωρινή υλοποίηση του εξυπηρέτη SmAS απέχει πολύ από την ιδεατή αρχιτεκτονική, που ορίζουμε σαν ένα λειτουργικό σύστημα που υλοποιεί μόνο τις απαραίτητες λειτουργίες για την εξυπηρέτηση client. Η μεταφορά του SmAS στο *Hurd* και η μελέτη των δυνατοτήτων αυτού του λειτουργικού συστήματος, ανοίγουν τον δρόμο για μία νέα υλοποίηση του SmAS που θα αποτελεί τμήμα του *Hurd*.

Συγκεκριμένα το σύστημα SmAS μπορεί να υλοποιηθεί σαν ένα σύνολο από νέους server του *Hurd* ή με την τροποποίηση των υπάρχοντων. Για παράδειγμα η εφαρμογή φίλτρων στα αρχεία μπορεί να γίνει με την εφαρμογή ενός translator (Ενότητα 3.2) πάνω στο αρχείο, οπότε και αυτός θα εκτελεί το φίλτρο, ή με την τροποποίηση του server διαχείρισης αρχείων του *Hurd*. Η υλοποίηση αυτή αποτελεί ένα πολύ δύσκολο έργο αλλά με αυτό τον τρόπο θα έχουμε ένα πραγματικό λειτουργικό σύστημα για αυτόνομους δίσκους.

6.2.2 Προανάκτηση δεδομένων

Το σύστημα προανάκτησης που υλοποιήθηκε περιορίζεται εσωτερικά στον εξυπηρέτη SmAS, δηλαδή έχουμε την επιτάχυνση μόνο μερικών λειτουργιών του εξυπηρέτη SmAS ενώ το σύστημα της μεταφοράς των δεδομένων παραμένει άθικτο. Επόμενο βήμα είναι η υλοποίηση προανάκτησης και στην πλευρά του client οπότε θα έχουμε την αποστολή δεδομένων στο client πριν αυτά ζητηθούν και την αποθήκευσή τους σε τοπικούς buffer. Με αυτό τον τρόπο θα υπάρχει μείωση του χρόνου επικοινωνίας μέσω του δικτύου. Είναι ένα αρκετά περίπλοκο εγχείρημα αφού η διατήρηση της συνέπειας των δεδομένων ανάμεσα στο client και τον εξυπηρέτη είναι αρκετά δύσκολη.

6.2.3 Χρονοπρογραμματισμός

Οι αλγόριθμοι χρονοπρογραμματισμού που υλοποιήσαμε είναι μόλις δύο ενώ υπάρχουν και αρκετοί άλλοι που αναφέρονται στη βιβλιογραφία [11, 12]. Έχουμε ήδη αναφέρει τον αλγόριθμο shortest job first τον οποίο όμως δεν υλοποιήσαμε. Η υλοποίησή του περιλαμβάνει την εύρεση ενός αποτελεσματικού τρόπου *εκτίμησης* του χρόνου εκτέλεσης των φίλτρων, ενώ αναμένονται ενδιαφέροντα αποτελέσμα στην απόδοση του εξυπηρέτη. Χρήσιμη θα είναι και η υλοποίηση ενός αλγόριθμου χρονοπρογραμματισμού με χρήση προτεραιοτήτων. Η ανάθεση προτεραιοτήτων στα φίλτρα είναι ένα θέμα που πρέπει να επιλυθεί στην υλοποίηση αυτού του αλγορίθμου με την επιλογή κατάλληλων κριτηρίων. Ένα κριτήριο θα μπορούσε να είναι ο χρήστης που χρησιμοποιεί το SmAS σε περίπτωση που είναι δυνατός ο διαχωρισμός των χρηστών.

6.2.4 Χρήστες στο SmAS

Η τελευταία πρόταση μας υπενθυμίζει ότι δεν έχει υλοποιηθεί ακόμα έλεγχος πρόσβασης στο σύστημα SmAS κάτι που προϋποθέτει την ύπαρξη χρηστών και τη δυνατότητα πιστοποίησής τους. Μία τέτοια λειτουργία είναι απαραίτητο να προστεθεί προκειμένου να αυξηθεί η ασφάλεια και η αξιοπιστία του συστήματος. Η υλοποίηση μίας τέτοιας δυνατότητας εγείρει θέματα όπως η ασφάλεια των προσωπικών δεδομένων του χρήστη και οι άδειες των χρηστών. Στον τομέα της ασφάλειας μπορούμε να προσθέσουμε και κρυπτογράφηση δεδομένων αλλά αυτό θα έχει σημαντική επίδραση στην απόδοση του εξυπηρέτη ενώ έτσι ξεφεύγουμε από την απλότητα που επιθυμούμε ως τώρα να έχει μία συσκευή SmAS.

6.2.5 Ομαδοποίηση συσκευών SmAS

Η χρήση μίας συσκευής SmAS μπορεί να βελτιώσει την αναζήτηση και μεταφορά δεδομένων, ένα θέμα που προκύπτει είναι πως μπορούμε να εκμεταλλευτούμε τη χρήση περισσοτέρων συσκευών SmAS. Αυτό αποτελεί ένα πολύ ενδιαφέρον πρόβλημα κατανεμημένης επεξεργασίας και ανοίγει ένα τεράστιο πεδίο στη χρήση και εφαρμογή των συσκευών SmAS, με την δημιουργία ομάδων από συκευές.

6.2.6 Πράκτορες και συσκευές SmAS

Επιπλέον πολύ μεγάλο ενδιαφέρον στην κατανεμημένη επεξεργασία δεδομένων παρουσιάζει η τεχνολογία των κινητών πρακτόρων η οποία θα μπορούσε να συνδυαστεί με την τεχνολογία των συσκευών SmAS. Εξάλλου και τα φίλτρα που εκτελούνται στον εξυπηρέτη μπορούν να θεωρηθούν σαν μία περιορισμένη μορφή πράκτορα, οπότε δεν θα είναι ιδιαίτερα δύσκολο να υλοποιηθούν πράκτορες για συστήματα SmAS, αν υλοποιηθεί ένα filter API που θα παρέχει τις απαραίτητες λειτουργίες.

Βιβλιογραφία

[1] A. Acharya, M. Uysal, and J. Saltz, "Active disks: programming model, algorithms and evaluation" In *ASPLOS '98*, 8th Conference on Architectural Support for Programming Languages and Operating Systems, pages 212-217, San Jose, California, October 1998.

[2] K. Keeton, D. A. Patterson, and J. M. Hellerstein, "A case for intelligent disks (idisks)", *SIGMOD Record*, 27(3):42-52, July 1998

[3] E. Riedel, G. Gibson, and C. Faloutsos, "Active storage for large-scale data minig and multimedia", In *VLDB '98, 24th int'l Conference on Very Large Data Bases*, pages 62-73, New York, USA, August 1998.

[4] A. Acharya, M. Uysal, and J. Saltz, "An evaluation of architectural alternatives for rapidly growing datasets: active disks, clusters, SMPs", Technical Report, Dept. of Computer Science, University of California, Santa Barbara, Technical Report TRCS98-27, October 1998

[5] A. Acharya, M. Uysal, and J. Saltz, "Evaluation of active disks for decision support databases", In *HPCA,* 2000.

[6] V.V. Dimakopoulos, A. Kinalis, E. Pitoura and I. Tsoulos, "On Deploying and Executing Data-Intensive Code on Smart Autonomous Storage (SmAS) Disks", *Proceedings ADBIS-DASFAA 2000*, International Conference on Database Systems for Advanced Applications, Prague, Czech Republic, pp. 323–330, Sept. 2000.

[7] V.V. Dimakopoulos, A. Kinalis, S. Mastrogiannakis and E. Pitoura, "The Smart Autonomous Storage (SmAS) System", *Proceedings PACRIM01*, 2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, Victoria, B.C., Canada, Aug. 2001.

[8] V.V. Dimakopoulos, S. Mastrogiannakis and E. Pitoura, “Application Programming in SmAS”, Technical Report TR-15-2001, Dept. of Computer Science, University of Ioannina, 2001.

[9] Keith Loepere, “Mach 3 Server Writer’s Guide” , Open Software Foundation and Carnegie Mellon University, NORMA-MK12, user15:July 15, 1992.

[10] Keith Loepere, “Mach 3 Kernel Principles” , Open Software Foundation and Carnegie Mellon University, NORMA-MK12, user15:July 15, 1992.

[11] A. S. Tanenbaum and A. S. Woodhall, ”Operating System Design and Implementation”.

[12] A. Silberschatz and P. B. Galvin, ”Operating Systems Concepts”, 5th edition, Addison Wesley Publishing Company 1998.

[13] Andrew S. Tanenbaum, ”Distributed Operating Systems”, Prentice Hall 1995, Chapter 8, pages 431-474 (Case Study 2).

[14] Andrew S. Tanenbaum, ”Modern Operating Systems”, Prentice Hall 1992, Chapter 15, pages 637-682 (Case Study 4).

[15] Joseph Boykin, David Kirschen, Alan Langerman, Susan LoVerso, ”Programming Under Mach”, Addison-Wesley Publishing Co,. 1993.

[16] The GNU project
http://www.gnu.org/

[17] The Hurd
http://www.gnu.org/software/hurd

[18] Introduction to translators
http://www.debian.org/ports/hurd/hurd-doc-translator

[19] The Mach project
http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/mach/public/www/mach.html

[20] The Mach4 project
http://www.cs.utah.edu/flux/mach4/html/Mach4-proj.html

[21] The xMach project
http://xmach.org

[22] BSD Operating Systems
http://www.bsd.org

[23] The LINUX kernel homepage
http://www.linux.org

Παράρτημα A

Κώδικας SmAS εξυπηρέτη

Λίστα αρχείων:

`LinuxMakefile`
`Makefile`
`server/buffers.c`
`server/buffers.h`
`server/files.c`
`server/files.h`
`server/filters.c`
`server/filters.h`
`server/handlers.c`
`server/handlers.h`
`server/server.c`
`server/server.h`
`server/smas_stop.c`
`server/threadfunc.c`
`server/timer.h`
`include/smasdefs.h`
`include/socks.c`
`include/socks.h`

A.1 LinuxMakefile

```
#
# Makefile for SmAS Server
# VVD, May 2000
#
PROG = smas_start
CC   = gcc
O    = obj
```

```
#
# Various Modules
#
MAINSRV    = server
FILES      = files
FILTERS    = filters
HANDLERS   = handlers
BUFFERS    = buffers
THFUNC     = threadfunc
OBJS       = $(O)/$(MAINSRV).o $(O)/$(FILES).o $(O)/$(FILTERS).o \
             $(O)/$(HANDLERS).o $(O)/$(BUFFERS).o $(O)/$(THFUNC).o

#
# Options
#
COPTS      = -g -I. -I../include -D_REENTRANT -DLINUX_DIST #-DSMAS_SRV_DEBUG
LOPTS      = -lpthread -ldl

#
# Stopper
#
STOPPER = smas_stop

#
# Actual rules
#
all:    $(PROG) $(STOPPER)

$(STOPPER): smas_stop.c
$(CC) -o smas_stop smas_stop.c

$(PROG): $(OBJS)
$(CC) -o $(PROG) $(OBJS) $(COPTS) $(LOPTS)

$(O)/%.o: %.c server.h ../include/socks.c
$(CC) -c -o $(O)/$*.o $(COPTS) $<
clean:
rm -f obj/*.o
rm -f smas_start smas_stop
```

A.2 Makefile

```
#
# Makefile for SmAS Server
# VVD, May 2000
#
PROG = smas_start
HPROG = hsmas_start     #There has to be an easier way than
                        #this; read on to understand what I mean
CC   = gcc
O    = obj

#
# Various Modules
#
MAINSRV    = server
FILES      = files
FILTERS    = filters
HANDLERS   = handlers
BUFFERS    = buffers
THFUNC     = threadfunc
OBJS       = $(O)/$(MAINSRV).o $(O)/$(FILES).o $(O)/$(FILTERS).o \
             $(O)/$(HANDLERS).o $(O)/$(BUFFERS).o $(O)/$(THFUNC).o
HOBJS       = $(O)/$(MAINSRV).ho $(O)/$(FILES).ho $(O)/$(FILTERS).ho \
              $(O)/$(HANDLERS).ho $(O)/$(BUFFERS).ho $(O)/$(THFUNC).ho
      #Stupid way to seperate object files

#
# Options
#
#Use these for native cthreads
HCOPTS      = -g -I. -I../include -D_REENTRANT -DHURD_DIST  #-DSMAS_SRV_DEBUG
HLOPTS      = -lthreads -ldl
#Use these for mycthreads
COPTS      = -g -I. -I../include -D_REENTRANT -DHURD_DIST -DUSE_MYCTHREAD \
             #-DUSE_COUNTER #-DSMAS_SRV_DEBUG
```

```
LOPTS      =  -ldl -lmythreads
#
# Stopper
#
STOPPER = smas_stop
#
# Actual rules
#
all:    $(PROG) $(STOPPER)
$(STOPPER): smas_stop.c
$(CC) -o smas_stop smas_stop.c
$(PROG): $(OBJS)  ../libthreads/libmythreads.a
$(CC) -o $(PROG) $(OBJS) $(COPTS) $(LOPTS)
$(O)/%.o: %.c server.h ../include/socks.c
$(CC) -c -o $(O)/$*.o $(COPTS) $<
#Using HURD's cthreads
hurd: $(HPROG) $(STOPPER)
cp -f hsmas_start smas_start

$(HPROG): $(HOBJS)
$(CC) -o $(HPROG) $(HOBJS) $(HCOPTS) $(HLOPTS)
$(O)/%.ho: %.c server.h ../include/socks.c
$(CC) -c -o $(O)/$*.ho $(HCOPTS) $<
#Cleanup
clean:
rm -f $(O)/*o
rm -f smas_start smas_stop hsmas_start
```

A.3 server/buffers.c

```
/*
  Buffer implementation for SmAS
  Kinalis Athanasios July 2001
 */
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "buffers.h"
#include "server.h"
#include <string.h>
#ifdef USE_MYCTHREAD
#include "../libthreads/cthread_internals.h"
/*Get's cproc from cthread - See cthreads.h#cthread_assoc()*/
#define get_cproc(c) ((cproc_t)((c)->ur))
#endif
int FILE_BUFFER_SIZE = 50000;
int FILTER_BUFFER_SIZE = 5;
int file_buf_init(struct file_buffer *buf, int fd, int size,
                  void *th)
{
    buf->src.fd = fd;
    buf->buffer = NULL;
#ifdef HURD_DIST
    buf->fillth = (cthread_t) th;
    memset(&buf->lock, 0, sizeof(struct mutex));
    memset(&buf->wait_lock, 0, sizeof(struct mutex));
    memset(&buf->data_needed, 0, sizeof(struct condition));
    memset(&buf->stop, 0, sizeof(struct mutex));
#endif
#ifdef LINUX_DIST
    buf->fillth = (pthread_t) th;
    pthread_mutex_init(&buf->lock, NULL);
```

```
    pthread_mutex_init(&buf->wait_lock, NULL);
    pthread_cond_init(&buf->data_needed, NULL);
    pthread_mutex_init(&buf->stop, NULL);
#endif
    alloc_file_buf(buf, size);
    buf->status = BUF_SFETCH;

    return 0;
}

/*Always invalidates buffer*/
/*Sets buffer reading position. Call after buf_init()*/
int set_file_buf_pos(file_buffer_t * buf, int f)
{
    mutex_lock(&buf->lock);
    if (f == BUF_NOPOS) {       /* This is a socket/filter buffer - no
                                   file positioning available */
        buf->filepos = -1;
        buf->vstart = -1;
        buf->voffset = 0;
        buf->vlength = 0;
    } else if (f == BUF_CURPOS) {       /* Moves buffer position to
                                           current file pointer position */
        buf->filepos = lseek(buf->src.fd, 0, SEEK_CUR);
        buf->vstart = buf->filepos;
        buf->voffset = 0;
        buf->vlength = 0;
    } else if (f == BUF_NORMALPOS) {    /* Moves file pointer to buffer
                                           file position */
        buf->filepos =
            lseek(buf->src.fd, buf->filepos, SEEK_SET);
        buf->vstart = buf->filepos;
        buf->voffset = 0;
        buf->vlength = 0;
    }
    if (buf->status & BUF_SEOF) /* Also remove EOF bit */
        buf->status &= (~BUF_SEOF);
    mutex_unlock(&buf->lock);
    return buf->filepos;
}

int alloc_file_buf(file_buffer_t * buf, int size)
{
    mutex_lock(&buf->lock);
    buf->size = size;
    buf->buffer =
        (unsigned char *) malloc(sizeof(unsigned char) * size);
    buf->vstart = buf->filepos;
    buf->voffset = 0;
    buf->vlength = 0;
    mutex_unlock(&buf->lock);
    return 0;
}

int dealloc_file_buf(file_buffer_t * buf)
{
    mutex_lock(&buf->lock);
    free(buf->buffer);
    mutex_unlock(&buf->lock);
    return 0;
}

/*Indicates that all reads from buffer are valid, advance filepos*/
void file_buf_validate(file_buffer_t * buf)
{
    struct stat st;

    mutex_lock(&buf->lock);
```

```
    buf->filepos = buf->vstart;
    fstat(buf->src.fd, &st);    /* Check for EOF */
    if (buf->filepos == st.st_size && buf->vlength == 0)
        buf->status |= BUF_SEOF;
    mutex_unlock(&buf->lock);
}

/*Indicates that reads from buffer are valid until pos*/
void file_buf_validate2(file_buffer_t * buf, int pos)
{
    struct stat st;

    mutex_lock(&buf->lock);
    if (pos > buf->vstart) {
        mutex_unlock(&buf->lock);
        return;
    }
    buf->filepos = pos;
    fstat(buf->src.fd, &st);    /* Check for EOF */
    if (buf->filepos == st.st_size && buf->vlength == 0)
        buf->status |= BUF_SEOF;
    mutex_unlock(&buf->lock);
}

/*Fetches as much bytes as possible(?) from fd to fill buffer*/
int file_buf_fetch(file_buffer_t * buf)
{
    int i = 0, f = 0;

    mutex_lock(&buf->lock);
    if ((f = buf->size - buf->vlength) != 0) {  /* Check if buffer is
                                                   full; if not f holds
                                                   the amount of free
                                                   space */
        i = buf->size - (buf->voffset + buf->vlength);  /* i shows how far
                                                           from buffer's
                                                           end data
                                                           extends */
        if (i == 0) {
            buf->voffset =
                (buf->voffset == buf->size) ? 0 : buf->voffset;
            i = buf->size - buf->vlength;
        }
        if (i < 0) {            /* Buffer's end is full and there is a gap
                                   at the begging of buffer. */
            buf->vlength += (i =
                             read(buf->src.fd, &buf->buffer[-i],
                                  f));
        } else {                /* There is free space in buffer's end.
                                   Fill it. */
            buf->vlength += (i =
                             read(buf->src.fd,
                                  &buf->
                                  buffer[(buf->voffset +
                                          buf->vlength) %
                                         buf->size], i));
        }
    }
    if (i == 0) {               /* Check for EOF */
        struct stat st;
        fstat(buf->src.fd, &st);
        if (buf->vstart == st.st_size && buf->vlength == 0)
            buf->status |= BUF_SEOF;
    }
    mutex_unlock(&buf->lock);
    return i;
}

/*Appends l bytes from array b to buffer buf*/
```

```
int file_buf_append(file_buffer_t * buf, unsigned char *b, int l)
{
    int i = 0, f = 0;

    mutex_lock(&buf->lock);
    if ((f = buf->size - buf->vlength) != 0) {  /* Check if buffer is
                                                   full; if not f holds
                                                   the amount of free
                                                   space */
        f = (f > l) ? l : f;    /* append f bytes, as much as buffer can
                                   hold or l */
        i = buf->size - (buf->voffset + buf->vlength);  /* i shows how far
                                                           from buffer's
                                                           end data
                                                           extends */
        if (i < 0) {            /* Buffer's end is full and there is free
                                   space at the begging of buffer. */
            memcpy(buf->buffer + (-i), b, f);
        } else {                /* There is free space in buffer's end.
                                   Fill it. */
            memcpy(buf->buffer + buf->voffset + buf->vlength, b,
                   i);
            if (f - i > 0)      /* There more free space at the begging of
                                   buffer */
                memcpy(buf->buffer, b + i, f - i);
        }
        buf->vlength += f;
    }
    mutex_unlock(&buf->lock);

    return f;
}

/*Reads s bytes in array b from buffer buf*/
/*If s bytes not available blocks until fetched*/
int file_buf_read(file_buffer_t * buf, unsigned char *b, int s)
{
    int i, d, tot = 0;

    do {
        i = d = 0;
        mutex_lock(&buf->lock);

        if (s > buf->vlength) {
            d = s - buf->vlength;       /* Store additional required data */
            s = buf->vlength;   /* Read all contents - watch for
                                   buf->vlength==0 */
        }
        if (s > 0 && (i = buf->size - buf->voffset) >= s) {
            memcpy(b, &buf->buffer[buf->voffset], s);
            b += s;
            buf->voffset = (buf->voffset + s) % buf->size;
        } else if (i > 0) {     /* wrap around */
            memcpy(b, &buf->buffer[buf->voffset], i);
            b += i;
            buf->voffset = 0;   /* continue from beggining */
            memcpy(b, buf->buffer, s - i);
            buf->voffset += (s - i);

        }
        buf->vstart += s;
        buf->vlength -= s;
        tot += s;
        if (d > 0) {            /* We require additional data */
            if (buf_eof(buf)) { /* Check for EOF */
                mutex_unlock(&buf->lock);
                return tot;
            }
            mutex_unlock(&buf->lock);
```

```
#ifdef USE_MYCTHREAD
            thread_switch(get_cproc(buf->fillth)->mach_thread,
                          SWITCH_OPTION_DEPRESS, 30);
#else
            mutex_lock(&buf->wait_lock);
            condition_wait(&buf->data_needed, &buf->wait_lock); /* Block
                                                                   until
                                                                   data is
                                                                   there */
            mutex_unlock(&buf->wait_lock);
#endif
            s = d;              /* Prepare for next iterration */
        }
    } while (d != 0);
    mutex_unlock(&buf->lock);
    return tot;
}

/*Probably not the best way to do that*/
/*Returns in vs vstart value after this read*/
int file_buf_read2(file_buffer_t * buf, unsigned char *b, int s,
                   int *vs)
{
    int i, d, tot = 0;

    do {
        i = d = 0;
        mutex_lock(&buf->lock);

        if (s > buf->vlength) {
            d = s - buf->vlength;       /* Store additional required data */
            s = buf->vlength;   /* Read all contents - watch for
                                   buf->vlength==0 */
        }
        if (s > 0 && (i = buf->size - buf->voffset) >= s) {
            memcpy(b, &buf->buffer[buf->voffset], s);
            b += s;
            buf->voffset = (buf->voffset + s) % buf->size;
        } else if (i > 0) {     /* wrap around */
            memcpy(b, &buf->buffer[buf->voffset], i);
            b += i;
            buf->voffset = 0;   /* continue from beggining */
            memcpy(b, buf->buffer, s - i);
            buf->voffset += (s - i);

        }
        buf->vstart += s;
        buf->vlength -= s;
        tot += s;
        if (d > 0) {            /* We require additional data */
            if (buf_eof(buf)) { /* Check for EOF */
                *vs = buf->vstart;
                mutex_unlock(&buf->lock);
                return tot;
            }
            mutex_unlock(&buf->lock);
#ifdef USE_MYCTHREAD
            thread_switch(get_cproc(buf->fillth)->mach_thread,
                          SWITCH_OPTION_DEPRESS, 30);
#else
            mutex_lock(&buf->wait_lock);
            condition_wait(&buf->data_needed, &buf->wait_lock); /* Block
                                                                   until
                                                                   data is
                                                                   there */
            mutex_unlock(&buf->wait_lock);
#endif
            s = d;              /* Prepare for next iterration */
        }
    } while (d != 0);
```

```
    *vs = buf->vstart;
    mutex_unlock(&buf->lock);
    return tot;
}
void *file_buf_destroy(file_buffer_t * buf)
{
    void *res;
    mutex_lock(&buf->lock);
    buf->status |= BUF_SDESTROY;
    mutex_unlock(&buf->lock);
#ifdef HURD_DIST
    res = cthread_join(buf->fillth);
#endif
#ifdef LINUX_DIST
    pthread_join(buf->fillth, &res);
#endif
    dealloc_file_buf(buf);
    return res;
}
int file_buf_readvalid(file_buffer_t * buf)
{
    int res = 0;
    mutex_lock(&buf->lock);
    if (buf->vstart == buf->filepos)
        res = 1;
    mutex_unlock(&buf->lock);
    return res;
}
/***********************************************/
/***********************************************/
/***********************************************/
/****************FILTER BUFFERS*****************/
/***********************************************/
/***********************************************/
/***********************************************/

int filter_buf_init(struct filter_buffer *buf, int fd,
                    int capacity, int size)
{
    buf->src.fd = fd;
    buf->buffer = NULL;
#ifdef HURD_DIST
/* buf->fillth=(cthread_t)th; This is initialized elsewhere*/
    memset(&buf->lock, 0, sizeof(struct mutex));
    memset(&buf->wait_lock, 0, sizeof(struct mutex));
    memset(&buf->data_needed, 0, sizeof(struct condition));
    memset(&buf->stop, 0, sizeof(struct mutex));
#endif
#ifdef LINUX_DIST
    pthread_mutex_init(&buf->lock, NULL);
    pthread_mutex_init(&buf->wait_lock, NULL);
    pthread_cond_init(&buf->data_needed, NULL);
    pthread_mutex_init(&buf->stop, NULL);
#endif
    alloc_filter_buffer(buf, capacity, size);
    buf->status = BUF_SFETCH;
    return 0;
}
int alloc_filter_buffer(filter_buffer_t * buf, int capacity,
                        int size)
{
    int i;
    mutex_lock(&buf->lock);
```

```
    buf->capacity = capacity;
    buf->size = size;
    buf->buffer =
        (rec_buf_t *) malloc(sizeof(rec_buf_t) * capacity);
    for (i = 0; i < capacity; i++) {
        buf->buffer[i].size = 0;        /* Means no data available */
        buf->buffer[i].buffer =
            (unsigned char *) malloc(sizeof(unsigned char) *
                                     size);
    }
    buf->vstart = buf->filepos;
    buf->voffset = 0;
    buf->vlength = 0;
    mutex_unlock(&buf->lock);
    return 0;
}

int dealloc_filter_buf(filter_buffer_t * buf)
{
    int i;

    mutex_lock(&buf->lock);
    for (i = 0; i < buf->capacity; i++)
        free(buf->buffer[i].buffer);
    free(buf->buffer);
    mutex_unlock(&buf->lock);
    return 0;
}

int set_filter_buf_pos(filter_buffer_t * buf, int f)
{
    mutex_lock(&buf->lock);
    if (f == BUF_NOPOS) {       /* This is a socket/filter buffer - no
                                   file positioning available */
        buf->filepos = -1;
        buf->vstart = -1;
        buf->voffset = 0;
        buf->vlength = 0;
        buf->lastvs = -1;
    } else if (f == BUF_CURPOS) {       /* Moves buffer position to
                                           current file pointer position */
        buf->filepos = lseek(buf->src.fd, 0, SEEK_CUR);
        buf->vstart = buf->filepos;
        buf->voffset = 0;
        buf->vlength = 0;
        buf->lastvs = -1;
    } else if (f == BUF_NORMALPOS) {    /* Moves file pointer to buffer
                                           file position */
        buf->filepos =
            lseek(buf->src.fd, buf->filepos, SEEK_SET);
        buf->vstart = buf->filepos;
        buf->voffset = 0;
        buf->vlength = 0;
        buf->lastvs = -1;
    }
    if (buf->status & BUF_SEOF) /* Also remove EOF bit */
        buf->status &= (~BUF_SEOF);
    mutex_unlock(&buf->lock);
    return buf->filepos;
}

int filter_buf_append_record(filter_buffer_t * buf,
                             unsigned char *rec, int size,
                             int vs)
{
    mutex_lock(&buf->lock);
    if (buf->vlength < buf->capacity)
        buf->vlength++;
```

```
    else {
        mutex_unlock(&buf->lock);
        return 0;
    }
    memcpy(buf->
           buffer[(buf->voffset + buf->vlength -
                   1) % buf->capacity].buffer, rec, size);
    buf->buffer[(buf->voffset + buf->vlength - 1) %
                buf->capacity].size = size;
    buf->buffer[(buf->voffset + buf->vlength - 1) %
                buf->capacity].vs = vs;
    mutex_unlock(&buf->lock);

    return size;
}

int filter_buf_nextrecsize(filter_buffer_t * buf)
{
    int rs = 0;

    mutex_lock(&buf->lock);
    rs = buf->buffer[buf->voffset].size;
    mutex_unlock(&buf->lock);
    return rs;
}

int filter_buf_nextrec(filter_buffer_t * buf, unsigned char *dst)
{
    int s = 0, done = 0;

    mutex_lock(&buf->lock);
    if (buf_eof(buf)) {
        mutex_unlock(&buf->lock);
        return 0;
    }
    do {
        if (buf->vlength > 0) {
            memcpy(dst, buf->buffer[buf->voffset].buffer, s =
                   buf->buffer[buf->voffset].size);
            buf->lastvs = buf->buffer[buf->voffset].vs;
            buf->voffset = (buf->voffset + 1) % buf->capacity;
            buf->vlength--;
            done = 1;
        }
        mutex_lock(&((file_buffer_t *) buf->src.buf)->lock);
        if (buf->vlength == 0 && buf_eof((file_buffer_t *) buf->src.buf)) {     /* Check
                                                                                   for
                                                                                   EOF
                                                                                 */
            buf->status |= BUF_SEOF;
            done = 1;
        }
        mutex_unlock(&((file_buffer_t *) buf->src.buf)->lock);
        mutex_unlock(&buf->lock);
        if (done)
            return s;
#ifdef USE_MYCTHREAD
        thread_switch(get_cproc(buf->fillth)->mach_thread,
                      SWITCH_OPTION_DEPRESS, 30);
#else
        mutex_lock(&buf->wait_lock);
        condition_wait(&buf->data_needed, &buf->wait_lock);     /* Block
                                                                   until
                                                                   data is
                                                                   there */
        mutex_unlock(&buf->wait_lock);
#endif
    } while (!done);

    return s;
}
```

```
void filter_buf_validate(filter_buffer_t * buf)
{
    mutex_lock(&buf->lock);
    file_buf_validate2((file_buffer_t *) buf->src.buf,
                       buf->lastvs);
    if (buf->vlength == 0
        && buf_eof((file_buffer_t *) buf->src.buf))
        buf->status |= BUF_SEOF;
    mutex_unlock(&buf->lock);
}

void *filter_buf_destroy(filter_buffer_t * buf)
{
    void *res;

    mutex_lock(&buf->lock);
    buf->status |= BUF_SDESTROY;
    mutex_unlock(&buf->lock);
#ifdef HURD_DIST
    res = cthread_join(buf->fillth);
#endif
#ifdef LINUX_DIST
    pthread_join(buf->fillth, &res);
#endif
    dealloc_filter_buf(buf);

    return res;
}
```

A.4 server/buffers.h

```
#ifndef BUFFERS_H
#define BUFFERS_H

#ifdef HURD_DIST
#ifdef USE_MYCTHREADS
#include "../libthreads/cthreads.h"     /* Use relative path to avoid
                                           conflicts */
#include "../libthreads/cthread_internals.h"
#else
#include <cthreads.h>
#endif
#endif
#ifdef LINUX_DIST
#include <pthread.h>
#define cthread_exit(x) pthread_exit(x)
#define cthread_join(x) pthread_join(x);
#define cthread_self() pthread_self()
#define mutex_lock(t) pthread_mutex_lock(t)
#define mutex_unlock(t) pthread_mutex_unlock(t)
#define condition_wait(t,y) pthread_cond_wait(t,y)
#define condition_signal(t) pthread_cond_signal(t)
#define cthread_yield()
#endif

#define BUF_SFETCH 1
#define BUF_SDESTROY 2
#define BUF_SEOF 4

extern int FILE_BUFFER_SIZE;
extern int FILTER_BUFFER_SIZE;


typedef struct file_buffer {
    int status;                 /* The state of this buffer */
    int size;                   /* Max data length this buffer can hold.
                                   Value of -1 means this buffer should
                                   be discarded */
    unsigned char *buffer;      /* Buffer */
```

```
    union {
        int fd;
        void *buf;
    } src;                      /* The file descriptor this buffer is
                                   reading from or another buffer */
    int filepos;                /* The normal (without buffering) file
                                   position Note: For some buffers it's
                                   meaningless (buffers for
                                   sockets,filters) Set to -1 for these
                                   fds [set_buf_pos( buf, BUF_NOPOS)] */
    int vstart;                 /* The first byte from file (file offset)
                                   that is contained in buffer (filepos to
                                   file length) Note: For some buffers
                                   it's meaningless (buffers for
                                   sockets,filters) */
    int voffset;                /* Valid data offset in buffer (0 to
                                   buflen-1) */
    int vlength;                /* Length of valid data stored in buffer
                                   (0 to buflen) */
#ifdef HURD_DIST
    cthread_t fillth;           /* The thread that fills this buffer.
                                   Controlling thread. */
    struct mutex lock;          /* For locking access to buffer */
    struct mutex wait_lock;     /* For locking access to buffer */
    struct condition data_needed;
    struct mutex stop;          /* For blocking buffer's fill thread */
#endif
#ifdef LINUX_DIST
    pthread_t fillth;           /* The thread that fills this buffer
                                   (probably not required under LINUX */
    pthread_mutex_t lock;       /* For locking access to buffer */
    pthread_mutex_t wait_lock;
    pthread_cond_t data_needed;
    pthread_mutex_t stop;       /* For blocking buffer's fill thread */
#endif
} file_buffer_t;

typedef struct rec_buf {
    int size;
    int vs;
    unsigned char *buffer;
} rec_buf_t;

typedef struct filter_buffer {
    int status;                 /* The state of this buffer */
    int capacity;               /* Max records this buffer can hold. Value
                                   of -1 means this buffer should be
                                   discarded */
    /* Usually this should be a small number e.g. 5 */
    int size;                   /* Max size of each record */
    rec_buf_t *buffer;          /* Array of buffers for records */
    union {
        int fd;
        void *buf;
    } src;                      /* The file descriptor this buffer is
                                   reading from or another buffer */
    int filepos;                /* The normal (without buffering) file
                                   position Note: For some buffers it's
                                   meaningless (buffers for
                                   sockets,filters) Set to -1 for these
                                   fds [set_buf_pos( buf, BUF_NOPOS)] */
    int vstart;                 /* The first byte from file (file offset)
                                   that is contained in buffer (filepos
                                   to file length) Note: For some buffers
                                   it's meaningless (buffers for
```

```
                                   sockets,filters) */
    int voffset;                /* Valid data offset in buffer (0 to
                                   capacity-1) */
    int vlength;                /* Length of valid data stored in buffer
                                   (0 to capacity) */
    int lastvs;
#ifdef HURD_DIST
    cthread_t fillth;           /* The thread that fills this buffer.
                                   Controlling thread. */
    struct mutex lock;          /* For locking access to buffer */
    struct mutex wait_lock;     /* For locking access to buffer */
    struct condition data_needed;
    struct mutex stop;          /* For blocking buffer's fill thread */
#endif
#ifdef LINUX_DIST
    pthread_t fillth;           /* The thread that fills this buffer
                                   (probably not required under LINUX */
    pthread_mutex_t lock;       /* For locking access to buffer */
    pthread_mutex_t wait_lock;
    pthread_cond_t data_needed;
    pthread_mutex_t stop;       /* For blocking buffer's fill thread */
#endif
} filter_buffer_t;

/*Constants to define buffer-file seek used by set_buf_pos()*/
#define BUF_NOPOS  (-1)
#define BUF_CURPOS (-2)
#define BUF_NORMALPOS (-3)

#define buf_eof(b) ((b)->status&BUF_SEOF)

#define filter_buf_invalidate(b) set_filter_buf_pos(b,BUF_NOPOS);
#define file_buf_invalidate(b) set_file_buf_pos(b,BUF_NORMALPOS);

extern int file_buf_init(file_buffer_t * buf, int fd, int size,
                         void *th);
extern int set_file_buf_pos(file_buffer_t * buf, int f);
extern int alloc_file_buffer(file_buffer_t * buf, int size);
extern int dealloc_file_buf(file_buffer_t * buf);
extern void file_buf_validate(file_buffer_t * buf);
extern void file_buf_validate2(file_buffer_t * buf, int vs);
extern int file_buf_fetch(file_buffer_t * buf);
extern int file_buf_append(file_buffer_t * buf, unsigned char *b,
                           int l);
extern int file_buf_read(file_buffer_t * buf, unsigned char *b,
                         int s);
extern int file_buf_read2(file_buffer_t * buf, unsigned char *b,
                          int s, int *vs);
extern int file_buf_readvalid(file_buffer_t * buf);

extern int filter_buf_init(filter_buffer_t * buf, int fd,
                           int capacity, int size);
extern int set_filter_buf_pos(filter_buffer_t * buf, int f);
extern int alloc_filter_buffer(filter_buffer_t * buf,
                               int capacity, int size);
extern int dealloc_filter_buf(filter_buffer_t * buf);

extern int filter_buf_append_record(filter_buffer_t * buf,
                                    unsigned char *rec, int size,
                                    int vs);
extern int filter_buf_nextrecsize(filter_buffer_t * buf);
extern int filter_buf_nextrec(filter_buffer_t * buf,
                              unsigned char *dst);
extern void filter_buf_validate(filter_buffer_t * buf);
extern void *filter_buf_destroy(filter_buffer_t * buf);

#endif
```

A.5 server/files.c

```
/* FILES.C
 * Code for storing and handling info regarding files opened by clients
 *
 * Revisions:
 * ==========
 * AFTS 0.2.1 06/06/2000
 *   Added file_cleanup_owner() to cleanup (close & delete) any
 *   not-closed files of a departing client
 *
 * AFTS 0.2 05/06/2000
 *   Completely rewritten by VVD, CS, UoI
 *
 * AFTS 0.1 ??/06/2000
 *   Untouched
 */
#include <stdio.h>
#include <stdlib.h>
#if 0
#ifdef HURD_DIST
#ifdef USE_MYCTHREADS
#include "../libthreads/cthreads.h"     /* Use relative path to avoid
                                           conflicts */
#include "../libthreads/cthread_internals.h"
#else
#include <cthreads.h>
#endif
#endif
#ifdef LINUX_DIST
#include <pthread.h>
#define cthread_self() pthread_self()
#define mutex_lock(t) pthread_mutex_lock(t)
#define mutex_unlock(t) pthread_mutex_unlock(t)
#endif
#endif

#include "buffers.h"
#include "smasdefs.h"
#include "server.h"
#include "files.h"

extern void *readth(void *);

/* The basic structure
 */
typedef struct {
    int ffd;                    /* The file descriptor */
    int sfd;                    /* The client's (owner's) socket */
    int fid;                    /* The applied filter (-1 if none) */
    char *fmem;                 /* The memory needed by (and allocated to)
                                   the filter */
    file_buffer_t *buf;         /* The buffer for this file */
} file_t;

/* The table of opened files
 */
static file_t *files = NULL;    /* Table of open files */
static int filenum = 0;         /* Size of table */
#ifdef HURD_DIST
static struct mutex file_lock   /* For locking table access */
    = MUTEX_INITIALIZER;
#endif
#ifdef LINUX_DIST
static pthread_mutex_t file_lock        /* For locking table access */
    = PTHREAD_MUTEX_INITIALIZER;
#endif
#define lock_file_table()   mutex_lock(&file_lock);
#define unlock_file_table() mutex_unlock(&file_lock);

#define isempty(i) (files[i].ffd == -1)
```

```
file_add(int ffd, int sfd)
{
    int i, pos;
#ifdef LINUX_DIST
    pthread_t pid;
#endif
    lock_file_table();

    if (files == NULL) {        /* The very first time */
        if ((files =
             (file_t *) malloc(10 * sizeof(file_t))) == NULL) {
            SRV_DEBUG(-3, "(%u) file_add cannot alloc table!\n",
                      cthread_self());
            unlock_file_table();
            return (-1);
        }

        for (i = 0; i < 10; i++)
            files[i].ffd = files[i].fid = -1;
        filenum = 10;
    }

    /* Search for empty spot */
    for (pos = 0; pos < filenum; pos++)
        if (isempty(pos))
            break;

    if (pos == filenum) {       /* No empty spot found - realloc 10 more
                                   entries */
        if ((files = (file_t *)
             realloc(files,
                     (filenum + 10) * sizeof(file_t))) == NULL) {
            SRV_DEBUG(-3,
                      "(%u) file_add cannot REalloc table (to size %d)!\n",
                      cthread_self(), filenum + 10);
            unlock_file_table();
            return (-2);        /* original table is left untouched! */
        }

        for (i = 1; i < 10; i++)
            files[filenum + i].ffd = files[filenum + i].fid = -1;
        filenum += 10;
    }

    files[pos].ffd = ffd;
    files[pos].sfd = sfd;
    files[pos].fid = -1;
    files[pos].fmem = NULL;

  /******Buffer initialization******/
    files[pos].buf =
        (file_buffer_t *) malloc(sizeof(file_buffer_t));

#ifdef HURD_DIST
#ifdef USE_MYCTHREAD
    file_buf_init(files[pos].buf, files[pos].ffd,
                  FILE_BUFFER_SIZE, cthread_spawn(readth,
                                                  (void *)
                                                  files[pos].
                                                  buf));
#else
    file_buf_init(files[pos].buf, files[pos].ffd,
                  FILE_BUFFER_SIZE, cthread_fork(readth,
                                                 (void *)
                                                 files[pos].
                                                 buf));
#endif
#endif
#ifdef LINUX_DIST
    pthread_create(&pid, NULL, readth, (void *) files[pos].buf);
```

```
    file_buf_init(files[pos].buf, files[pos].ffd,
                  FILE_BUFFER_SIZE, pid);
#endif
    set_file_buf_pos(files[pos].buf, BUF_CURPOS);

    unlock_file_table();
    return (pos);
}

file_del(int f)
{
    file_buffer_t *buf;

    lock_file_table();
    if (f < 0 || f >= filenum) {
        unlock_file_table();
        return (-1);
    }

    files[f].ffd = files[f].sfd = files[f].fid = -1;
    if (files[f].fmem != NULL)
        free(files[f].fmem);
    files[f].fmem = NULL;
    if ((buf = files[f].buf) != NULL) {
        file_buf_destroy(buf);  /* Finalize buffer */
        free(buf);
    }
    files[f].buf = NULL;
    unlock_file_table();
    return (0);
}

/* Delete any onclosed files opened by this particular client
 */
file_cleanup_owner(int sfd)
{
    int i;
    file_buffer_t *buf;

    lock_file_table();

    for (i = 0; i < filenum; i++)
        if (!isempty(i) && files[i].sfd == sfd) {
            SRV_DEBUG(-3,
                      "(%u) cleaning up file td %d owned by sfd %d\n",
                      cthread_self(), i, sfd);
            close(files[i].ffd);
            files[i].ffd = files[i].sfd = files[i].fid = -1;
            if (files[i].fmem != NULL)
                free(files[i].fmem);
            files[i].fmem = NULL;
            if ((buf = files[i].buf) != NULL) {
                file_buf_destroy(buf);  /* Finalize buffer */
                free(buf);
            }
        };

    unlock_file_table();
    return (0);
}

/* Check if ffd is a legal file id
 */
isfile(int ffd)
{
    int val = 0;

    lock_file_table();

    if (ffd >= 0 && ffd < filenum)
```

```
        if (!isempty(ffd))
            val = 1;
    unlock_file_table();
    return (val);
}

file_ffd(int f)
{
    int val;
    lock_file_table();
    if (f < 0 || f >= filenum) {
        unlock_file_table();
        return (-1);
    }
    val = files[f].ffd;
    unlock_file_table();
    return (val);
}

file_fid(int f)
{
    int val;
    lock_file_table();
    if (f < 0 || f >= filenum) {
        unlock_file_table();
        return (-1);
    }
    val = files[f].fid;
    unlock_file_table();
    return (val);
}

char *file_filtermem(int f)
{
    char *val = NULL;
    lock_file_table();
    if (f < 0 || f >= filenum) {
        unlock_file_table();
        return (NULL);
    }
    if (!isempty(f))
        val = files[f].fmem;
    unlock_file_table();
    return (val);
}

/* This sets the fid of the filter to be applied.
 * The fid MUST be owned by this client.
 */
int file_applyfilter(int f, int fid, int size)
{
    char *m;
    int client;
    if ((m = malloc(size)) == NULL) {
        SRV_DEBUG(-3,
                  "(%u) file_applyfilter cannot alloc fmem!\n",
                  cthread_self());
        return (-1);
    }
    lock_file_table();
```

```
    if (f < 0 || f >= filenum) {
        unlock_file_table();
        return (-1);
    }
    if (isempty(f)) {
        unlock_file_table();
        return (-1);
    }
    client = files[f].sfd;
    unlock_file_table();

    if (filterowner(fid) != client) {   /* Not owner! */
        SRV_DEBUG(-3, "(%u) client %d not owner of filter %d!\n",
                  cthread_self(), client, fid);
        return (-2);
    }

    lock_file_table();
    files[f].fid = fid;
    files[f].fmem = m;
    unlock_file_table();
    return (0);
}

void *file_getbuffer(int f)
{
    file_buffer_t *b;

    lock_file_table();
    if (f < 0 || f >= filenum) {
        unlock_file_table();
        return NULL;
    }
    if (isempty(f)) {
        unlock_file_table();
        return NULL;
    }
    b = files[f].buf;
    unlock_file_table();
    return (void *) b;
}
```

A.6 server/files.h

```
/* FILES.H
 * Interface to FILES.C
 */
#ifndef __FILES_H
#include "buffers.h"
extern int isfile(int /* file fd */ );
extern int file_add(int /* ffd */ , int /* sfd */ );
extern int file_del(int /* file fd */ );
extern int file_cleanup_owner(int /* client's sfd */ );

extern int file_ffd(int);
extern int file_fid(int);
extern int file_applyfilter(int, int /* fid */ ,
                            int /* size */ );

extern char *file_filtermem(int);

extern void *file_getbuffer(int);

#define __FILES_H
#endif
```

A.7 server/filters.c

```
/* FILTERS.C
 * Code for storing and using filters registered by the user
 *
 * Revisions:
 * ==========
 * AFTS 0.2.1 06/06/2000
 *    Added "owner" field to hold the client (sfd) who registered the
 *    filter; added filter_del_owner() to delete all the filters owned
 *    by a client. filter_del() should be considered obsolete.
 *
 * AFTS 0.2 05/06/2000
 *    Completely rewritten by VVD, CS, UoI
 *
 * AFTS 0.1 ??/06/2000
 *    Untouched
 */
/* July-August/2001
Buffer utility added
 */
#if 0                           /* The following aren't necessary to files
                                   that include "buffers.h" */
#ifdef HURD_DIST
#ifdef USE_MYCTHREADS
#include "../libthreads/cthreads.h"     /* Use relative path to avoid
                                           conflicts */
#include "../libthreads/cthread_internals.h"
#else
#include <cthreads.h>
#endif
#endif
#ifdef LINUX_DIST
#include <pthread.h>
#define cthread_self() pthread_self()
#define mutex_lock(t) pthread_mutex_lock(t)
#define mutex_unlock(t) pthread_mutex_unlock(t)
#endif
#endif

#include <stdio.h>
#include <dlfcn.h>
#include "buffers.h"
#include "smasdefs.h"
#include "server.h"
#include "filters.h"


/* The filter's three parts:
 */
typedef void (*initfunc_t) (void *);
typedef int (*bodyfunc_t) (void *, int *, void **);
typedef int (*tinifunc_t) (int *, void **);

/* The basic structure
 */
typedef struct {
    initfunc_t finit;           /* The init() function */
    bodyfunc_t fbody;           /* The body() function */
    tinifunc_t ftini;           /* The tini() function */
    int memsize;                /* Memory needed by the filter */
    int readsize;               /* # bytes to read per file record
                                   (dynamic) */
    void *code;                 /* The actual code */
    char fname[MAX_PNAME];      /* Obvious */
    int owner;                  /* Who registered this filter??? (socket
                                   fd) */
    filter_buffer_t *buf;       /* The buffer this filter outputs to */
} filter_t;

/* The table of registered filters
 */
```

```
static filter_t *regfilters = NULL;     /* Table of registered filters */
static int regfilnum = 0;       /* Number of registered filters */
#ifdef HURD_DIST
static struct mutex regfil_lock /* For locking table access */
    = MUTEX_INITIALIZER;
#endif
#ifdef LINUX_DIST
static pthread_mutex_t regfil_lock      /* For locking table access */
    = PTHREAD_MUTEX_INITIALIZER;
#endif
#define lock_regfil_table()   mutex_lock(&regfil_lock);
#define unlock_regfil_table() mutex_unlock(&regfil_lock);

#define isempty(i) (regfilters[i].code == NULL)

/* Check if fid is a legal filter id
 */
isregfilter(int fid)
{
    int val = 0;

    lock_regfil_table();

    if (fid >= 0 && fid < regfilnum)
        if (!isempty(fid))
            val = 1;

    unlock_regfil_table();
    return (val);
}

/* Return the owner of the filter
 */
filterowner(int fid)
{
    int val = -1;

    lock_regfil_table();

    if (fid >= 0 && fid < regfilnum)
        if (!isempty(fid))
            val = regfilters[fid].owner;

    unlock_regfil_table();
    return (val);
}

filter_load(char *fname, char *fullpath, filter_t * flt)
{
    void *dlh;
    int *var;
    char symbol[MAX_FNAME + 10];

  /************/
    static int isinit = 0;

    /* dlopen() the file - be careful about the way dlopen() searches for
       files! */
    /* A workaround to a strange behaviour, dlopenning a filter but never
       dlclosing it seems to supress a bus error. */
    if (!isinit) {
        dlh =
            dlopen("/usr/local/SmAS/filters/libselect10.so",
                   RTLD_NOW);
        if (dlh == NULL)
            fprintf(stderr, "Error opening initial filter");
        isinit = 1;
    }
    if ((dlh = dlopen(fullpath, RTLD_LAZY)) == NULL) {
        SRV_DEBUG(-2, "(%u) cannot dlopen() the filter!\n",
```

```
                cthread_self());
        return (-1);
    }
    flt->code = dlh;

    /* Now find the needed symbols */
    sprintf(symbol, "%s_init", fname);
    if ((flt->finit = (initfunc_t) dlsym(dlh, symbol)) == NULL) {
        SRV_DEBUG(-2, "(%u) no %s function in the filter!\n",
                  cthread_self(), symbol);
        dlclose(dlh);
        return (-2);
    }

    sprintf(symbol, "%s_body", fname);
    if ((flt->fbody = (bodyfunc_t) dlsym(dlh, symbol)) == NULL) {
        SRV_DEBUG(-2, "(%u) no %s function in the filter!\n",
                  cthread_self(), symbol);
        dlclose(dlh);
        return (-2);
    }

    sprintf(symbol, "%s_tini", fname);
    if ((flt->ftini = (tinifunc_t) dlsym(dlh, symbol)) == NULL) {
        SRV_DEBUG(-2, "(%u) no %s function in the filter!\n",
                  cthread_self(), symbol);
        dlclose(dlh);
        return (-2);
    }

    sprintf(symbol, "%s_memsize", fname);
    if ((var = (int *) dlsym(dlh, symbol)) == NULL) {
        SRV_DEBUG(-2, "(%u) no %s variable in the filter!\n",
                  cthread_self(), symbol);
        dlclose(dlh);
        return (-2);
    }
    flt->memsize = *var;

    sprintf(symbol, "%s_readsize", fname);
    if ((var = (int *) dlsym(dlh, symbol)) == NULL) {
        SRV_DEBUG(-2, "(%u) no %s variable in the filter!\n",
                  cthread_self(), symbol);
        dlclose(dlh);
        return (-2);
    }
    flt->readsize = *var;

    strcpy(flt->fname, fullpath);
    return (0);
}

filter_add(char *fname, char *fullpath, int owner)
{
    filter_t flt;
    int i, pos;

    if ((i = filter_load(fname, fullpath, &flt)) < 0)
        return (i);

    /* Now do the table stuff */
    lock_regfil_table();

    if (regfilters == NULL) {   /* The very first time */
        if ((regfilters =
             (filter_t *) malloc(10 * sizeof(filter_t))) ==
            NULL) {
            dlclose(flt.code);
            unlock_regfil_table();
```

```
            return (-3);
        }
        for (i = 0; i < 10; i++)
            regfilters[i].code = NULL;
        regfilnum = 10;
    }
    /* Search for empty spot */
    for (pos = 0; pos < regfilnum; pos++)
        if (isempty(pos))
            break;
    if (pos == regfilnum) {     /* No empty spot found - realloc 10 more
                                   entries */
        if ((regfilters = (filter_t *)
             realloc(regfilters,
                     (regfilnum + 10) * sizeof(filter_t))) ==
            NULL) {
            dlclose(flt.code);
            unlock_regfil_table();
            return (-3);        /* original table is left untouched! */
        }
        for (i = 1; i < 10; i++)
            regfilters[regfilnum + i].code = NULL;
        regfilnum += 10;
    }
    regfilters[pos] = flt;
    regfilters[pos].owner = owner;
    regfilters[pos].buf = (filter_buffer_t *) malloc(sizeof(filter_buffer_t));  /* Must
                                                                                   initialize
                                                                                   before
                                                                                   applying
                                                                                 */
    unlock_regfil_table();
    return (pos);
}

/* ------> We should UNLINK the filter file - but what if the same
 * ------> filter is used by other clients????
 */
filter_del(int fid)
{
    filter_buffer_t *buf;
    lock_regfil_table();
    if (fid < 0 || fid >= regfilnum) {
        unlock_regfil_table();
        return (-1);
    }
    if (!isempty(fid)) {
        buf = (filter_buffer_t *) filter_getbuffer(fid);
        filter_buf_destroy(buf);        /* Finalize buffer - Make sure
                                           that it's source file buffer is
                                           finalized first */
        free(buf);
        dlclose(regfilters[fid].code);
    }
    regfilters[fid].code = NULL;
    unlock_regfil_table();
    return (0);
}

/* Delete ALL filters owned by the given owner
```

```
 * BE CAREFULL TO CHECK FOR .code != NULL !!!!
 */
filter_del_owner(int owner)
{
    int i;
    filter_buffer_t *buf;

    lock_regfil_table();

    for (i = 0; i < regfilnum; i++)
        if (!isempty(i) && regfilters[i].owner == owner) {
            SRV_DEBUG(-3,
                      "(%u) removing filter %d owned by sfd %d\n",
                      cthread_self(), i, owner);

            buf = regfilters[i].buf;
            filter_buf_destroy(buf);    /* Finalize buffer - Make sure
                                           that it's source file buffer is
                                           finalized first */
            free(buf);
            dlclose(regfilters[i].code);
            regfilters[i].code = NULL;
            break;
        }

    unlock_regfil_table();
}

filter_init(int fid, void *rec)
{
    initfunc_t init;

    lock_regfil_table();
    if (fid < 0 || fid >= regfilnum) {
        unlock_regfil_table();
        return (-1);
    }

    init = regfilters[fid].finit;

    unlock_regfil_table();

    init(rec);
    return (0);
}

/* The filter's body execution is NOT seriallized :-)
 * i.e. it is called *after* unlocking the table.
 *
 * FILTER BODIES SHOULD RETURN: 0 if record does not pass
 *                             1 if it passes and should be fowrarded
 */
filter_body(int fid, void *rec, int *n, void **out)
{
    int t;
    bodyfunc_t body;

    lock_regfil_table();
    if (fid < 0 || fid >= regfilnum) {
        unlock_regfil_table();
        return (-1);
    }

    body = regfilters[fid].fbody;

    unlock_regfil_table();

    return (body(rec, n, out));
}

/* Like body, it SHOULD RETURN: 0 if record does not pass
 *                             1 if it passes and should be fowrarded
```

```
 */
filter_tini(int fid, int *n, void **out)
{
    int t;
    tinifunc_t tini;

    lock_regfil_table();
    if (fid < 0 || fid >= regfilnum) {
        unlock_regfil_table();
        return (-1);
    }

    tini = regfilters[fid].ftini;

    unlock_regfil_table();

    return (tini(n, out));
}

filter_memsize(int fid)
{
    int t;

    lock_regfil_table();
    if (fid < 0 || fid >= regfilnum) {
        unlock_regfil_table();
        return (-1);
    }

    t = regfilters[fid].memsize;

    unlock_regfil_table();
    return (t);
}

filter_readsize(int fid)
{
    int t;

    lock_regfil_table();
    if (fid < 0 || fid >= regfilnum) {
        unlock_regfil_table();
        return (-1);
    }

    t = regfilters[fid].readsize;

    unlock_regfil_table();
    return (t);
}

void *filter_getbuffer(int fid)
{
    filter_buffer_t *t;

    lock_regfil_table();
    if (fid < 0 || fid >= regfilnum) {
        unlock_regfil_table();
        return NULL;
    }

    t = regfilters[fid].buf;

    unlock_regfil_table();
    return ((void *) (t));
}
```

A.8 server/filters.h

```
/* FILTERS.H
```

```
 * Interface to FILTERS.C
 */
#ifndef __FILTERS_H
#include "buffers.h"
/* Info functions
 */
extern int isregfilter(int /* filter id */ );
extern int filterowner(int /* filter id */ );

/* Bookkeeping functions
 */
extern int filter_add(char * /* name */ , char * /* fullpath */ ,
                      int /* owner sfd */ );
extern int filter_del(int /* fid */ );
extern int filter_del_owner(int /* owner id */ );

/* Filter usage function
 */
extern int filter_init(int /* fid */ , void * /* filter mem */ );
extern int filter_body(int /* fid */ , void * /* inrec */ ,
                       int * /* &outsize */ ,
                       void ** /* &outbufptr */ );
extern int filter_tini(int /* fid */ ,
                       int * /* &outsize */ ,
                       void ** /* &outbufptr */ );
extern int filter_memsize(int /* fid */ );
extern int filter_readsize(int /* fid */ );

extern void *filter_getbuffer(int);

#define __FILTERS_H
#endif
```

A.9 server/handlers.c

```
/* HANDLERS.C
 * Message Handlers -- here we service all the SMAS_xxx messages
 * (client requests).
 *
 * Revisions:
 * ==========
 * 0.2.3, 06/12/2000
 *    Code to support general installations (single-user or system-wide)
 *    Modified handle_smas_open() to use smas_root_files[] and
 *    handle_smas_registerfilter() to use smas_root_filters[].
 *
 * AFTS 0.2.2 14/06/2000
 *    Fixed smas_close bug on illegal fd; utilized sock_writev().
 *
 * AFTS 0.2.1 06/06/2000
 *    Slightly modified for new owener-cleanup stuff
 *
 * AFTS 0.2 05/06/2000
 *    Written by VVD, CS, UoI
 *
 * AFTS 0.1 ??/06/2000
 *    Non-existant
 */
#include <stdio.h>
#include <stdlib.h>             /* for malloc() */
#include <sys/ioctl.h>
#include <unistd.h>
#include <fcntl.h>
#include <dlfcn.h>
#include "buffers.h"
#include "smasdefs.h"
#include "socks.h"
#include "handlers.h"
#include "server.h"
#include "filters.h"
#include "files.h"
```

```
#include "timer.h"

extern void *nextrecth(void *);

/*Uniform open() args */
/*We need this because these flags may be defined with different values across platforms*/
/*Not all values from  <bits/fcnl.h> are defined here, only those used by open()*/

/*

These are defined in ../lib/smas.h as they are used only by the client, displayed here for convinience

#define SMAS_O_RDONLY 1
#define SMAS_O_WRONLY  2
#define SMAS_O_RDWR  4
#define SMAS_O_CREAT  8
#define SMAS_O_EXCL  16
#define SMAS_O_NOCTTY   32
#define SMAS_O_TRUNC                64
#define SMAS_O_APPEND   128
#define SMAS_O_NONBLOCK   256
#define SMAS_O_SYNC   512
*/

/*We use the above values to index the array below*/
unsigned long int open_flags[] = {
    O_RDONLY,
    O_WRONLY,
    O_RDWR,
    O_CREAT,
    O_EXCL,
    O_NOCTTY,
    O_TRUNC,
    O_APPEND,
    O_NONBLOCK,
    O_SYNC
};
/*Convert SmAS defined flags from x to system defined flags in y*/
/*WARNING : Always use different variables for x and y*/
#define GET_OFLAGS(x,y) do { \
int i;   \
\
(y)=0; \
    for (i=0;i<10;i++) { \
if ((x)&(1<<i))  \
(y)|=open_flags[i]; \
                                                    } \
} while(0)

/* The table of handler functions -- one for each message type
 */
#define ARGS int

extern void handle_smas_open(ARGS), handle_smas_close(ARGS),
handle_smas_lseek(ARGS), handle_smas_applyfilter(ARGS),
handle_smas_read(ARGS), handle_smas_write(ARGS),
handle_smas_nextrec(ARGS), handle_smas_registerfilter(ARGS);
void (*msghandler[10]) (ARGS);

handlers_init()
{
    msghandler[SMAS_OPEN] = handle_smas_open;
    msghandler[SMAS_CLOSE] = handle_smas_close;
    msghandler[SMAS_READ] = handle_smas_read;
    msghandler[SMAS_WRITE] = handle_smas_write;
    msghandler[SMAS_LSEEK] = handle_smas_lseek;
    msghandler[SMAS_REGFIL] = handle_smas_registerfilter;
    msghandler[SMAS_APPFIL] = handle_smas_applyfilter;
    msghandler[SMAS_NEXTREC] = handle_smas_nextrec;
}
```

```
/* Returns:  0 if ok
 *          -1 on memory allocation failure
 *          -2 on unknown message reception
 */
/*Should go in a header file*/
handle_message(int sfd, int msgtype)
{
    if (msgtype < SMAS_OPEN || msgtype > SMAS_NEXTREC)
        return (-2);

    (msghandler[msgtype]) (sfd);
    return (0);
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                     *
 *       THE HANDLER FUNCTIONS                                         *
 *                                                                     *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */


void handle_smas_open(int sfd)
{
    int fd, td;
    msgopen_t m;
    char fullpath[500];

    if (sock_read(sfd, (char *) &m, sizeof(m)) < 1)
        cthread_exit(NULL);
    m.flags = ntohl(m.flags);
    m.mode = ntohl(m.mode);

    SRV_DEBUG(1,
              "(%u) received OPEN message (file: %s, flags: %o)\n",
              cthread_self(), m.fname, m.flags);

    if (strlen(smas_root_files) + strlen(m.fname) > 500)
        goto FERROR;
    strcat(strcpy(fullpath, smas_root_files), m.fname);
    GET_OFLAGS(m.flags, td);
    m.flags = td;
    if ((fd = open(fullpath, m.flags, m.mode)) < 0) {
      FERROR:

        SRV_DEBUG(-2, "(%u) cannot open the file!\n",
                  cthread_self());
        fd = htonl(fd);
        if (sock_write(sfd, (char *) &fd, sizeof(fd)) < 1)
            cthread_exit(NULL);
        return;
    }
    if ((td = file_add(fd, sfd)) < 0)
        close(fd);
    td = htonl(td);
    if (sock_write(sfd, (char *) &td, sizeof(td)) < 1)
        cthread_exit(NULL);

    td = ntohl(td);
    SRV_DEBUG((td < 0) ? -2 : 2,
              "(%u) file opened (fd = %d, td = %d), info sent to client\n",
              cthread_self(), fd, td);
}

void handle_smas_close(int sfd)
{
    int value;
    msgclose_t m;

    if (sock_read(sfd, (char *) &m, sizeof(m)) < 1)
```

```
        cthread_exit(NULL);
    m.fd = ntohl(m.fd);

    SRV_DEBUG(1, "(%u) received CLOSE message for td %d\n",
              cthread_self(), m.fd);

    if (!isfile(m.fd)) {
        SRV_DEBUG(-2, "(%u) invalid file to close\n",
                  cthread_self());
        value = htonl(-1);
        if (sock_write(sfd, (char *) &value, sizeof(value)) < 1)
            cthread_exit(NULL);
        return;
    }

    /* Notice that we DON'T DELETE THE FILTERS applied to this file.
       Filter deletion (for all registered filters) is deferred to the
       time the client departs - see service(). */
    value = htonl(close(file_ffd(m.fd)));
    file_del(m.fd);
    if (sock_write(sfd, (char *) &value, sizeof(value)) < 1)
        cthread_exit(NULL);

    SRV_DEBUG(2, "(%u) file closed; info sent to client\n",
              cthread_self());
}

void handle_smas_read(int sfd)
{
    int fid = 0, value = 0, nv = 0;
    msgread_t m;
    char *buf = NULL;
    file_buffer_t *fbuffer;
    filter_buffer_t *flbuffer = NULL;

    if (sock_read(sfd, (char *) &m, sizeof(m)) < 1)
        cthread_exit(NULL);
    m.fd = ntohl(m.fd);
    m.nbytes = ntohl(m.nbytes);

    SRV_DEBUG(1,
              "(%u) received READ message for td %d (%d bytes)\n",
              cthread_self(), m.fd, m.nbytes);

    if (m.nbytes <= 0) {
        SRV_DEBUG(-2, "(%u) illegal number of bytes requested\n",
                  cthread_self());
        nv = htonl(value = -10);
        m.nbytes = 0;
    } else {
        if ((buf = calloc(m.nbytes, sizeof(char))) == NULL) {
            SRV_DEBUG(-2, "(%u) cannot allocate read buffer\n",
                      cthread_self());
            nv = htonl(value = -20);
            m.nbytes = 0;
        } else {
            if ((fid = file_fid(m.fd)) > -1) {
                flbuffer =
                    (filter_buffer_t *) filter_getbuffer(fid);
                mutex_lock(&flbuffer->stop);
                filter_buf_invalidate(flbuffer);        /* Stop filter
                                                           buffer's fill
                                                           thread */
            }
            fbuffer = (file_buffer_t *) file_getbuffer(m.fd);
            value = file_ffd(m.fd);
            if (!file_buf_readvalid(fbuffer))
                file_buf_invalidate(fbuffer);   /* Invalidate buffer */
            nv = htonl(value =
                       file_buf_read(fbuffer, buf, m.nbytes));
```

```
            file_buf_validate(fbuffer);
            if (flbuffer != NULL) {
                filter_buf_invalidate(flbuffer);
                mutex_unlock(&flbuffer->stop);  /* Resume filter buffer's
                                                   fill thread */
            }
        }
    }
    CHECK_ERROR(value);
    if (value <= 0) {
        if (sock_write(sfd, (char *) &nv, sizeof(nv)) < 1)
            cthread_exit(NULL);
    } else {                    /* i.e. read "value" bytes " */
        if (sock_write(sfd, (char *) &nv, sizeof(nv)) < 1)
            cthread_exit(NULL);
        if (sock_write(sfd, (char *) buf, value) < 1)
            cthread_exit(NULL);
    }
    if (buf != NULL)
        free(buf);
    SRV_DEBUG((value < m.nbytes) ? -2 : 2,
              "(%u) read done (retval = %d); info sent to clent\n",
              cthread_self(), value);
}

void handle_smas_write(int sfd)
{
    int value;
    char *buf;
    msgwrite_t m;
    file_buffer_t *fbuffer;
    filter_buffer_t *flbuffer;
    if (sock_read(sfd, (char *) &m, sizeof(m)) < 1)
        cthread_exit(NULL);
    m.fd = ntohl(m.fd);
    m.nbytes = ntohl(m.nbytes);
    SRV_DEBUG(1,
              "(%u) received WRITE message for td %d (%d bytes)\n",
              cthread_self(), m.fd, m.nbytes);
    if (m.nbytes <= 0)
        value = htonl(-5);
    else if ((buf = malloc(m.nbytes)) == NULL) {
        sock_flush(sfd, m.nbytes);
        value = htonl(-6);
    } else {
        fbuffer = (file_buffer_t *) file_getbuffer(m.fd);
        flbuffer =
            (filter_buffer_t *) filter_getbuffer(file_fid(m.fd));
        if (flbuffer != NULL) { /* Stop filter buffer's fill thread */
            mutex_lock(&flbuffer->stop);
            filter_buf_invalidate(flbuffer);
        }
        mutex_lock(&fbuffer->stop);     /* Stop file buffer's fill thread */
        file_buf_invalidate(fbuffer);
        if (sock_read(sfd, buf, m.nbytes) < 1)
            cthread_exit(NULL);
        value = htonl(write(file_ffd(m.fd), buf, m.nbytes));
        mutex_unlock(&fbuffer->stop);   /* Resume file buffer's fill
                                           thread */
        if (flbuffer != NULL) {
            filter_buf_invalidate(flbuffer);
            mutex_unlock(&flbuffer->stop);      /* Resume filter buffer's
                                                   fill thread */
        }
```

```
    }
    if (sock_write(sfd, (char *) &value, sizeof(value)) < 1)
        cthread_exit(NULL);
    value = ntohl(value);
    SRV_DEBUG((value < 0) ? -2 : 2,
              "(%u) write done (retval = %d); info sent to clent\n",
              cthread_self(), value);
}

void handle_smas_lseek(int sfd)
{
    int value;
    msglseek_t m;
    file_buffer_t *fbuffer;
    filter_buffer_t *flbuffer;
    if (sock_read(sfd, (char *) &m, sizeof(m)) < 1)
        cthread_exit(NULL);
    m.fd = ntohl(m.fd);
    m.offset = ntohl(m.offset);
    m.whence = ntohl(m.whence);
    SRV_DEBUG(1,
              "(%u) received LSEEK message for td %d (offset %d, whence %d)\n",
              cthread_self(), m.fd, m.offset, m.whence);
    fbuffer = (file_buffer_t *) file_getbuffer(m.fd);
    flbuffer =
        (filter_buffer_t *) filter_getbuffer(file_fid(m.fd));
    if (flbuffer != NULL) {     /* Stop filter buffer's fill thread */
        mutex_lock(&flbuffer->stop);
        filter_buf_invalidate(flbuffer);
    }
    mutex_lock(&fbuffer->stop); /* Stop file buffer's fill thread */
    file_buf_invalidate(fbuffer);
    value = htonl(lseek(file_ffd(m.fd), m.offset, m.whence));
    mutex_unlock(&fbuffer->stop);       /* Resume file buffer's fill
                                           thread */
    if (flbuffer != NULL) {
        filter_buf_invalidate(flbuffer);
        mutex_unlock(&flbuffer->stop);  /* Resume filter buffer's fill
                                           thread */
    }
    write(sfd, &value, sizeof(value));
    SRV_DEBUG(2,
              "(%u) lseek done (retval = %d); info sent to clent\n",
              cthread_self(), ntohl(value));
}

void handle_smas_registerfilter(int sfd)
{
    int nfd, val;
    msgregfil_t m;
    char *buf, fullpath[MAX_PNAME];

    if (sock_read(sfd, (char *) &m, sizeof(m)) < 1)
        cthread_exit(NULL);
    m.codesize = ntohl(m.codesize);
    SRV_DEBUG(1, "(%u) received REGFIL message (%s, %d bytes)\n",
              cthread_self(), m.fname, m.codesize);
    if (strlen(smas_root_filters) + strlen(m.fname) > 500)
        goto FERROR;
    strcat(strcpy(fullpath, smas_root_filters), m.fname);
    /* Open file for saving the code */
```

```
    if ((nfd = open(fullpath, O_CREAT | O_WRONLY, 0755)) < 0) {
      FERROR:
        SRV_DEBUG(-2,
                  "(%u) cannot open file for saving filter\n",
                  cthread_self());
        sock_flush(sfd, m.codesize);    /* Empty it */
        val = htonl(nfd);
        if (sock_write(sfd, (char *) &val, sizeof(val)) < 1)
            cthread_exit(NULL);
        return;
    }

    /* Allocate a large enough buffer and read the code */
    if ((buf = (char *) malloc(m.codesize)) == NULL) {
        SRV_DEBUG(-2, "(%u) cannot malloc() for filter code\n",
                  cthread_self());
        val = htonl(-5);
        sock_flush(sfd, m.codesize);
        close(nfd);
        if (sock_write(sfd, (char *) &val, sizeof(val)) < 1)
            cthread_exit(NULL);
        return;
    }
    if (sock_read(sfd, buf, m.codesize) < 1)
        cthread_exit(NULL);

    /* Save the code, release buf and close the file */
    val = write(nfd, buf, m.codesize);
    close(nfd);
    free(buf);

    if (val < 0) {
        SRV_DEBUG(-2, "(%u) could not write filter to file\n",
                  cthread_self());
        val = htonl(val);
        if (sock_write(sfd, (char *) &val, sizeof(val)) < 1)
            cthread_exit(NULL);
        return;
    }
    SRV_DEBUG(2, "(%u) filter written to file %s\n",
              cthread_self(), fullpath);
    val = htonl(filter_add(m.fname, fullpath, sfd));
    if (sock_write(sfd, (char *) &val, sizeof(val)) < 1)
        cthread_exit(NULL);

    val = ntohl(val);
    SRV_DEBUG((val < 0) ? -2 : 2,
              "(%u) filter registered (fid: %d, owner: %d)\n",
              cthread_self(), val, sfd);
}

void handle_smas_applyfilter(int sfd)
{
    int memsize, value;
    msgappfil_t m;

    if (sock_read(sfd, (char *) &m, sizeof(m)) < 1)
        cthread_exit(NULL);
    m.fd = ntohl(m.fd);
    m.fid = ntohl(m.fid);

    SRV_DEBUG(1,
              "(%u) received APPFIL message (filter id %d for td %d)\n",
              cthread_self(), m.fid, m.fd);

    if (!isregfilter(m.fid)) {
        value = htonl(-1);
        SRV_DEBUG(-2, "(%u) filter id %d invalid\n",
                  cthread_self(), m.fid);
```

```
    } else {
      /******Buffer initialization******/
        file_buffer_t *fbuf =
            (file_buffer_t *) file_getbuffer(m.fid);
        filter_buffer_t *buf =
            (filter_buffer_t *) filter_getbuffer(m.fid);
        memsize = filter_readsize(m.fid);
        mutex_lock(&fbuf->stop);        /* This will save some trouble */
        filter_buf_init(buf, (int) file_getbuffer(m.fd),
                        FILTER_BUFFER_SIZE, memsize);
        set_filter_buf_pos(buf, BUF_NOPOS);
        mutex_lock(&buf->stop);
#ifdef HURD_DIST
#ifdef USE_MYCTHREAD
        buf->fillth = cthread_spawn(nextrecth, (void *) m.fid);
#else
        buf->fillth = cthread_fork(nextrecth, (void *) m.fid);
#endif
#endif
#ifdef LINUX_DIST
        pthread_create(&buf->fillth, NULL, nextrecth,
                       (void *) m.fid);
#endif
        mutex_unlock(&fbuf->stop);
        if ((value =
             file_applyfilter(m.fd, m.fid, memsize)) == 0)
            filter_init(m.fid, (void *) file_filtermem(m.fd));
        value = htonl(value);
        mutex_unlock(&buf->stop);
    }

    if (sock_write(sfd, (char *) &value, sizeof(value)) < 1)
        cthread_exit(NULL);

    value = ntohl(value);
    SRV_DEBUG((value < 0) ? -2 : 2,
              "(%u) filter %d set and initialized (%d)\n",
              cthread_self(), m.fid, value);
}

void handle_smas_nextrec(int sfd)
{
    msgnextrec_t m;
    filter_buffer_t *buf;
    char *recbuf;
    void *MyRam;
    int fid, val, iseof = 0, recsize, size, nbytes;
    struct iovec vec[2];
    struct {
        int val;
        int nbytes;
    } reply;

    if (sock_read(sfd, (char *) &m, sizeof(m)) < 1)
        cthread_exit(NULL);

    CHECK_ERROR(m.fd = ntohl(m.fd));
    fid = file_fid(m.fd);
    size = filter_readsize(fid);
    buf = (filter_buffer_t *) filter_getbuffer(fid);
    SRV_DEBUG(1,
              "(%u) received NEXTREC message (td %d, filter id %d)\n",
              cthread_self(), m.fd, fid);

    recbuf = (size > 0) ? malloc(size) : NULL;
    recsize = filter_buf_nextrec(buf, recbuf);
    mutex_lock(&buf->lock);
    if (buf->status & BUF_SEOF && recsize == 0) {
        MyRam = (void *) file_filtermem(m.fd);
        filter_tini(fid, &recsize, &MyRam);
```

```
        iseof = 1;
        mutex_unlock(&buf->lock);
    } else {
        mutex_unlock(&buf->lock);
        MyRam = (void *) recbuf;
        filter_buf_validate(buf);
    }

    reply.val = htonl(!iseof);  /* 0 = LASTREC, 1 = continue */
    reply.nbytes = htonl(recsize);
    if (sock_write(sfd, (char *) &reply, sizeof(reply)) < 0)
        cthread_exit(NULL);
    if (sock_write(sfd, (char *) MyRam, recsize) < 0)
        cthread_exit(NULL);

    SRV_DEBUG(2,
              "(%u) rec sent to client (flag = %d, size = %d)\n",
              cthread_self(), val, recsize);

    if (iseof == 0 && recbuf != NULL)
        free(recbuf);
    return;
}
```

A.10 server/handlers.h

```
/* HANDLERS.H
 * Interface to HANDLERS.C
 */

extern handlers_init();

/* Returns:  0 if ok, -1 on memalloc failure, -2 if unknown message
 */
extern handle_message(int /* sfd */ , int /* msgsize */ );
```

A.11 server/server.c

```
/* SERVER.C
 * The Server (SmaS Disk)
 *
 * Revisions:
 * ==========
 * 0.2.4, 18/12/2000
 *    Now catches SIGTERM, too.
 *
 * 0.2.3, 06/12/2000
 *    Added code to allow system-wide installation for SmAS
 *    server (filesystem_init()), plus allow a couple of
 *    command line options.
 *    A lock file is now used to allow only 1 server to run.
 *    Also, now the server runs (by itself) on the background.
 *
 * AFTS 0.2.1 06/06/2000
 *    No MAX_CLIENTS limmitation. No bookeeping for service
 *    threads! new_server_thread() and done_server_thread()
 *    redefined and simplified.
 *
 * AFTS 0.2 05/06/2000
 *    Improved threaded code & debugging info
 *
 * AFTS 0.1 ??/06/2000
 *    Completely rewritten by VVD, CS, UoI
 */
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
```

```
#include <unistd.h>
#include <signal.h>
/*We try to use a unified version for all possible platforms that SmAS is used*/
#ifdef HURD_DIST
#ifdef USE_MYCTHREADS
#include "../libthreads/cthreads.h"     /* Use relative path to avoid
                                           conflicts */
#include "../libthreads/cthread_internals.h"
#else
#include <cthreads.h>
#endif
#endif
#ifdef LINUX_DIST
#include <pthread.h>
#define cthread_self() pthread_self()
#define mutex_lock(t) pthread_mutex_lock(t)
#define mutex_unlock(t) pthread_mutex_unlock(t)
#endif
#include <stdarg.h>
#include "socks.c"
#include "smasdefs.h"
#include "server.h"
#include "handlers.h"
#include "files.h"
#include "filters.h"
#include "timer.h"
/*From buffers.c*/
extern int FILTER_BUFFER_SIZE;
extern int FILE_BUFFER_SIZE;
int smas_server_sfd;
char smas_root[500];
char smas_root_filters[500];
char smas_root_files[500];

void (*oldsigint_handler) (int);
void (*oldsigterm_handler) (int);


void onsigintterm(int sig)
{
    close(smas_server_sfd);
    signal(SIGINT, oldsigint_handler);  /* Restore old signal */
    signal(SIGINT, oldsigterm_handler); /* Restore old signal */
    exit_smas(0, ">>> SmAS shut down completed <<<\n");
    return;
}

filesystem_init()
{
    int installing = 0;
    if (*smas_root)
        if (!isdir(smas_root)) {
            installing = 1;
            if (mkdir(smas_root, 0755) != 0)
                exit_smas(1,
                          "[SmAS_server]: cannot install system directories\n");
        };
    sprintf(smas_root_files, "%s/files", smas_root);
    sprintf(smas_root_filters, "%s/filters", smas_root);
    if (!isdir(smas_root_files)) {
        installing = 1;
        if (mkdir(smas_root_files, 0755) != 0)
            exit_smas(1,
                      "[SmAS_server]: cannot install userfile directories\n");
    }
```

```
    if (!isdir(smas_root_filters)) {
        installing = 1;
        if (mkdir(smas_root_filters, 0755) != 0)
            exit_smas(1,
                      "[SmAS_server]: cannot install filter directories\n");
    }
    if (installing)
        SRV_DEBUG(0,
                  ">>> SmAS .. installation of file systems complete <<<\n");
    strcat(smas_root_files, "/");
    strcat(smas_root_filters, "/");
}

int main(int argc, char *argv[])
{
    fd_set rfds, fdlist;
    int sfd, len, i;
    struct sockaddr_in claddr;
    extern void *service(void *);

    /* Go to the background! */
#ifdef LINUX_DIST
    if (fork() != 0)
        exit(0);
#endif
#ifdef HURD_DIST
    /* Initialize threads */
    cthread_init();
#ifdef USE_MYCTHREAD
    set_sched(SMAS_SCHED_ROUNDROBIN, (any_t) 150);
#endif
#endif
    /* No '/' at the end of directory name - makes mkdir() to fail under
       HURD, propably a libc issue */
    strcpy(smas_root, "/usr/local/SmAS");

    lockfile();

#if 0                           /* A better parameter checking is done
                                   bellow */
    if (argc > 1) {
        for (i = 1; i < argc; i++)
            if (strcmp(argv[i], "-s") == 0)
                *smas_root = 0;
            else if (strcmp(argv[i], "-r") == 0) {
                i++;
                if (i >= argc)
                    exit_smas(1,
                              "[SmAS Server]: usage: SmAS_server [-s] [-r <path>]\n");
                strncpy(smas_root, argv[i], 480);       /* We add stuff
                                                           later ... */

                /* Add the trailing "/" */

                if (strlen(smas_root) > 1)
                    if (smas_root[strlen(smas_root) - 1] != '/')
                        strcat(smas_root, "/");
            } else
                exit_smas(1,
                          "[SmAS Server]: usage: SmAS_server [-s] [-r <path>]\n");
    }
#endif

    while ((i = getopt(argc, argv, "sr:F:R:")) != -1) {
        char c = (char) i;
        if (c == 's')
            *smas_root = 0;
        if (c == 'r')
```

```
            strncpy(smas_root, optarg, 480);    /* We add stuff later ... */
        if (c == 'R') {
            FILTER_BUFFER_SIZE = atoi(optarg);
            if (FILTER_BUFFER_SIZE < 1
                || FILTER_BUFFER_SIZE > 1000)
                c = '?';
        }
        if (c == 'F') {
            FILE_BUFFER_SIZE = atoi(optarg);
            if (FILE_BUFFER_SIZE < 1
                || FILE_BUFFER_SIZE > 10000000)
                c = '?';
        }
        if (c == '?' || c == ':') {
            exit_smas(1,
                      "[SmAS Server]: usage: SmAS_server [-s] [-r <path>] [-F <file buffer size>] [-R <record buffer size>]\n");
        }
    }

    filesystem_init();
    handlers_init();

    /* Start socket service */
    if ((smas_server_sfd =
         server_open_socket(SERVER_PORT, 5)) < 0)
        exit_smas(1,
                  "[SmAS Server]: cannot open main socket (%d)\n",
                  smas_server_sfd);

    /* Intsall signals: (a) for graceful interruption and (b) for when we
       try to write to a socket that is not connected (e.g. the client
       died prematurely) */
    oldsigint_handler = signal(SIGINT, onsigintterm);
    oldsigterm_handler = signal(SIGTERM, onsigintterm);
    signal(SIGPIPE, SIG_IGN);

    SRV_DEBUG(0,
              ">>> SmAS started (pid %d), listening on socket %d <<<\n",
              getpid(), smas_server_sfd);

    /* We will use select() => we construct the fd list */
    FD_ZERO(&rfds);
    FD_SET(smas_server_sfd, &rfds);

    for (;;) {
        fdlist = rfds;          /* Because select() modifies the fd list */

        if (select(FD_SETSIZE, &fdlist, NULL, NULL, NULL) < 1) {
            fprintf(stderr,
                    "[SmAS Server]: unexpected select() error\n");
            onsigintterm(SIGINT);
        }

        if (FD_ISSET(smas_server_sfd, &fdlist)) {
            sfd =
                accept(smas_server_sfd,
                       (struct sockaddr *) &claddr, &len);
            SRV_DEBUG(0,
                      ">>> SmAS new client (%s) @ socket %d <<<\n",
                      inet_ntoa(claddr.sin_addr), sfd);
            new_server_thread(sfd, service);
        }
    }
}

/* Create a new service thread -- and put in the detached state
 * Well, we should normally check for the success of
 * pthread_create() - but for now we don't!
 */
```

```
new_server_thread(int sfd, void *(*func) (void *))
{
#ifdef HURD_DIST
    cthread_t tid;

    tid = cthread_fork(func, (void *) sfd);
    cthread_detach(tid);
#endif
#ifdef LINUX_DIST
    pthread_t tid;

    pthread_create(&tid, NULL, func, (void *) sfd);
    pthread_detach(tid);
#endif

}

/* When the client leaves, we clean up after him (i.e. delete
 * all the filters he registered and close all the files he opened)
 */
done_server_thread(int sfd)
{
    int i;

    filter_del_owner(sfd);
    file_cleanup_owner(sfd);
    close(sfd);
#ifdef HURD_DIST
    cthread_exit(NULL);
#endif
#ifdef LINUX_DIST
    pthread_exit(NULL);
#endif
}

/* This is what each server thread executes
 */
void *service(void *data)
{
    int mysock, msgtype;
    fd_set fdlist, lfds;

    mysock = (int) data;
    FD_ZERO(&lfds);
    FD_SET(mysock, &lfds);

    signal(SIGPIPE, SIG_IGN);

    SRV_DEBUG(1, "Service thread %u started @ socket %d\n",
              cthread_self(), mysock);

    for (;;) {
        fdlist = lfds;
        if (select(FD_SETSIZE, &fdlist, NULL, NULL, NULL) < 1) {
            fprintf(stderr,
                    "[service thread]: unexpected select() error\n");
            close(mysock);
            onsigintterm(SIGINT);
        }

        if (!FD_ISSET(mysock, &fdlist))
            continue;

        SRV_DEBUG(1, "(%d) activity on my socket\n",
                  cthread_self());

        /* Check if detected activity is of 0 bytes => client disconnected
           himself! */
        if (sock_unrecved(mysock) == 0) {
            SRV_DEBUG(1, "(%u) client done - exitting thread\n",
                      cthread_self());
            done_server_thread(mysock);
```

```
        }
        /* Now just look at the message's type without removing it from
           the input queue. */
        recv(mysock, (char *) &msgtype, sizeof(int), MSG_PEEK);
        switch (handle_message(mysock, ntohl(msgtype))) {
        case -1:
            SRV_DEBUG(-1,
                      "(%u) handle_message() allocation problem\n",
                      cthread_self());
            break;
        case -2:
            SRV_DEBUG(-1,
                      "(%u) handle_message() unknown message type\n",
                      cthread_self());
            /* Try to recover: flush the socket completely! */
            sock_flush(mysock, sock_unrecved(mysock));
            break;
        case 0:
            SRV_DEBUG(1, "(%u) done; waiting again ..\n",
                      cthread_self());
            break;
        }
    }
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                   *
 *       DIRECTORY, LOCKS AND EXIT STUFF                            *
 * *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#include <sys/stat.h>

isdir(char *path)
{
    struct stat sb;

    stat(path, &sb);
    return (S_ISDIR(sb.st_mode));
}

lockfile()
{
    FILE *fp;

    /* Check if it exists */

    if ((fp = fopen("/tmp/.SmAS_server_lock", "r")) != NULL) {
        fclose(fp);
        fprintf(stderr,
                "[SmAS_server]: a server is already running\n");
        exit(1);
    }

    if ((fp = fopen("/tmp/.SmAS_server_lock", "w")) == NULL)
        exit_smas(1, "[SmAS_server]: cannot create lock file\n");
    fprintf(fp, "%d", getpid());
    fclose(fp);
}

exit_smas(int value, char *format, ...)
{
    va_list ap;

    if (format != NULL) {
        va_start(ap, format);
        vfprintf(stderr, format, ap);
    }
```

```
    /* Remove the lock file */
    unlink("/tmp/.SmAS_server_lock");
    exit(value);
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * *
 *       DEBUGGING ... *
 * *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#ifdef SMAS_SRV_DEBUG

  /* It may seem strange but SRV_DEBUG will crash if we do not lock it!!
     --- maybe it is an fputc() or vprintf() problem. */
#ifdef HURD_DIST
struct mutex debuglock = MUTEX_INITIALIZER;
#endif
#ifdef LINUX_DIST
pthread_mutex_t debuglock = PTHREAD_MUTEX_INITIALIZER;
#endif

SRV_DEBUG(int depth, char *format, ...)
{
    va_list ap;
    char c = ' ', i;

    if (depth < 0) {
        depth = -depth;
        c = '#';
    }

    mutex_lock(&debuglock);

    for (i = 1; i < 2 * depth; i++)
        fputc(c, stderr);
    if (depth > 0)
        fputc(' ', stderr);

    va_start(ap, format);
    vfprintf(stderr, format, ap);

    if (c == '#')
        fputc(7, stderr);

    mutex_unlock(&debuglock);
}

#else

SRV_DEBUG(int depth, char *format, ...)
{
}

#endif
```

A.12 server/server.h

```
/* SERVER.H
 * Interface to SERVER.C
 */

/* #define the following to get debugging info at the server's stderr
 * Should only be used by the developers!
#define SMAS_SRV_DEBUG
 */

/* Used to print debugging info - it actually does so only if
 * SMAS_SRV_DEBUG has been defined.
 * This first parameter (int depth) has the following meaning:
 *  abs(depth) is the function call depth (gives the indentation to be used)
```

```
 * sign(depth) determines whether the info refers to an error or not.
 * Examples:
 *   SRV_DEBUG(0, ">>> SmAS started");
 *            Gives a nice info to the developer
 *   SRV_DEBUG(1, "Message handled ok");
 *            Nice info, too but will be indented by 2 spaces (1 level)
 *   SRV_DEBUG(1,  "unknown message received");
 *            This marks an error; it will be indented by positions
 *            which will bot be spaces but '#'s.
 */
extern SRV_DEBUG(int, char *, ...);

extern char smas_root_filters[500];     /* Where filters go */
extern char smas_root_files[500];       /* Where files go */

extern exit_smas(int, char *, ...);
```

A.13 server/smas_stop.c

```
/* SMAS_STOP.C
 * Stops the server (if running)
 *
 * Revisions:
 * ==========
 * 0.2.4, 18/12/2000
 *    Now removes the lock file when the server cannot be
 *    stopped (covers the case where SmAS was killed and
 *    did not have the chance to remove the lock).
 * 0.2.3, 07/12/2000
 *    First time!
 */
#include <stdio.h>
#include <signal.h>

main()
{
    FILE *fp;
    int spid;

    if ((fp = fopen("/tmp/.SmAS_server_lock", "r")) == NULL)
        exit(0);
    fscanf(fp, "%d", &spid);
    fclose(fp);
    if (kill(spid, SIGINT) != 0) {
        fprintf(stderr,
                "SmAS server could not be stopped; cleaning any leftovers ..\n");
        /* Here we must assume that the lock file has been left over when
           SmAS server was brutally killed. So, we try to remove it
           ourselves! */
        unlink("/tmp/.SmAS_server_lock");
    }
}
```

A.14 server/threadfunc.c

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "buffers.h"
#include "filters.h"
#include "files.h"

#ifdef USE_MYCTHREAD
#include "../libthreads/cthread_internals.h"
extern volatile cproc_t running_cproc;
```

```
/*Get's cproc from cthread - See cthreads.h#cthread_assoc()*/
#define get_cproc(c) ((cproc_t)((c)->ur))
#endif

void *readth(void *b)
{
    register file_buffer_t *buf = (file_buffer_t *) b;
    int res = 0;
#ifdef USE_MYCTHREAD
    cproc_t p;
#endif

    while (!(buf->status & BUF_SDESTROY)) {
        mutex_lock(&buf->stop);
        if (buf_eof(buf)) {
            mutex_unlock(&buf->stop);
#ifndef USE_MYCTHREAD
            mutex_lock(&buf->wait_lock);
            condition_signal(&buf->data_needed);
            mutex_unlock(&buf->wait_lock);
#endif
            cthread_yield();
            continue;
        }
        res = file_buf_fetch(buf);
        mutex_unlock(&buf->stop);
        if (res == 0) {
#ifdef USE_MYCTHREAD
            p = get_cproc(cthread_self());
            if (p->group_next != 0)     /* This means that this buffer is
                                           been read by a filter thread */
                p = p->group_next;
            else
                p = running_cproc;      /* Or let get service thread and
                                           pass control */
            thread_switch(p->mach_thread, SWITCH_OPTION_DEPRESS,
                          15);
#else
            mutex_lock(&buf->wait_lock);
            condition_signal(&buf->data_needed);
            mutex_unlock(&buf->wait_lock);
            cthread_yield();
#endif
        }
    }
    return NULL;
}

void *nextrecth(void *i)
{
    int fd = (int) i;
    int fid, recsize;
    int size = 0, res, vs;
    unsigned char *fmem;
    unsigned char *rbuf;
    filter_buffer_t *buf;
#ifdef USE_MYCTHREAD
    cproc_t p;
#endif

    fid = file_fid(fd);
    fmem = file_filtermem(fd);
    size = filter_readsize(fid);
    buf = (filter_buffer_t *) filter_getbuffer(fid);
    rbuf = (size > 0) ? malloc(size) : NULL;
    while (!(buf->status & BUF_SDESTROY)) {
        mutex_lock(&buf->stop);
        if (buf_eof(buf)) {
```

```
            mutex_unlock(&buf->stop);
#ifndef USE_MYCTHREAD
            mutex_lock(&buf->wait_lock);
            condition_signal(&buf->data_needed);
            mutex_unlock(&buf->wait_lock);
#endif
            cthread_yield();
            continue;
        }
        res = file_buf_read2(buf->src.buf, rbuf, size, &vs);
        if (res == size
            && filter_body(fid, rbuf, &recsize,
                           (void **) &fmem)) {
            while (!filter_buf_append_record
                   (buf, fmem, recsize, vs)) {
                mutex_unlock(&buf->stop);
                /* Buffer is full - wakeup reader */
#ifdef USE_MYCTHREAD
                /* In our implementation we can switch to the consuming
                   thread quite easily */
                p = running_cproc;      /* Let's get service thread and
                                           pass control */
                thread_switch(p->mach_thread,
                              SWITCH_OPTION_DEPRESS, 15);
#else
                mutex_lock(&buf->wait_lock);
                condition_signal(&buf->data_needed);
                mutex_unlock(&buf->wait_lock);
#endif
                cthread_yield();
                mutex_lock(&buf->stop);
            }
        }
        mutex_unlock(&buf->stop);
#ifdef USE_MYCTHREAD
        /* In our implementation we can switch to the consuming thread
           quite easily */
        p = running_cproc;      /* Let's get service thread and pass
                                   control */
        thread_switch(p->mach_thread, SWITCH_OPTION_DEPRESS, 15);
#else
        mutex_lock(&buf->wait_lock);
        condition_signal(&buf->data_needed);
        mutex_unlock(&buf->wait_lock);
        cthread_yield();
#endif
    }
    free(rbuf);

    return NULL;
}
```

A.15 server/timer.h

```
#ifndef TIMERMAN_H
#define TIMERMAN_H 1

/*Provides time measurement using different function.
  Currently supports ftime(),gettimeofday(),times().

  The following structures are defined:
  ttimer: A structure that holds timer data
  ttimer_t: A pointer to the above structure

  The following macros are defined:
  new_timer(ttimer_t): Allocates a new zeroed ttimer structure
  free_timer(ttimer_t): Deallocates a previously allocated structure with new_timer()
```

```
  clear_timer(ttime_t): Initializes a timer struct to zero
  eq_timer(ttimer_t ,ttimer_t): True if the 2 arguments represent the same moment
  ne_timer(ttimer_t ,ttimer_t): True if the 2 arguments represent the different moments
  sub_timer(ttimer_t ,ttimer_t): Returns the amount of time between the 2 arguments. Note that when using gettimeofday() you get microseconds(10^-6)
  refresh_timer(ttimer_t): Stores current moment in timer struct
*/
//#define TIMER_FTIME
#define TIMER_GETTIMEOFDAY
//#define TIMER_TIMES

/*Using gettimeofday*/
#ifdef TIMER_GETTIMEOFDAY
#include <sys/time.h>
#include <unistd.h>

typedef struct timeval ttimer;
typedef struct timeval *ttimer_t;

#define new_timer(tvp) (tvp)=(ttimer_t)calloc(1,sizeof(struct timeval))
#define free_timer(tvp) free(tvp)

#define clear_timer(tvp)  timerclear(tvp)

#define eq_timer(tv1p,tv2p) ((tv1p)->tv_sec == (tv2p)->tv_sec &&\
(tv1p)->tv_usec == (tv2p)->tv_usec)
#define ne_timer(tv1p,tv2p) timercmp(tv1p,tv2p,!=)

#define sub_timer(tv1p,tv2p) (((tv1p)->tv_sec == (tv2p)->tv_sec)? \
((tv2p)->tv_usec - (tv1p)->tv_usec) \
:((tv2p)->tv_sec - (tv1p)->tv_sec)*1000000 - \
((tv1p)->tv_usec - (tv2p)->tv_usec))
#define refresh_timer(tv1p) gettimeofday((tv1p),NULL)
#endif

/*Using ftime*/
#ifdef TIMER_FTIME
#include <sys/timeb.h>
typedef struct timeb ttimer;
typedef struct timeb *ttimer_t;

#define new_timer(tvp) (tvp)=(ttimer_t)calloc(1,sizeof(struct timeb))
#define free_timer(tvp) free(tvp)

#define clear_timer(tvp)  (tvp)->time=(tvp)->millitm=(tvp)->timezone=(tvp)->dstflag=0;

#define eq_timer(tv1p,tv2p) ((tv1p)->time == (tv2p)->time &&\
(tv1p)->millitm == (tv2p)->millitm)
#define ne_timer(tv1p,tv2p) ((tv1p)->time != (tv2p)->time ||\
(tv1p)->millitm != (tv2p)->millitm)

#define sub_timer(tv1p,tv2p) (((tv1p)->time == (tv2p)->time)? \
((tv2p)->millitm - (tv1p)->millitm) \
:((tv2p)->time - (tv1p)->time)*1000 - \
((tv1p)->millitm - (tv2p)->millitm))
#define refresh_timer(tv1p) ftime((tv1p))


#endif

/*Using mach's thread info*/
/*This doesn't work very well*/
#if 0
#ifdef TIMER_MACH
#include <mach.h>
#include <unistd.h>

typedef struct thread_basic_info ttimer;
typedef struct thread_basic_info *ttimer_t;

#define new_timer(tvp) (tvp)=(ttimer_t)calloc(1,sizeof(ttimer))
#define free_timer(tvp) free(tvp)

#define clear_timer(tvp)  memset(tvp,0,sizeof(ttimer))
```

```
#define eq_timer(tv1p,tv2p) ((tv1p)->user_time == (tv2p)->user_time &&\
(tv1p)->system_time == (tv2p)->system_time)
#define ne_timer(tv1p,tv2p) ((tv1p)->user_time.seconds  != (tv2p)->user_time.seconds  ||\
(tv1p)->system_time.seconds  != (tv2p)->system_time.seconds  ||\
(tv1p)->user_time.microseconds  != (tv2p)->user_time.microseconds  ||\
(tv1p)->system_time.microseconds  != (tv2p)->system_time.microseconds )

#define sub_timeru(tv1p,tv2p) (((tv1p)->user_time.seconds == (tv2p)->user_time.seconds)? \
((tv2p)->user_time.microseconds - (tv1p)->user_time.microseconds) \
:((tv2p)->user_time.seconds - (tv1p)->user_time.seconds)*1000000 - \
((tv1p)->user_time.microseconds - (tv2p)->user_time.microseconds))
#define sub_timers(tv1p,tv2p) (((tv1p)->system_time.seconds == (tv2p)->system_time.seconds)? \
((tv2p)->system_time.microseconds - (tv1p)->system_time.microseconds) \
:((tv2p)->system_time.seconds - (tv1p)->system_time.seconds)*1000000 - \
((tv1p)->system_time.microseconds - (tv2p)->system_time.microseconds))

#define sub_timer(tv1p,tv2p) (sub_timeru(tv1p,tv2p)+sub_timers(tv1p,tv2p))
#define refresh_timer(t,tv1p,x) thread_info((t),THREAD_BASIC_INFO,(tv1p),&x)
#endif
#endif

/*Using times*/
#ifdef TIMER_TIMES
#include <sys/times.h>
#include <unistd.h>

typedef clock_t ttimer;
typedef clock_t *ttimer_t;
static struct tms dumtms;
#define new_timer(tvp) (tvp)=(ttimer_t)calloc(1,sizeof(ttimer))
#define free_timer(tvp) free(tvp)

#define clear_timer(tvp)  ((*(tvp))=(clock_t)0)

#define eq_timer(tv1p,tv2p) ((*(tv1p)) == (*(tv2p)))
#define ne_timer(tv1p,tv2p) ((*(tv1p)) != (*(tv2p)))
#define sub_timer(tv1p,tv2p) ((*(tv2p))-(*(tv1p)))
#define refresh_timer(tv1p) ((*(tv1p))=times(&dumtms))
#endif

#endif
```

A.16 include/smasdefs.h

```
/* SMASDEFS.H
 * SmAS Definifitions: the interface file between libsmas and the server
 *
 * Revisions:
 * ==========
 * AFTS 0.2 05/06/2000
 *    Completely rewritten by VVD, CS, UoI
 *
 * AFTS 0.1 ??/06/2000
 *    Untouched
 */

#ifndef __SMASDEFS_H

#include <errno.h>              /* Used for new error reporting code */

#define SERVER_PORT   9735

#ifndef __MYBYTE__
typedef unsigned char byte;
#define __MYBYTE__
#endif

#define MAX_PNAME 256           /* For a full path */
#define MAX_FNAME 64            /* Just for the filename */

/* The various message types
```

```
 */
typedef enum { SMAS_OPEN = 1,
    SMAS_CLOSE = 2,
    SMAS_READ = 3,
    SMAS_WRITE = 4,
    SMAS_LSEEK = 5,
    SMAS_REGFIL = 6,
    SMAS_APPFIL = 7,
    SMAS_NEXTREC = 8
} smas_msg_t;

/*typedef unsigned long int smas_msg_t; /*Always 4 bytes*/
/* The various message bodies
 */
typedef struct {
    smas_msg_t msgtype;
    int flags;
    int mode;
    char fname[MAX_PNAME];
} msgopen_t;

typedef struct {
    smas_msg_t msgtype;
    int fd;
} msgclose_t;

typedef struct {
    smas_msg_t msgtype;
    int fd;
    int nbytes;
} msgread_t;

typedef struct {
    smas_msg_t msgtype;
    int fd;
    int nbytes;
} msgwrite_t;

typedef struct {
    smas_msg_t msgtype;
    int fd;
    int offset;
    int whence;
} msglseek_t;

typedef struct {
    smas_msg_t msgtype;
    int codesize;
    char fname[MAX_FNAME];
} msgregfil_t;

typedef struct {
    smas_msg_t msgtype;
    int fd;
    int fid;
} msgappfil_t;

typedef struct {
    smas_msg_t msgtype;
    int fd;
} msgnextrec_t;
/*******Scheduler related code**********/
/*This is a remnant of our attempt to implement
 * shortest job scheduling everything else is removed*/
struct sched_dat {
    int pri;
    int count;
    void *tp;
};

#define CHECK_ERROR(x) if ((x)<0) SRV_DEBUG(-2,"(%u) %s @ %d : %s\n", \
cthread_self(),__FILE__, __LINE__ ,strerror(errno))
```

```
#define __SMASDEFS_H
#endif
```

A.17 include/socks.c

```
/* SOCKS.C
 * Standard code for socket programming
 * VVD, CS, UoI, 2000
 *
 * Revisions:
 * ==========
 * AFTS 0.2.2 14/06/2000
 *      Added sock_writev().
 */
#include <string.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include "socks.h"
/*We don't need this*/
#if 0
int sock_unblock(int sockfd)
{
    int flags;
    /* Get file descriptor associated flags */
    if ((flags = fcntl(sockfd, F_GETFL)) < 0)
        return flags;
    /* Add non-blocking option */
    flags |= O_NONBLOCK;
    /* Set file descriptor associated flags */
    if ((flags = fcntl(sockfd, F_SETFL, flags)) < 0)
        return flags;
}
#endif
server_open_socket(short port, int maxclients)
{
    struct sockaddr_in addr;
    int sfd;
    /* Open a socket fd */
    if ((sfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        return (-1);
    /* Bind to a port */
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    addr.sin_port = htons(port);
    if (bind(sfd, (struct sockaddr *) &addr, sizeof(addr)) < 0) {
        close(sfd);
        return (-2);
    }
    /* Listen for connection - not needed in SOCK_DGRAM sockets */
    listen(sfd, maxclients);
    return (sfd);
}

server_wait_connection(int sfd, struct sockaddr_in * client)
{
    struct sockaddr_in remote;
    int addrlen;
```

```
    if (client == NULL)
        client = &remote;
    return (accept(sfd, (struct sockaddr *) client, &addrlen));
}

client_open_socket(char *serveraddr, short port)
{
    struct sockaddr_in inaddr;
    struct in_addr ina;
    struct hostent *serverinfo;
    unsigned long int ip;
    int sfd;

    /* Open a socket fd */
    if ((sfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        return (-1);

    /* Now find server's real address. The following code works whether
       user passed a numeric or a named address! serverinfo = ((ip =
       inet_addr(serveraddr)) == INADDR_NONE) ? gethostbyname(serveraddr)
       : gethostbyaddr((char *) &ip, sizeof(int), AF_INET); */

    serverinfo = gethostbyname(serveraddr);
    if (serverinfo == NULL)
        return (-2);
/*  if (ip == INADDR_NONE)
  {
    struct hostent *t = serverinfo;

    for (; *(t->h_addr_list); )
    {
      bcopy(*(t->h_addr_list), (char *) &ina, t->h_length);
      (t->h_addr_list)++;
      ip = inet_addr( inet_ntoa(ina) );
    }
    serverinfo = gethostbyname(serveraddr);
  }

  if (serverinfo == NULL)
    return (-2);*/

    /* Bind to a port */
    memset(&inaddr, 0, sizeof(inaddr));
    inaddr.sin_family = AF_INET;
    memcpy((void *) &inaddr.sin_addr,
           (void *) serverinfo->h_addr_list[0], 4);
    inaddr.sin_port = htons(port);
    if (connect(sfd, (struct sockaddr *) &inaddr, sizeof(inaddr))
        < 0) {
        close(sfd);
        return (-3);
    }

    return (sfd);
}

client_open_numeric_socket(char *serverbyteaddr, short port)
{
    struct sockaddr_in addr;
    int sfd;

    /* Open a socket fd */
    if ((sfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        return (-1);

    /* Bind to a port */
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = inet_addr(serverbyteaddr);
    addr.sin_port = htons(port);
```

```
    if (connect(sfd, (struct sockaddr *) &addr, sizeof(addr)) <
        0) {
        close(sfd);
        return (-2);
    }

    return (sfd);
}
#if 0
#undef CHECK_ERROR
#define CHECK_ERROR(x) if ((x)<0) fprintf(stderr,"%s @ %d : %s\n", \
__FILE__, __LINE__ ,strerror(errno))
/*Dummy function that enable better error checking*/

sock_write(int fd, char *buf, int n)
{
    int res = Sock_write(fd, buf, n);
    CHECK_ERROR(res);
    printf("Write returns : %d\n", res);
}

sock_read(int fd, char *buf, int n)
{
    int res = Sock_read(fd, buf, n);
    CHECK_ERROR(res);
    printf("Read returns : %d\n", res);
}
#undef CHECK_ERROR
#endif
sock_write(int fd, char *buf, int n)
{
    int res = 0, sent = 0;

    for (; sent < n;) {
        res = write(fd, buf + sent, n - sent);
        if (res < 0 && errno != EAGAIN && errno != EINTR)
            return res;
        else if (res == 0)
            return 0;
        else if (res > 0)
            sent += res;
    }
}

sock_read(int fd, char *buf, int n)
{
    int res = 0, recv = 0;

    for (; recv < n;) {
        res = read(fd, buf + recv, n - recv);
        if (res < 0 && errno != EAGAIN && errno != EINTR)
            return res;
        else if (res == 0)
            return 0;
        else if (res > 0)
            recv += res;
    }
}


sock_writev(int fd, struct iovec *vec, int rows)
{
    int nw;

    for (;;) {
        nw = writev(fd, vec, rows);
        if (nw < 0 && errno != EAGAIN)  /* The rest of the code isn't
                                           checked for non-blocking
                                           sockets */
            return nw;
        else if (nw == 0)
            return 0;
```

```
        for (;;) {              /* Check how much was written by writev() */
            if (nw < vec->iov_len) {    /* This row not entirely written
                                            (A) */
                vec->iov_base += nw;    /* Point at unwritten part */
                vec->iov_len -= nw;
                break;          /* Continue with current row (B) */
            }
            /* The row was sent; prepare to ckeck if nw covered the next
               row, too */
            nw -= vec->iov_len;
            vec++;              /* Next row */
            rows--;             /* i.e. one row less to send */
            if (nw == 0)        /* It means that (A) was not satisfied
                                   because */
                return rows;    /* nw was EQUAL to vec->iov_len */
        }
        /* We come here only because of (B) */
    }                           /* Continue sending */
    return rows;
}

/* This funtion empties a socket a character at a time
 * It is useful in panic situations
 */
sock_flush(int fd, int n)
{
    char c;

    for (; n > 0; n--)
        if (read(fd, &c, 1) < 0)
            return n;
}

/* Returns the number of bytes of unsent data in the send queue.
 * Got it from tcp(4) man page.
 */
sock_unsent(int sfd)
{
    int n;

    ioctl(sfd, TIOCOUTQ, &n);
    return (n);
}

sock_unrecved(int sfd)
{
    int n;

    ioctl(sfd, FIONREAD, &n);
    return (n);
}
```

A.18 include/socks.h

```
/* SOCKS.H
 * Standard code for socket programming
 * VVD, CS, UoI, 2000
 */
#ifndef __SOCKS_H

#include <sys/types.h>
#include <sys/socket.h>         /* Includes gethostbyaddr() */
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>              /* For gethostbyname() */
```

```
#include <sys/uio.h>            /* For writev() */

#if 0
extern unblock(int /* sfd */ );
#endif

extern server_open_socket(short /* port */ ,
                          int /* maxclients */ );
extern server_wait_connection(int /* sfd */ ,
                              struct sockaddr_in * /* client */
                              );

extern client_open_socket(char * /* serveraddr */ ,
                          short /* port */ );
extern client_open_numeric_socket(char * /* serverbyteaddr */ ,
                                  short /* port */ );

extern sock_write(int /* sfd */ , char * /* buf */ ,
                  int /* nbytes */ );
extern sock_read(int /* sfd */ , char * /* buf */ ,
                 int /* nbytes */ );
extern sock_writev(int /* sfd */ , struct iovec * /* vec */ ,
                   int /* rows */ );

extern sock_flush(int /* sfd */ , int /* nbytes */ );

extern sock_unsent(int /* sfd */ );
extern sock_unrecved(int /* sfd */ );

#define __SOCKS_H
#endif
```

Παράρτημα Β

Κώδικας βιβλιοθήκης CThread

Λίστα αρχείων, με * σημειώνονται τα αρχεία που τροποποιήθηκαν στη δική μας υλοποίηση:

`libthreads/Makeconf`
`libthreads/Makefile*`
`libthreads/call.c`
`libthreads/cancel-cond.c`
`libthreads/cprocs.c*`
`libthreads/cthread_data.c`
`libthreads/cthread_internals.h*`
`libthreads/cthreads.c*`
`libthreads/cthreads.h*`
`libthreads/libthreads.map`
`libthreads/lockfile.c`
`libthreads/mig_support.c`
`libthreads/options.h`
`libthreads/rwlock.c`
`libthreads/rwlock.h`
`libthreads/stack.c`
`libthreads/sync.c`
`libthreads/timer.h`

Παρακάτω παρουσιάζονται μόνο τα αρχεία που μεταβλήθηκαν στη δική μας υλοποίηση.

B.1 libthreads/Makefile

```
#
#   Copyright (C) 1994,95,96,97,2000 Free Software Foundation, Inc.
#
#   This program is free software; you can redistribute it and/or
#   modify it under the terms of the GNU General Public License as
#   published by the Free Software Foundation; either version 2, or (at
#   your option) any later version.
#
#   This program is distributed in the hope that it will be useful, but
#   WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
#   General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with this program; if not, write to the Free Software
#   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

dir := libthreads
makemode := library

# In GNU mig_support.c, sync.c and machine/lock.s are omitted; that work is
# all done in libc.
SRCS := call.c cprocs.c cthread_data.c cthreads.c stack.c \
cancel-cond.c rwlock.c lockfile.c
I386SRCS := i386/csw.S i386/thread.c

# In GNU machine/cthreads.h is omitted; that work is done in libc headers.
LCLHDRS := cthread_internals.h options.h cthreads.h rwlock.h

OBJS = $(addsuffix .o,$(basename $(notdir $(SRCS) $(I386SRCS))))

OTHERTAGS = $(I386SRCS) $(I386HDRS)

libname = libmythreads
installhdrs = cthreads.h rwlock.h
installhdrsubdir = .

VPATH += $(srcdir)/$(asm_syntax)

include ./Makeconf

# The threads library was written by CMU.  If you've ever experienced
# what that means, you'll understand this line.
CFLAGS := $(filter-out -Wall,$(CFLAGS))

lndist: lndist-i386-files

lndist-i386-files: $(top_srcdir)/hurd-snap/$(dir)/i386
ln $(addprefix $(srcdir)/,$(I386SRCS) $(I386HDRS)) $<

$(top_srcdir)/hurd-snap/$(dir)/i386:
mkdir $@

ifeq ($(VERSIONING),yes)

# Adding this dependency gets it included in the command line,
# where ld will read it as a linker script.
$(libname).so.$(hurd-version): $(srcdir)/$(libname).map
endif
```

B.2 libthreads/cprocs.c

```
/*
 * Mach Operating System
 * Copyright (c) 1991,1990,1989 Carnegie Mellon University
 * All Rights Reserved.
 *
 * Permission to use, copy, modify and distribute this software and its
 * documentation is hereby granted, provided that both the copyright
 * notice and this permission notice appear in all copies of the
 * software, derivative works or modified versions, and any portions
```

```
 * thereof, and that both notices appear in supporting documentation.
 *
 * CARNEGIE MELLON ALLOWS FREE USE OF THIS SOFTWARE IN ITS "AS IS"
 * CONDITION.  CARNEGIE MELLON DISCLAIMS ANY LIABILITY OF ANY KIND FOR
 * ANY DAMAGES WHATSOEVER RESULTING FROM THE USE OF THIS SOFTWARE.
 *
 * Carnegie Mellon requests users of this software to return to
 *
 *  Software Distribution Coordinator  or  Software.Distribution@CS.CMU.EDU
 *  School of Computer Science
 *  Carnegie Mellon University
 *  Pittsburgh PA 15213-3890
 *
 * any improvements or extensions that they make and grant Carnegie Mellon
 * the rights to redistribute these changes.
 */
/*
 * HISTORY
 *
 *2001 Kinalis Athanasios
 *
 *
 * $Log: cprocs.c,v $
 * Revision 1.12  2000/01/10 14:42:30  kettenis
 * 2000-01-10  Mark Kettenis  <kettenis@gnu.org>
 *
 *  * cprocs.c: Include <assert.h>
 *
 * Revision 1.11  2000/01/09 23:00:18  roland
 * 2000-01-09  Roland McGrath  <roland@baalperazim.frob.com>
 *
 *  * cprocs.c (cproc_alloc): Initialize P->wired and P->msg here (code
 *  from cthread_wire).
 *  (cthread_wire): Reduce to just an assert, cthreads always wired.
 *  (chtread_unwire): Abort if called.
 *
 * Revision 1.10  1998/07/20 06:59:14  roland
 * 1998-07-20  Roland McGrath  <roland@baalperazim.frob.com>
 *
 *  * i386/csw.S (cproc_prepare): Take address of cthread_body as third
 *  arg, so we don't have to deal with PIC magic to find its address
 *  without producing a text reloc.
 *  * cprocs.c (cproc_create): Pass &cthread_body to cproc_prepare.
 *
 * Revision 1.9  1996/11/18 23:54:51  thomas
 * Mon Nov 18 16:36:56 1996  Thomas Bushnell, n/BSG  <thomas@gnu.ai.mit.edu>
 *
 *  * cprocs.c (cproc_create): Cast CHILD in assignment.
 *
 * Revision 1.8  1995/12/06 19:48:34  mib
 * (condition_unimplies): Take address of (*impp)->next in assignment to
 * IMPP on loop step instruction.
 *
 * Revision 1.7  1995/09/22 17:51:10  roland
 * Include hurd/threadvar.h.
 *
 * Revision 1.6  1995/08/30 15:57:47  mib
 * Repair typos.
 *
 * Revision 1.5  1995/08/30 15:50:53  mib
 * (cond_signal): If this condition has implications, see if one of them
 * needs to be signalled when we have no waiters.
 * (cond_broadcast): Signal the implications list too.
 * (condition_implies, condition_unimplies): New functions.
 *
 * Revision 1.4  1995/04/04 21:04:29  roland
 * (mutex_lock_solid, mutex_unlock_solid): Renamed to __*.
 * (_cthread_mutex_lock_routine, _cthread_mutex_unlock_routine): Variables
 * removed.
 *
 * Revision 1.3  1994/05/19  04:55:30  roland
 * entered into RCS
 *
 * Revision 2.15  92/03/06  14:09:31  rpd
 *  Replaced swtch_pri with yield.
```

```
*  [92/03/06            rpd]
*
* Revision 2.14  91/08/28  11:19:16  jsb
*  Fixed the loop in cproc_fork_child that frees cprocs.
*  [91/08/23            rpd]
*
* Revision 2.13  91/07/31  18:33:04  dbg
*  Fix some more bad types.  Ints are NOT pointers.
*
*  Fix argument type mismatch in cproc_create.
*  [91/07/30  17:32:59  dbg]
*
* Revision 2.12  91/05/14  17:56:11  mrt
*  Correcting copyright
*
* Revision 2.11  91/02/14  14:19:26  mrt
*  Added new Mach copyright
*  [91/02/13  12:40:50  mrt]
*
* Revision 2.10  90/11/05  14:36:41  rpd
*  Added cproc_fork_{prepare,parent,child}.
*  [90/11/02            rwd]
*
*  Fix for positive stack growth.
*  [90/11/01            rwd]
*
*  Add spin_lock_t.
*  [90/10/31            rwd]
*
* Revision 2.9  90/10/12  13:07:12  rpd
*  Fix type
*  [90/10/10  15:09:59  rwd]
*
*  Comment code.
*  [90/10/02            rwd]
*
* Revision 2.8  90/09/09  14:34:44  rpd
*  Remove special mutex.  Remove thread_calls and debug_mutex
*  [90/08/24            rwd]
*  Fix up old call to cthread_msg_busy to new format.
*  [90/08/22            rwd]
*
* Revision 2.7  90/08/06  15:09:17  rwd
*  Fixed arguments to cthread_mach_msg.
*  [90/06/26            rwd]
*  Add additional STATISTICS.
*  [90/06/07            rwd]
*
*  Attempt to reduce number of times a cthread is released to to a
*  msg_receive by adding min/max instead of single number to
*  cthread_msg calls.
*  [90/06/06            rwd]
*
* Revision 2.6  90/06/02  15:13:36  rpd
*  Converted to new IPC.
*  [90/03/20  20:46:16  rpd]
*
* Revision 2.5  90/05/29  18:40:11  rwd
*  Don't incr special field until the mutex grab is successful.
*  [90/05/09            rwd]
*
* Revision 2.4  90/03/14  21:12:02  rwd
*  Added WAIT_DEBUG code for deadlock debugging.
*  [90/03/01            rwd]
*  Insert cprocs in cproc_list as allocated.
*  [90/03/01  10:20:16  rwd]
*
* Revision 2.3  90/01/19  14:36:57  rwd
*  Make cthread_msg_busy only release new thread if this is still
*  busy.  Ie don't release two on back to back calls.
*  [90/01/11            rwd]
*  Add THREAD_CALL code.  Add CPROC_ARUN state.
```

```
*  [90/01/03            rwd]
*  Add new cthread_msg_rpc call
*  [89/12/20            rwd]
*  Change cproc_self pointer to top of stack.  Now need to change
*  the stack of the first thread.
*  [89/12/12            rwd]
*
* Revision 2.2  89/12/08  19:53:13  rwd
*  Added CPROC_CONDWAIT state to deal with lock held
*  across mutex_unlock problem.
*  [89/11/29            rwd]
*  Changed mutexes to not hand off.  MUTEX_EXTRA conditional is
*  now obsolete.
*  [89/11/27            rwd]
*
*  Add MUTEX_EXTRA code for extra kernel threads to serve special
*  mutexes in time of need.
*  [89/11/25            rwd]
*  Add MUTEX_SPECIAL and DEBUG_MUTEX code
*  [89/11/24            rwd]
*  Changed mutex_lock to mutex_lock_solid.  Mutex_lock is now a
*  macro which tries the spin_lock before making a subroutine call.
*  Mutex_unlock is now a macro with mutex_unlock_solid for worst case.
*  [89/11/13            rwd]
*
*  Rewrite most to merge coroutine and thread implementation.
*  New routines are cthread_set_kernel_limit, cthread_kernel_limit,
*  cthread_wire, cthread_unwire, and cthread_receive.
*  [89/10/23            rwd]
*
* Revision 2.1  89/08/03  17:07:10  rwd
* Created.
*
* 11-Apr-89  David Golub (dbg) at Carnegie-Mellon University
* Made condition_yield loop break if swtch_pri returns TRUE (in
* case we fix it).
*
* 31-Mar-89  David Golub (dbg) at Carnegie-Mellon University
* Change cond_signal, cond_broadcast, and cproc_continue so that
* the condition's spin lock is not held while continuing the
* process.
*
* 16-Jan-89  David Golub (dbg) at Carnegie-Mellon University
* Changes for stand-alone library to run on pure kernel:
* . made IPC_WAIT standard, as calls that are used if IPC_WAIT == 0
*   vanished a year ago.
* . Removed (as much as possible) references to stdio or other U*X
*   features.
*
*
* 01-Apr-88  Eric Cooper (ecc) at Carnegie Mellon University
* Changed condition_clear(c) to acquire c->lock,
* to serialize after any threads still doing condition_signal(c).
* Suggested by Dan Julin.
*
* 19-Feb-88  Eric Cooper (ecc) at Carnegie Mellon University
*  Extended the inline scripts to handle spin_unlock() and mutex_unlock().
*
* 28-Jan-88  David Golub (dbg) at Carnegie Mellon University
* Removed thread_data argument from thread_create
* and converted to new thread_set_state call.
*
* 01-Dec-87  Eric Cooper (ecc) at Carnegie Mellon University
* Added inline expansion for cthread_sp() function.
*
* 21-Aug-87  Eric Cooper (ecc) at Carnegie Mellon University
* Fixed uninitialized reply_port in cproc_alloc() (found by rds).
*
* 14-Aug-87  Eric Cooper (ecc) at Carnegie Mellon University
* Tried using return value of swtch() to guide condition_wait().
* Performance was worse than using a hybrid spin/yield/block
* scheme, so the version using swtch() was commented out.
```

```
 * Disabled IPC_WAIT in released version.
 *
 * 13-Aug-87  Eric Cooper (ecc) at Carnegie Mellon University
 * Added IPC_WAIT option.
 * If defined, thread synchronization (condition_wait() and
 * cproc_continue()) are implemented using msg_receive() and
 * msg_send() instead of thread_suspend() and thread_resume().
 *
 * 11-Aug-87  Eric Cooper (ecc) at Carnegie Mellon University
 * Moved thread reply port to cproc structure in cthread_internals.h,
 * because mig calls are made while cproc is idle (no cthread structure).
 * Changed cproc_switch() and cproc_start (COROUTINE implementation)
 * to use address of saved context, rather than address of enclosing cproc,
 * to eliminate dependency on cproc layout.
 */
/*
 *  File:  cprocs.c
 * Author: Eric Cooper, Carnegie Mellon University
 * Date: Aug, 1987
 *
 *  Implementation of cprocs (lightweight processes)
 *  and primitive synchronization operations.
 */

#include "cthreads.h"
#include "cthread_internals.h"
#include <mach/message.h>
#include <hurd/threadvar.h>     /* GNU */
#include <assert.h>
#define ECHO_ON 1 /**/
#include "echoer.h"
#include "timer.h"
/*
 * C Threads imports:
 */
extern void alloc_stack();
extern void cproc_switch();     /* cproc context switch */
extern void cproc_start_wait(); /* cproc idle thread */
extern vm_offset_t cproc_stack_base();  /* return start of stack */
extern vm_offset_t stack_init();

/***********Scheduler related code**************************/

extern int schedon;
extern int user_init;
int ready_queue_counter = 0;
struct cthread_queue sched_cprocs_ready = QUEUE_INITIALIZER;
struct mutex sched_queue_lock = MUTEX_INITIALIZER;
struct cthread_queue cprocs_done = QUEUE_INITIALIZER;
struct mutex done_queue_lock = MUTEX_INITIALIZER;
cproc_t running_cproc = NO_CPROC;
cproc_t sched_cproc = NO_CPROC;
/********End of scheduler related code*********************/

/*
 * Port_entry's are used by cthread_mach_msg to store information
 * about each port/port_set for which it is managing threads
 */

typedef struct port_entry {
    struct port_entry *next;    /* next port_entry */
    mach_port_t port;           /* which port/port_set */
    struct cthread_queue queue; /* queue of runnable threads for this
                                   port/port_set */
    int min;                    /* minimum number of kernel threads to be
                                   used by this port/port_set */
    int max;                    /* maximum number of kernel threads to be
                                   used by this port/port_set */
```

```
    int held;                   /* actual number of kernel threads
                                   currentlt in use */
    spin_lock_t lock;           /* lock governing all above fields */
} *port_entry_t;

#define PORT_ENTRY_NULL ((port_entry_t) 0)

/* Available to outside for statistics */

int cthread_wait_stack_size = 8192;     /* stack size for idle threads */
int cthread_max_kernel_threads = 0;     /* max kernel threads */
int cthread_kernel_threads = 0; /* current kernel threads */
private spin_lock_t n_kern_lock = SPIN_LOCK_INITIALIZER;
                                        /* lock for 2 above */

cproc_t cproc_list = NO_CPROC;  /* list of all cprocs */
private cproc_list_lock = SPIN_LOCK_INITIALIZER;
                                        /* lock for above */
private int cprocs_started = FALSE;     /* initialized? */
private struct cthread_queue ready = QUEUE_INITIALIZER;
/* ready queue *//*This queue is always empty */
private int ready_count = 0;    /* number of ready threads on ready queue
                                   - number of messages sent */
private spin_lock_t ready_lock = SPIN_LOCK_INITIALIZER;
                                        /* lock for 2 above */
private mach_port_t wait_port = MACH_PORT_NULL;
                                        /* port on which idle threads wait
                                         */
private int wait_count = 0;     /* number of waiters - messages pending to
                                   wake them */
private struct cthread_queue waiters = QUEUE_INITIALIZER;
                                        /* queue of cthreads to run as
                                           idle */
private spin_lock_t waiters_lock = SPIN_LOCK_INITIALIZER;
                                        /* lock for 2 above */
private port_entry_t port_list = PORT_ENTRY_NULL;
                                        /* master list of port_entries */
private spin_lock_t port_lock = SPIN_LOCK_INITIALIZER;
                                        /* lock for above queue */
private mach_msg_header_t wakeup_msg;   /* prebuilt message used by idle
                                           threads */

/*
 * Return current value for max kernel threads
 * Note: 0 means no limit
 */

cthread_kernel_limit()
{
    return cthread_max_kernel_threads;
}

/*
 * Set max number of kernel threads
 * Note: This will not currently terminate existing threads
 *  over maximum.
 */

cthread_set_kernel_limit(n)
int n;
{
    cthread_max_kernel_threads = n;
}

/*
 * Wire a cthread to its current kernel thread
 */

void cthread_wire()
{
    register cproc_t p = cproc_self();
```

```
    kern_return_t r;

    /* In GNU, we wire all threads on creation (in cproc_alloc).  */
    assert(p->wired != MACH_PORT_NULL);
}

/*
 * Unwire a cthread.  Deallocate its wait port.
 */

void cthread_unwire()
{
    register cproc_t p = cproc_self();

    /* This is bad juju in GNU, where all cthreads must be wired.  */
    abort();
#if 0
    if (p->wired != MACH_PORT_NULL) {
        MACH_CALL(mach_port_mod_refs(mach_task_self(), p->wired,
                                     MACH_PORT_RIGHT_SEND, -1),
                  r);
        MACH_CALL(mach_port_mod_refs
                  (mach_task_self(), p->wired,
                   MACH_PORT_RIGHT_RECEIVE, -1), r);
        p->wired = MACH_PORT_NULL;
#ifdef STATISTICS
        spin_lock(&wired_lock);
        cthread_wired--;
        spin_unlock(&wired_lock);
#endif                          /* STATISTICS */
    }
#endif
}

private cproc_t cproc_alloc(int f)
{
    register cproc_t p = (cproc_t) malloc(sizeof(struct cproc));
    kern_return_t r;
    cproc_t self;

    p->incarnation = NO_CTHREAD;
#if 0
    /* This member is not used in GNU.  */
    p->reply_port = MACH_PORT_NULL;
#endif

    spin_lock_init(&p->lock);
        /*********************/
    if (sched_cproc == NO_CPROC)
        sched_cproc = p;
    p->group_next = NULL;
    mutex_lock(&sched_queue_lock);
    if (user_init) {

        if (f) {
            sched_queue_enq(&sched_cprocs_ready, p);
        } else {
            self = cproc_self();
            while (self->group_next != NULL)    /* Enter new cproc in this
                                                   cproc's group queue */
                self = self->group_next;
            self->group_next = p;
        }
    }
        /*********************/
    p->state = CPROC_RUNNING;
    p->busy = 0;
    /*
     * In GNU, every cthread must be wired.  So we just
     * initialize P->wired on creation.
     *
```

```
     * A wired thread has a port associated with it for all
     * of its wait/block cases.  We also prebuild a wakeup
     * message.
     */
    MACH_CALL(mach_port_allocate(mach_task_self(),
                                 MACH_PORT_RIGHT_RECEIVE,
                                 &p->wired), r);
    MACH_CALL(mach_port_insert_right(mach_task_self(),
                                     p->wired, p->wired,
                                     MACH_MSG_TYPE_MAKE_SEND),
              r);
    p->msg.msgh_bits =
        MACH_MSGH_BITS(MACH_MSG_TYPE_COPY_SEND, 0);
    p->msg.msgh_size = 0;       /* initialized in call */
    p->msg.msgh_remote_port = p->wired;
    p->msg.msgh_local_port = MACH_PORT_NULL;
    p->msg.msgh_kind = MACH_MSGH_KIND_NORMAL;
    p->msg.msgh_id = 0;
    spin_lock(&cproc_list_lock);
    p->list = cproc_list;
    cproc_list = p;
    spin_unlock(&cproc_list_lock);
    mutex_unlock(&sched_queue_lock);
    return p;
}
/*
 * Called by cthread_init to set up initial data structures.
 */
vm_offset_t cproc_init()
{
    kern_return_t r;
    cproc_t p = cproc_alloc(1);
    cthread_kernel_threads = 1;
    MACH_CALL(mach_port_allocate(mach_task_self(),
                                 MACH_PORT_RIGHT_RECEIVE,
                                 &wait_port), r);
    MACH_CALL(mach_port_insert_right(mach_task_self(),
                                     wait_port, wait_port,
                                     MACH_MSG_TYPE_MAKE_SEND),
              r);
    wakeup_msg.msgh_bits =
        MACH_MSGH_BITS(MACH_MSG_TYPE_COPY_SEND, 0);
    wakeup_msg.msgh_size = 0;   /* initialized in call */
    wakeup_msg.msgh_remote_port = wait_port;
    wakeup_msg.msgh_local_port = MACH_PORT_NULL;
    wakeup_msg.msgh_kind = MACH_MSGH_KIND_NORMAL;
    wakeup_msg.msgh_id = 0;
    cprocs_started = TRUE;

    /*
     * We pass back the new stack which should be switched to
     * by crt0.  This guarantess correct size and alignment.
     */
    return (stack_init(p));
}
/*
 * Insert cproc on ready queue.  Make sure it is ready for queue by
 * synching on its lock.  Just send message to wired cproc.
 */
private int cproc_ready(p, preq)
```

```
register cproc_t p;
register int preq;
{
    kern_return_t r;

    r = mach_msg(&p->msg, MACH_SEND_MSG,
                 sizeof p->msg, 0, MACH_PORT_NULL,
                 MACH_MSG_TIMEOUT_NONE, MACH_PORT_NULL);
#ifdef CHECK_STATUS
    if (r != MACH_MSG_SUCCESS) {
        mach_error("mach_msg", r);
        exit(1);
    }
#endif                          /* CHECK_STATUS */
    return TRUE;
}

/*
 * This is only run on a partial "waiting" stack and called from
 * cproc_start_wait
 */

void cproc_waiting(p)
register cproc_t p;
{
    mach_msg_header_t msg;
    register cproc_t new;
    kern_return_t r;

#ifdef STATISTICS
    spin_lock(&ready_lock);
    cthread_waiting++;
    cthread_waiters++;
    spin_unlock(&ready_lock);
#endif                          /* STATISTICS */
    for (;;) {
        MACH_CALL(mach_msg(&msg, MACH_RCV_MSG,
                           0, sizeof msg, wait_port,
                           MACH_MSG_TIMEOUT_NONE,
                           MACH_PORT_NULL), r);
        spin_lock(&ready_lock);
        cthread_queue_deq(&ready, cproc_t, new);
        if (new != NO_CPROC)
            break;
        wait_count++;
        ready_count++;
#ifdef STATISTICS
        cthread_none++;
#endif                          /* STATISTICS */
        spin_unlock(&ready_lock);
    }
#ifdef STATISTICS
    cthread_ready--;
    cthread_running++;
    cthread_waiting--;
#endif                          /* STATISTICS */
    spin_unlock(&ready_lock);
    spin_lock(&new->lock);
    new->state = CPROC_RUNNING;
    spin_unlock(&new->lock);
    spin_lock(&waiters_lock);
    cthread_queue_enq(&waiters, p);
    spin_lock(&p->lock);
    spin_unlock(&waiters_lock);
    cproc_switch(&p->context, &new->context, &p->lock);
}

/*
 * Get a waiter with stack
```

```
 *
 */
cproc_t cproc_waiter()
{
    register cproc_t waiter;

    spin_lock(&waiters_lock);
    cthread_queue_deq(&waiters, cproc_t, waiter);
    spin_unlock(&waiters_lock);
    if (waiter == NO_CPROC) {
        vm_address_t base;
        kern_return_t r;
#ifdef STATISTICS
        spin_lock(&waiters_lock);
        cthread_wait_stacks++;
        spin_unlock(&waiters_lock);
#endif                          /* STATISTICS */
        waiter = cproc_alloc(1);
        MACH_CALL(vm_allocate(mach_task_self(), &base,
                              cthread_wait_stack_size, TRUE), r);
        waiter->stack_base = base;
        waiter->stack_size = cthread_wait_stack_size;
    }
    return (waiter);
}

/*
 * Current cproc is blocked so switch to any ready cprocs, or, if
 * none, go into the wait state.
 *
 * You must hold cproc_self()->lock when called.
 */
cproc_block()
{
    extern unsigned int __hurd_threadvar_max;   /* GNU */
    register cproc_t waiter, new, p = cproc_self();
    register int extra;

    mach_msg_header_t msg;
    kern_return_t r;

    spin_unlock(&p->lock);
    MACH_CALL(mach_msg(&msg, MACH_RCV_MSG,
                       0, sizeof msg, p->wired,
                       MACH_MSG_TIMEOUT_NONE, MACH_PORT_NULL),
              r);
    return;
}

/*
 * Implement C threads using MACH threads.
 */
cproc_t cproc_create(cthread_t t, int f)
{
    register cproc_t child = cproc_alloc(f);
    register kern_return_t r;
    extern void cproc_setup();
    extern void cproc_prepare();
    extern void cthread_body();
    thread_t n;

    alloc_stack(child);
    spin_lock(&n_kern_lock);
    cthread_kernel_threads++;
    spin_unlock(&n_kern_lock);
    MACH_CALL(thread_create(mach_task_self(), &n), r);
    cthread_assoc(child, t);    /* associate cthread with cproc */
    child->mach_thread = n;
```

```
    cproc_setup(child, n, cthread_body);        /* machine dependent */
    if (!user_init)
        MACH_CALL(thread_resume(n), r);
    if (!f) {
        MACH_CALL(thread_resume(n), r);
    }
#ifdef STATISTICS
    spin_lock(&ready_lock);
    cthread_running++;
    spin_unlock(&ready_lock);
#endif                          /* STATISTICS */
    return child;
}

void condition_wait(c, m)
register condition_t c;
mutex_t m;
{
    register cproc_t p = cproc_self();

    p->state = CPROC_CONDWAIT | CPROC_SWITCHING;

    spin_lock(&c->lock);
    cthread_queue_enq(&c->queue, p);
    spin_unlock(&c->lock);
#ifdef WAIT_DEBUG
    p->waiting_for = (char *) c;
#endif                          /* WAIT_DEBUG */

    mutex_unlock(m);

    spin_lock(&p->lock);
    if (p->state & CPROC_SWITCHING) {
        cproc_block();
    } else {
        p->state = CPROC_RUNNING;
        spin_unlock(&p->lock);
    }


#ifdef WAIT_DEBUG
    p->waiting_for = (char *) 0;
#endif                          /* WAIT_DEBUG */

    /*
     * Re-acquire the mutex and return.
     */
    mutex_lock(m);
}

/* Declare that IMPLICATOR should consider IMPLICATAND's waiter queue
   to be an extension of its own queue.  It is an error for either
   condition to be deallocated as long as the implication persists. */
void
condition_implies(condition_t implicator,
                  condition_t implicatand)
{
    struct cond_imp *imp;

    imp = malloc(sizeof(struct cond_imp));
    imp->implicatand = implicatand;
    imp->next = implicator->implications;
    implicator->implications = imp;
}

/* Declare that the implication relationship from IMPLICATOR to
   IMPLICATAND should cease. */
void
condition_unimplies(condition_t implicator,
                    condition_t implicatand)
{
```

```
    struct cond_imp **impp;

    for (impp = &implicator->implications; *impp;
         impp = &(*impp)->next) {
        if ((*impp)->implicatand == implicatand) {
            struct cond_imp *tmp = *impp;
            *impp = (*impp)->next;
            free(tmp);
            return;
        }
    }
}

/* Signal one waiter on C.  If there were no waiters at all, return
   0, else return 1. */
int cond_signal(c)
register condition_t c;
{
    register cproc_t p;
    struct cond_imp *imp;

    spin_lock(&c->lock);
    cthread_queue_deq(&c->queue, cproc_t, p);
    spin_unlock(&c->lock);
    if (p != NO_CPROC) {
        cproc_ready(p, 0);
        return 1;
    } else {
        for (imp = c->implications; imp; imp = imp->next)
            if (cond_signal(imp->implicatand))
                return 1;
    }
    return 0;
}

void cond_broadcast(c)
register condition_t c;
{
    register cproc_t p;
    struct cthread_queue blocked_queue;
    struct cond_imp *imp;

    cthread_queue_init(&blocked_queue);

    spin_lock(&c->lock);
    for (;;) {
        register int old_state;

        cthread_queue_deq(&c->queue, cproc_t, p);
        if (p == NO_CPROC)
            break;
        cthread_queue_enq(&blocked_queue, p);
    }
    spin_unlock(&c->lock);

    for (;;) {
        cthread_queue_deq(&blocked_queue, cproc_t, p);
        if (p == NO_CPROC)
            break;
        cproc_ready(p, 0);
    }

    for (imp = c->implications; imp; imp = imp->next)
        condition_broadcast(imp->implicatand);
}

void cthread_yield()
{
    yield();
    return;
```

```
}
/*
 * Mutex objects.
 */
void __mutex_lock_solid(void *ptr)
{
    register mutex_t m = ptr;
    register cproc_t p = cproc_self();
    register int queued;
    register int tried = 0;
#ifdef WAIT_DEBUG
    p->waiting_for = (char *) m;
#endif                          /* WAIT_DEBUG */
    while (1) {
        spin_lock(&m->lock);
        if (cthread_queue_head(&m->queue, cproc_t) == NO_CPROC) {
            cthread_queue_enq(&m->queue, p);
            queued = 1;
        } else {
            queued = 0;
        }
        if (spin_try_lock(&m->held)) {
            if (queued)
                cthread_queue_deq(&m->queue, cproc_t, p);
            spin_unlock(&m->lock);
#ifdef WAIT_DEBUG
            p->waiting_for = (char *) 0;
#endif                          /* WAIT_DEBUG */
            return;
        } else {
            if (!queued)
                cthread_queue_enq(&m->queue, p);
            spin_lock(&p->lock);
            spin_unlock(&m->lock);
            cproc_block();
            if (spin_try_lock(&m->held)) {
#ifdef WAIT_DEBUG
                p->waiting_for = (char *) 0;
#endif                          /* WAIT_DEBUG */
                return;
            }
#ifdef STATISTICS
            spin_lock(&mutex_count_lock);
            cthread_no_mutex++;
            spin_unlock(&mutex_count_lock);
#endif                          /* STATISTICS */
        }
    }
}
void __mutex_unlock_solid(void *ptr)
{
    register mutex_t m = ptr;
    register cproc_t new;
    if (!spin_try_lock(&m->held))
        return;
    spin_lock(&m->lock);
    cthread_queue_deq(&m->queue, cproc_t, new);
    spin_unlock(&m->held);
    spin_unlock(&m->lock);
    if (new) {
        cproc_ready(new, 0);
    }
}
```

```
/***********Scheduler related code**************************/
extern int ready_threads;
/*cthread status codes*/
#define T_MAIN 0x1
#define T_RETURNED 0x2
#define T_DETACHED 0x4
any_t round_robin(any_t param)
{
    register int dt = 0, time =
        ((int) param > 0) ? ((int) param) : 100;
    cproc_t proc, rdproc, nrproc;
    register kern_return_t r;
    static ttimer t = NULL_TIMER, tdum;

    if (t.tv_sec == 0 && t.tv_usec == 0) {
        refresh_timer(&t);
    }
    refresh_timer(&tdum);
    if ((dt = sub_timer(&t, &tdum) / 1000) >= time) {
        t = tdum;
        mutex_lock(&sched_queue_lock);
        if ((proc = running_cproc) != NO_CPROC) {
            /* Suspend currently running cproc and enqueue if not finished
             */
            if (!
                (running_cproc->incarnation->state & T_RETURNED))
                sched_queue_enq(&sched_cprocs_ready, running_cproc);    /* Insert
                                                                           back in
                                                                           queue so
                                                                           we can
                                                                           select it
                                                                           next time
                                                                         */
            while (proc != NULL) {      /* Also suspend other threads in
                                           this cproc's group */
                MACH_CALL(thread_suspend(proc->mach_thread), r);
                proc = proc->group_next;
            }
        }
        sched_queue_deq(&sched_cprocs_ready, running_cproc);
        if ((proc = running_cproc) != NO_CPROC) {       /* This will be
                                                           false when
                                                           there are no
                                                           cthreads
                                                           created */
            while (proc != NULL) {      /* Resume all threads in this
                                           cproc's group */
                MACH_CALL(thread_resume(proc->mach_thread), r);
                proc = proc->group_next;
            }
        }
        mutex_unlock(&sched_queue_lock);
    } else {
        time = time - dt;
    }
    schedyield(((running_cproc !=
                 NO_CPROC) ? running_cproc->
                mach_thread : MACH_PORT_NULL), time);
    return param;
}

any_t first_come(any_t param)
{
    register int time = ((int) param) ? ((int) param) : 100;
    register kern_return_t r;
    register cproc_t proc;
```

```
    mutex_lock(&sched_queue_lock);
    if ((proc = running_cproc) != NO_CPROC
        && running_cproc->incarnation->state & T_RETURNED) {
        /* running_cproc has finished suspend it */
        MACH_CALL(thread_suspend(running_cproc->mach_thread), r);
        while (proc->group_next != NULL) {      /* Also suspend other
                                                   threads in this cproc's
                                                   group */
            proc = proc->group_next;
            MACH_CALL(thread_suspend(proc->mach_thread), r);
        }
        running_cproc = NO_CPROC;
    }
    if (running_cproc == NO_CPROC) {
        /* We don't have a running cproc so we invoke the next one */
        sched_queue_deq(&sched_cprocs_ready, running_cproc);
        if ((proc = running_cproc) != NO_CPROC) {
            MACH_CALL(thread_resume(running_cproc->mach_thread),
                      r);
            while (proc->group_next != NULL) {  /* Resume all threads in
                                                   this cproc's group */
                proc = proc->group_next;
                MACH_CALL(thread_resume(proc->mach_thread), r);
            }
        }
    }
    mutex_unlock(&sched_queue_lock);
    schedyield(((running_cproc !=
                 NO_CPROC) ? running_cproc->
                mach_thread : MACH_PORT_NULL), time);
    return param;
}

/*This is real old don't pay attention at it*/
#if 0
any_t shortest_job(any_t param)
{
    int time = ((int) param) ? ((int) param) : 100;
    kern_return_t r;
    register cproc_t i = NO_CPROC, min = NO_CPROC;
    static ttimer t = NULL_TIMER, tdum;
    int dt = 0;

    if (t.tv_sec == 0 && t.tv_usec == 0) {
        refresh_timer(&t);
    }
    refresh_timer(&tdum);
    if ((dt = sub_timer(&t, &tdum) / 1000) >= time) {
        t = tdum;
        mutex_lock(&sched_queue_lock);
        sched_queue_deq(&sched_cprocs_ready, i);
        for (min = i; i != NO_CPROC; i = i->sched_next) {
            if (i->incarnation->state & T_RETURNED)
                continue;
            else if ((*(int *) (i->incarnation->sched_data)) <
                     (*(int *) (min->incarnation->sched_data))) {
                sched_queue_enq(&sched_cprocs_ready, min);      /* Enqueue
                                                                   previous
                                                                   minimum */
                min = i;
            } else
                sched_queue_enq(&sched_cprocs_ready, i);        /* Re-enqueue
                                                                   if not
                                                                   minimum */
        }
        if (running_cproc != NO_CPROC) {
```

```
            MACH_CALL(thread_suspend(running_cproc->mach_thread),
                      r);
            if (!
                (running_cproc->incarnation->state & T_RETURNED))
                sched_queue_enq(&sched_cprocs_ready, running_cproc);    /* Insert
                                                                           back in
                                                                           queue so
                                                                           we can
                                                                           select it
                                                                           next time
                                                                         */
        }
        if (min != NO_CPROC) {
            MACH_CALL(thread_resume(min->mach_thread), r);
            running_cproc = min;
        }
        mutex_unlock(&sched_queue_lock);
    } else {
        time = time - dt;
    }
    schedyield(((min !=
                 NO_CPROC) ? min->mach_thread : MACH_PORT_NULL),
               time);
    return param;
}
#endif
/*************************************************************************************/
cproc_fork_prepare()
{
    register cproc_t p = cproc_self();

    vm_inherit(mach_task_self(), p->stack_base, p->stack_size,
               VM_INHERIT_COPY);
    spin_lock(&port_lock);
    spin_lock(&cproc_list_lock);
}

cproc_fork_parent()
{
    register cproc_t p = cproc_self();

    spin_unlock(&cproc_list_lock);
    spin_unlock(&port_lock);
    vm_inherit(mach_task_self(), p->stack_base, p->stack_size,
               VM_INHERIT_NONE);
}

cproc_fork_child()
{
    register cproc_t l, p = cproc_self();
    cproc_t m;
    register port_entry_t pe;
    port_entry_t pet;
    kern_return_t r;

    vm_inherit(mach_task_self(), p->stack_base, p->stack_size,
               VM_INHERIT_NONE);
    spin_lock_init(&n_kern_lock);
    cthread_kernel_threads = 0;
#ifdef STATISTICS
    cthread_ready = 0;
    cthread_running = 1;
    cthread_waiting = 0;
    cthread_wired = 0;
    spin_lock_init(&wired_lock);
    cthread_wait_stacks = 0;
    cthread_waiters = 0;
    cthread_wakeup = 0;
```

```
    cthread_blocked = 0;
    cthread_rnone = 0;
    cthread_yields = 0;
    cthread_none = 0;
    cthread_switches = 0;
    cthread_no_mutex = 0;
    spin_lock_init(&mutex_count_lock);
#endif                          /* STATISTICS */

    for (l = cproc_list; l != NO_CPROC; l = m) {
        m = l->next;
        if (l != p)
            free(l);
    }

    cproc_list = p;
    p->next = NO_CPROC;
    spin_lock_init(&cproc_list_lock);
    cprocs_started = FALSE;
    cthread_queue_init(&ready);
    ready_count = 0;
    spin_lock_init(&ready_lock);

    MACH_CALL(mach_port_allocate(mach_task_self(),
                                 MACH_PORT_RIGHT_RECEIVE,
                                 &wait_port), r);
    MACH_CALL(mach_port_insert_right(mach_task_self(),
                                     wait_port, wait_port,
                                     MACH_MSG_TYPE_MAKE_SEND),
              r);
    wakeup_msg.msgh_remote_port = wait_port;
    wait_count = 0;
    cthread_queue_init(&waiters);
    spin_lock_init(&waiters_lock);
    for (pe = port_list; pe != PORT_ENTRY_NULL; pe = pet) {
        pet = pe->next;
        free(pe);
    }
    port_list = PORT_ENTRY_NULL;
    spin_lock_init(&port_lock);

    if (p->wired)
        cthread_wire();
}
```

B.3 libthreads/cthread_internals.h

```
/*
 * Mach Operating System
 * Copyright (c) 1991,1990,1989 Carnegie Mellon University
 * All Rights Reserved.
 *
 * Permission to use, copy, modify and distribute this software and its
 * documentation is hereby granted, provided that both the copyright
 * notice and this permission notice appear in all copies of the
 * software, derivative works or modified versions, and any portions
 * thereof, and that both notices appear in supporting documentation.
 *
 * CARNEGIE MELLON ALLOWS FREE USE OF THIS SOFTWARE IN ITS "AS IS"
 * CONDITION.  CARNEGIE MELLON DISCLAIMS ANY LIABILITY OF ANY KIND FOR
 * ANY DAMAGES WHATSOEVER RESULTING FROM THE USE OF THIS SOFTWARE.
 *
 * Carnegie Mellon requests users of this software to return to
 *
 *  Software Distribution Coordinator  or  Software.Distribution@CS.CMU.EDU
 *  School of Computer Science
 *  Carnegie Mellon University
 *  Pittsburgh PA 15213-3890
```

```
 *
 * any improvements or extensions that they make and grant Carnegie Mellon
 * the rights to redistribute these changes.
 */
/*
 * HISTORY
 * $Log: cthread_internals.h,v $
 * Revision 1.3  1995/08/29 14:49:20  mib
 * (cproc_block): Provide decl.
 *
 * Revision 1.2  1994/05/05 10:58:01  roland
 * entered into RCS
 *
 * Revision 2.14  92/08/03  18:03:56  jfriedl
 *  Made state element of struct cproc volatile.
 *  [92/08/02            jfriedl]
 *
 * Revision 2.13  92/03/06  14:09:24  rpd
 *  Added yield, defined using thread_switch.
 *  [92/03/06            rpd]
 *
 * Revision 2.12  92/03/01  00:40:23  rpd
 *  Removed exit declaration.  It conflicted with the real thing.
 *  [92/02/29            rpd]
 *
 * Revision 2.11  91/08/28  11:19:23  jsb
 *  Fixed MACH_CALL to allow multi-line expressions.
 *  [91/08/23            rpd]
 *
 * Revision 2.10  91/07/31  18:33:33  dbg
 *  Protect against redefinition of ASSERT.
 *  [91/07/30  17:33:21  dbg]
 *
 * Revision 2.9  91/05/14  17:56:24  mrt
 *  Correcting copyright
 *
 * Revision 2.8  91/02/14  14:19:42  mrt
 *  Added new Mach copyright
 *  [91/02/13  12:41:02  mrt]
 *
 * Revision 2.7  90/11/05  14:36:55  rpd
 *  Added spin_lock_t.
 *  [90/10/31            rwd]
 *
 * Revision 2.6  90/09/09  14:34:51  rpd
 *  Remove special field.
 *  [90/08/24            rwd]
 *
 * Revision 2.5  90/06/02  15:13:44  rpd
 *  Converted to new IPC.
 *  [90/03/20  20:52:47  rpd]
 *
 * Revision 2.4  90/03/14  21:12:11  rwd
 *  Added waiting_for field for debugging deadlocks.
 *  [90/03/01            rwd]
 *  Added list field to keep a master list of all cprocs.
 *  [90/03/01            rwd]
 *
 * Revision 2.3  90/01/19  14:37:08  rwd
 *  Keep track of real thread for use in thread_* substitutes.
 *  Add CPROC_ARUN for about to run and CPROC_HOLD to avoid holding
 *  spin_locks over system calls.
 *  [90/01/03            rwd]
 *  Add busy field to be used by cthread_msg calls to make sure we
 *  have the right number of blocked kernel threads.
 *  [89/12/21            rwd]
 *
 * Revision 2.2  89/12/08  19:53:28  rwd
 *  Added CPROC_CONDWAIT state
 *  [89/11/28            rwd]
 *  Added on_special field.
```

```
 *  [89/11/26            rwd]
 *  Removed MSGOPT conditionals
 *  [89/11/25            rwd]
 *  Removed old debugging code.  Add wired port/flag.  Add state
 *  for small state machine.
 *  [89/10/30            rwd]
 *  Added CPDEBUG code
 *  [89/10/26            rwd]
 *  Change TRACE to {x;} else.
 *  [89/10/24            rwd]
 *  Rewrote to work for limited number of kernel threads.  This is
 *  basically a merge of coroutine and thread.  Added
 *  cthread_receivce call for use by servers.
 *  [89/10/23            rwd]
 *
 */
/*
 * cthread_internals.h
 *
 *
 * Private definitions for the C Threads implementation.
 *
 * The cproc structure is used for different implementations
 * of the basic schedulable units that execute cthreads.
 *
 */

#include "options.h"
#include <mach/port.h>
#include <mach/message.h>
#include <mach/thread_switch.h>

#if !defined(__STDC__) && !defined(volatile)
# ifdef __GNUC__
#   define volatile __volatile__
# else
#   define volatile             /* you lose */
# endif
#endif

/*
 * Low-level thread implementation.
 * This structure must agree with struct ur_cthread in cthreads.h
 */
typedef struct cproc {
    struct cproc *next;         /* for lock, condition, and ready queues */
    cthread_t incarnation;      /* for cthread_self() */

    struct cproc *list;         /* for master cproc list */
#ifdef WAIT_DEBUG
    volatile char *waiting_for; /* address of mutex/cond waiting for */
#endif                          /* WAIT_DEBUG */

#if 0
    /* This is not needed in GNU; libc handles it.  */
    mach_port_t reply_port;     /* for mig_get_reply_port() */
#endif

    int context;
    spin_lock_t lock;
    volatile int state;         /* current state */
#define CPROC_RUNNING 0
#define CPROC_SWITCHING 1
#define CPROC_BLOCKED 2
#define CPROC_CONDWAIT 4

    mach_port_t wired;          /* is cthread wired to kernel thread */
    int busy;                   /* used with cthread_msg calls */

    mach_msg_header_t msg;

    unsigned int stack_base;
    unsigned int stack_size;
/*****Kinalis***********//* for use by the scheduler */
```

```
    struct cproc *sched_next;   /* For entering scheduling queue */
    thread_t mach_thread;       /* mach thread this cproc is attached to */
    struct cproc *group_next;   /* pointer to next cproc in group */
        /**********************/
} *cproc_t;

#define NO_CPROC ((cproc_t) 0)
#define cproc_self() ((cproc_t) ur_cthread_self())

int cproc_block();

#if 0
/* This declaration conflicts with <stdlib.h> in GNU.  */
/*
 * C Threads imports:
 */
extern char *malloc();
#endif

/*
 * Mach imports:
 */
extern void mach_error();

/*
 * Macro for MACH kernel calls.
 */
#ifdef CHECK_STATUS
#define MACH_CALL(expr, ret) \
if (((ret) = (expr)) != KERN_SUCCESS) { \
quit(1, "error in %s at %d: %s\n", __FILE__, __LINE__, \
     mach_error_string(ret)); \
} else
#else                           /* CHECK_STATUS */
#define MACH_CALL(expr, ret) (ret) = (expr)
#endif                          /* CHECK_STATUS */

#define private static
#ifndef ASSERT
#define ASSERT(x)
#endif
#define TRACE(x)

/*
 * What we do to yield the processor:
 * (This depresses the thread's priority for up to 10ms.)
 */

#define yield() \
(void) thread_switch(MACH_PORT_NULL, SWITCH_OPTION_DEPRESS, 10)
/*************************/
#define schedyield(n,t) \
(void) thread_switch((n), SWITCH_OPTION_WAIT, (t))
```

B.4 libthreads/cthreads.c

```
/*
 * Mach Operating System
 * Copyright (c) 1991,1990,1989 Carnegie Mellon University
 * All Rights Reserved.
 *
 * Permission to use, copy, modify and distribute this software and its
 * documentation is hereby granted, provided that both the copyright
 * notice and this permission notice appear in all copies of the
 * software, derivative works or modified versions, and any portions
 * thereof, and that both notices appear in supporting documentation.
 *
 * CARNEGIE MELLON ALLOWS FREE USE OF THIS SOFTWARE IN ITS "AS IS"
 * CONDITION.  CARNEGIE MELLON DISCLAIMS ANY LIABILITY OF ANY KIND FOR
 * ANY DAMAGES WHATSOEVER RESULTING FROM THE USE OF THIS SOFTWARE.
 *
```

```
 * Carnegie Mellon requests users of this software to return to
 *
 *  Software Distribution Coordinator  or  Software.Distribution@CS.CMU.EDU
 *  School of Computer Science
 *  Carnegie Mellon University
 *  Pittsburgh PA 15213-3890
 *
 * any improvements or extensions that they make and grant Carnegie Mellon
 * the rights to redistribute these changes.
 */
/*
 *
 *  Kinalis Athanasios 18/3/2001
 *
 *
 *
 * HISTORY
 * $Log: cthreads.c,v $
 * Revision 1.9  1998/11/22 18:18:10  roland
 * 1998-11-12  Mark Kettenis  <kettenis@phys.uva.nl>
 *
 *  * cthreads.c (cthread_init): Move cthread_alloc call before
 *  cproc_init call, since cthread_alloc uses malloc, and malloc won't
 *  work between initializing the new stack and switching over to it.
 *
 * Revision 1.8  1998/06/10 19:38:01  tb
 * Tue Jun  9 13:50:09 1998  Thomas Bushnell, n/BSG  <tb@mit.edu>
 *
 *  * cthreads.c (cthread_fork_prepare): Don't call
 *  malloc_fork_prepare since we are no longer providing our own
 *  malloc in this library.
 *  (cthread_fork_parent): Likewise, for malloc_fork_parent.
 *  (cthread_fork_child): Likewize, for malloc_fork_child.
 *
 * Revision 1.7  1997/08/20 19:41:20  thomas
 * Wed Aug 20 15:39:44 1997  Thomas Bushnell, n/BSG  <thomas@gnu.ai.mit.edu>
 *
 *  * cthreads.c (cthread_body): Wire self before calling user work
 *   function.  This way all cthreads will be wired, which the ports
 *  library (and hurd_thread_cancel, etc.) depend on.
 *
 * Revision 1.6  1997/06/10 01:22:19  thomas
 * Mon Jun  9 21:18:46 1997  Thomas Bushnell, n/BSG  <thomas@gnu.ai.mit.edu>
 *
 *  * cthreads.c (cthread_fork): Delete debugging oddity that crept
 *  into source.
 *
 * Revision 1.5  1997/04/04 01:30:35  thomas
 * *** empty log message ***
 *
 * Revision 1.4  1994/05/05 18:13:57  roland
 * entered into RCS
 *
 * Revision 2.11  92/07/20  13:33:37  cmaeda
 *  In cthread_init, do machine dependent initialization if it's defined.
 *  [92/05/11  14:41:08  cmaeda]
 *
 * Revision 2.10  91/08/28  11:19:26  jsb
 *  Fixed mig_init initialization in cthread_fork_child.
 *  [91/08/23            rpd]
 *
 * Revision 2.9  91/07/31  18:34:23  dbg
 *  Fix bad self-pointer reference.
 *
 *  Don't declare _setjmp and _longjmp; they are included by
 *  cthreads.h.
 *  [91/07/30  17:33:50  dbg]
 *
 * Revision 2.8  91/05/14  17:56:31  mrt
 *  Correcting copyright
 *
 * Revision 2.7  91/02/14  14:19:47  mrt
 *  Added new Mach copyright
 *  [91/02/13  12:41:07  mrt]
```

```
 *
 * Revision 2.6  90/11/05  14:37:03  rpd
 *  Added cthread_fork_{prepare,parent,child}.
 *  [90/11/02            rwd]
 *
 *  Add spin_lock_t.
 *  [90/10/31            rwd]
 *
 * Revision 2.5  90/08/07  14:30:58  rpd
 *  Removed RCS keyword nonsense.
 *
 * Revision 2.4  90/06/02  15:13:49  rpd
 *  Converted to new IPC.
 *  [90/03/20  20:56:44  rpd]
 *
 * Revision 2.3  90/01/19  14:37:12  rwd
 *  Make cthread_init return pointer to new stack.
 *  [89/12/18  19:17:45  rwd]
 *
 * Revision 2.2  89/12/08  19:53:37  rwd
 *  Change cproc and cthread counters to globals with better names.
 *  [89/11/02            rwd]
 *
 * Revision 2.1  89/08/03  17:09:34  rwd
 * Created.
 *
 *
 * 31-Dec-87  Eric Cooper (ecc) at Carnegie Mellon University
 * Changed cthread_exit() logic for the case of the main thread,
 * to fix thread and stack memory leak found by Camelot group.
 *
 * 21-Aug-87  Eric Cooper (ecc) at Carnegie Mellon University
 * Added consistency check in beginning of cthread_body().
 *
 * 11-Aug-87  Eric Cooper (ecc) at Carnegie Mellon University
 * Removed cthread_port() and cthread_set_port().
 * Removed port deallocation from cthread_free().
 * Minor changes to cthread_body(), cthread_exit(), and cthread_done().
 *
 * 10-Aug-87  Eric Cooper (ecc) at Carnegie Mellon University
 * Changed call to mig_init() in cthread_init() to pass 1 as argument.
 *
 * 31-Jul-87  Eric Cooper (ecc) at Carnegie Mellon University
 * Added call to mig_init() from cthread_init().
 */
/*
 *  File: cthreads.c
 * Author: Eric Cooper, Carnegie Mellon University
 * Date: July, 1987
 *
 *  Implementation of fork, join, exit, etc.
 */
#include "cthreads.h"
#include "cthread_internals.h"
#define ECHO_ON 1               /* Define this to output to terminal */
#include "echoer.h"
/*
 * C Threads imports:
 */
extern cproc_t cproc_create(cthread_t, int);
extern vm_offset_t cproc_init();
extern void mig_init();

/***********Scheduler related code**************/
static int sched_counter = 0;
static sched_func schedfunc = NULL;     /* Holds the function that the
                                           scheduling algorithm */
static any_t schedparams = NULL;        /* Holds the parameter to be
                                           passed to the above function */
struct mutex scheduler_lock = MUTEX_INITIALIZER;        /* A lock to
```

```
                                                            access above
                                                            data */
int ready_threads = 0;          /* Used to count threads in ready queue */

/********Scheduling algorithms************/
/*Implemented in cprocs.c*/
extern any_t round_robin(any_t);
extern any_t first_come(any_t);
extern any_t shortest_job(any_t);
extern struct cthread_queue sched_cprocs_ready;
extern struct mutex sched_queue_lock;
extern struct cthread_queue cprocs_done;
extern struct mutex done_queue_lock;

extern cproc_t running_cproc;
extern cproc_t sched_cproc;
int schedon = FALSE;
int user_init = FALSE;
/***********End of scheduler related code**************/
/*
 * Mach imports:
 */

/*
 * C library imports:
 */

/*
 * Thread status bits.
 */
#define T_MAIN 0x1
#define T_RETURNED 0x2
#define T_DETACHED 0x4

#ifdef DEBUG
int cthread_debug = FALSE;
#endif                          /* DEBUG */

private struct cthread_queue cthreads = QUEUE_INITIALIZER;
private struct mutex cthread_lock = MUTEX_INITIALIZER;
private struct condition cthread_needed = CONDITION_INITIALIZER;
private struct condition cthread_idle = CONDITION_INITIALIZER;
int cthread_cprocs = 0;
int cthread_cthreads = 0;
int cthread_max_cprocs = 0;

private cthread_t free_cthreads = NO_CTHREAD;   /* free list */
private spin_lock_t free_lock = SPIN_LOCK_INITIALIZER;  /* unlocked */

private struct cthread initial_cthread = { 0 };

private cthread_t cthread_alloc(func, arg)
cthread_fn_t func;
any_t arg;
{
    register cthread_t t = NO_CTHREAD;

    if (free_cthreads != NO_CTHREAD) {
        /*
         * Don't try for the lock unless
         * the list is likely to be nonempty.
         * We can't be sure, though, until we lock it.
         */
        spin_lock(&free_lock);
        t = free_cthreads;
        if (t != NO_CTHREAD)
            free_cthreads = t->next;
        spin_unlock(&free_lock);
    }
    if (t == NO_CTHREAD) {
        /*
         * The free list was empty.
         * We may have only found this out after
```

```
         * locking it, which is why this isn't an
         * "else" branch of the previous statement.
         */
        t = (cthread_t) malloc(sizeof(struct cthread));
    }
    *t = initial_cthread;
    t->func = func;
    t->arg = arg;
    return t;
}

private void cthread_free(t)
register cthread_t t;
{
    spin_lock(&free_lock);
    t->next = free_cthreads;
    free_cthreads = t;
    spin_unlock(&free_lock);
}

int cthread_init()
{
    static int cthreads_started = FALSE;
    register cproc_t p;
    register cthread_t t;
    vm_offset_t stack;

    if (cthreads_started) {
        user_init = TRUE;
        return 0;
    }
    t = cthread_alloc((cthread_fn_t) 0, (any_t) 0);
    stack = cproc_init();
    cthread_cprocs = 1;
#ifdef cthread_md_init
    cthread_md_init();
#endif
    cthread_cthreads = 1;
    t->state |= T_MAIN;
    cthread_set_name(t, "main");

    /* cproc_self() doesn't work yet, because we haven't yet switched to
       the new stack. */

    p = *(cproc_t *) & ur_cthread_ptr(stack);
    p->incarnation = t;
    /* The original CMU code passes P to mig_init.  In GNU, mig_init does
       not know about cproc_t; instead it expects to be passed the stack
       pointer of the initial thread.  */
    mig_init((void *) stack);   /* enable multi-threaded mig interfaces */
    cthreads_started = TRUE;
    return stack;
}

/*
 * Used for automatic initialization by crt0.
 * Cast needed since too many C compilers choke on the type void (*)().
 */
int (*_cthread_init_routine) () = (int (*)()) cthread_init;

/*
 * Procedure invoked at the base of each cthread.
 */
extern cproc_block();

void cthread_body(self)
cproc_t self;
{
    register cthread_t t;
```

```
    static once = 0;
    cproc_t tmp, i;

    ASSERT(cproc_self() == self);
    TRACE(printf("[idle] cthread_body(%x)\n", self));
    t = self->incarnation;
    if (_setjmp(t->catch) == 0) /* catch for cthread_exit() */
        t->result = (*(t->func)) (t->arg);
    /*
     * Return result from thread.
     */
    mutex_lock(&t->lock);
    t->state |= T_RETURNED;

    mutex_lock(&sched_queue_lock);
    TRACE(printf("[%s] done()\n", cthread_name(t)));
    /* Remove current cproc from sched_cprocs_ready queue */
    if (t->state & T_DETACHED) {
        mutex_unlock(&t->lock);
        cthread_free(t);
    } else {
        mutex_unlock(&t->lock);
        condition_signal(&t->done);
    }
    mutex_unlock(&sched_queue_lock);
    /* Signal that we're idle in case the main thread * is waiting to exit
     */
    cthread_cthreads -= 1;
    condition_signal(&cthread_idle);
    spin_lock(&self->lock);     /* Lock before calling cproc_block() */
    cproc_block();
}

cthread_t cthread_fork(func, arg)
cthread_fn_t func;
any_t arg;
{
    register cthread_t t;
    cproc_t cproc;
    kern_return_t r;

    mutex_lock(&cthread_lock);
    if (!schedon) {             /* First start the scheduler */
        t = cthread_alloc(sched, NULL);
        ++cthread_cthreads;
        cthread_cprocs += 1;
        cproc = cproc_create(t, 1);
        schedon = TRUE;
    }
    TRACE(printf("[%s] fork()\n", cthread_name(cthread_self())));
    t = cthread_alloc(func, arg);
    ++cthread_cthreads;
    cthread_cprocs += 1;
    cproc = cproc_create(t, 1);
    mutex_unlock(&cthread_lock);
    return t;
}

cthread_t cthread_spawn(func, arg)
cthread_fn_t func;
any_t arg;
{
    register cthread_t t;
    cproc_t cproc;
    kern_return_t r;

    mutex_lock(&cthread_lock);
    if (!schedon) {             /* First start the scheduler */
        t = cthread_alloc(sched, NULL);
```

```
        ++cthread_cthreads;
        cthread_cprocs += 1;
        cproc = cproc_create(t, 0);
        schedon = TRUE;
    }
    TRACE(printf("[%s] fork()\n", cthread_name(cthread_self())));
    t = cthread_alloc(func, arg);
    ++cthread_cthreads;
    cthread_cprocs += 1;
    cproc = cproc_create(t, 0);
    mutex_unlock(&cthread_lock);
    return t;
}

void cthread_detach(t)
cthread_t t;
{
    TRACE(printf
          ("[%s] detach(%s)\n", cthread_name(cthread_self()),
           cthread_name(t)));
    mutex_lock(&t->lock);
    if (t->state & T_RETURNED) {
        mutex_unlock(&t->lock);
        cthread_free(t);
    } else {
        t->state |= T_DETACHED;
        mutex_unlock(&t->lock);
    }
}

any_t cthread_join(t)
cthread_t t;
{
    any_t result;

    TRACE(printf
          ("[%s] join(%s)\n", cthread_name(cthread_self()),
           cthread_name(t)));
    mutex_lock(&t->lock);
    ASSERT(!(t->state & T_DETACHED));
    while (!(t->state & T_RETURNED))
        condition_wait(&t->done, &t->lock);
    result = t->result;
    mutex_unlock(&t->lock);
    cthread_free(t);
    return result;
}

void cthread_exit(result)
any_t result;
{
    register cthread_t t = cthread_self();

    TRACE(printf("[%s] exit()\n", cthread_name(t)));
    t->result = result;
    if (t->state & T_MAIN) {
        mutex_lock(&cthread_lock);
        while (cthread_cthreads > 1)
            condition_wait(&cthread_idle, &cthread_lock);
        mutex_unlock(&cthread_lock);
        exit((int) result);
    } else {
        _longjmp(t->catch, TRUE);
    }
}

/*
 * Used for automatic finalization by crt0.  Cast needed since too many C
 * compilers choke on the type void (*)().
```

```
 */
int (*_cthread_exit_routine) () = (int (*)()) cthread_exit;

void cthread_set_name(t, name)
cthread_t t;
char *name;
{
    t->name = name;
}

char *cthread_name(t)
cthread_t t;
{
    return (t == NO_CTHREAD
            ? "idle" : (t->name == 0 ? "?" : t->name));
}

int cthread_limit()
{
    return cthread_max_cprocs;
}

void cthread_set_limit(n)
int n;
{
    cthread_max_cprocs = n;
}

int cthread_count()
{
    return cthread_cthreads;
}

cthread_fork_prepare()
{
    spin_lock(&free_lock);
    mutex_lock(&cthread_lock);
    cproc_fork_prepare();
}

cthread_fork_parent()
{
    cproc_fork_parent();
    mutex_unlock(&cthread_lock);
    spin_unlock(&free_lock);
}

cthread_fork_child()
{
    cthread_t t;
    cproc_t p;

    cproc_fork_child();
    mutex_unlock(&cthread_lock);
    spin_unlock(&free_lock);
    condition_init(&cthread_needed);
    condition_init(&cthread_idle);

    cthread_max_cprocs = 0;

    stack_fork_child();

    while (TRUE) {              /* Free cthread runnable list */
        cthread_queue_deq(&cthreads, cthread_t, t);
        if (t == NO_CTHREAD)
            break;
        free((char *) t);
    }

    while (free_cthreads != NO_CTHREAD) {       /* Free cthread free list */
        t = free_cthreads;
        free_cthreads = free_cthreads->next;
        free((char *) t);
    }
```

```
    cthread_cprocs = 1;
    t = cthread_self();
    cthread_cthreads = 1;
    t->state |= T_MAIN;
    cthread_set_name(t, "main");

    p = cproc_self();
    p->incarnation = t;
    /* XXX needs hacking for GNU */
    mig_init(p);                /* enable multi-threaded mig interfaces */
}

/***********Scheduler related code*************************/

/*Scheduler thread.
 *Just calls the scheduling algorithm.
 *Maybe we should call the algorithm function from cthread_body directly or
 *use this instead of cthread_body.
 */
any_t sched(any_t t)
{
    register cproc_t p = cproc_self();
    register cproc_t i;
    register kern_return_t r;

    while (1) {
        mutex_lock(&scheduler_lock);    /* Not very successful lock if we
                                           want to change scheduler
                                           dynamically? */
        if (schedfunc != NULL) {
            schedfunc(t ? t : schedparams);
            mutex_unlock(&scheduler_lock);
        } else {
            mutex_unlock(&scheduler_lock);
            schedyield(MACH_PORT_NULL, 50);
        }
    }
}

/*Sets the scheduling algorithm to be called. It can a user defined one but
 *probably that won't be used. Better call this from the main thread*/

int set_sched(any_t f, any_t param)
{

    mutex_lock(&scheduler_lock);
    switch ((int) f) {
    case SMAS_SCHED_ROUNDROBIN:
        schedfunc = round_robin;
        break;
    case SMAS_SCHED_FIRSTCOME:
        schedfunc = first_come;
        break;
#if 0
    case SMAS_SCHED_SHORTESTJOB:
        schedfunc = shortest_job;
        break;
#endif
    default:
        schedfunc = (sched_func) f;
        break;
    }
    schedparams = param;
    mutex_unlock(&scheduler_lock);
    return FALSE;
}

int retcounter()
{
```

```
    return sched_counter;
}
```

B.5 libthreads/cthreads.h

```
/*
 * Mach Operating System
 * Copyright (c) 1991,1990,1989 Carnegie Mellon University
 * All Rights Reserved.
 *
 * Permission to use, copy, modify and distribute this software and its
 * documentation is hereby granted, provided that both the copyright
 * notice and this permission notice appear in all copies of the
 * software, derivative works or modified versions, and any portions
 * thereof, and that both notices appear in supporting documentation.
 *
 * CARNEGIE MELLON ALLOWS FREE USE OF THIS SOFTWARE IN ITS "AS IS"
 * CONDITION.  CARNEGIE MELLON DISCLAIMS ANY LIABILITY OF ANY KIND FOR
 * ANY DAMAGES WHATSOEVER RESULTING FROM THE USE OF THIS SOFTWARE.
 *
 * Carnegie Mellon requests users of this software to return to
 *
 *  Software Distribution Coordinator  or  Software.Distribution@CS.CMU.EDU
 *  School of Computer Science
 *  Carnegie Mellon University
 *  Pittsburgh PA 15213-3890
 *
 * any improvements or extensions that they make and grant Carnegie Mellon
 * the rights to redistribute these changes.
 */
/*
 * HISTORY
 * $Log: cthreads.h,v $
 * Revision 1.15  1999/06/13 18:54:42  roland
 * 1999-06-13  Roland McGrath  <roland@baalperazim.frob.com>
 *
 *  * cthreads.h (MACRO_BEGIN, MACRO_END): #undef before unconditionally
 *  redefining.  Use GCC extension for statement expression with value 0.
 *
 * Revision 1.14  1999/05/30 01:39:48  roland
 * 1999-05-29  Roland McGrath  <roland@baalperazim.frob.com>
 *
 *  * cthreads.h (mutex_clear): Change again, to call mutex_init.
 *
 * Revision 1.13  1999/05/29 18:59:10  roland
 * 1999-05-29  Roland McGrath  <roland@baalperazim.frob.com>
 *
 *  * cthreads.h (mutex_clear): Change from syntax error to no-op (with
 *  warning avoidance).
 *
 * Revision 1.12  1996/05/04 10:06:31  roland
 * [lint] (NEVER): Spurious global variable removed.
 * [!lint] (NEVER): Useless macro removed.
 *
 * Revision 1.11  1996/01/24 18:37:59  roland
 * Use prototypes for functions of zero args.
 *
 * Revision 1.10  1995/09/13 19:50:07  mib
 * (CONDITION_INITIALIZER): Provide initial zero for IMPLICATIONS member.
 * (condition_init): Bother initializing NAME and IMPLICATIONS members.
 *
 * Revision 1.9  1995/08/30 15:51:41  mib
 * (condition_implies, condition_unimplies): New functions.
 * (struct condition): New member `implications'.
 * (cond_imp): New structure.
 * (cond_signal): Return int now.
 * (condition_broadcast): Always call cond_broadcast if this condition
 * has implications.
 * (condition_signal): Always call cond_signal if this condition has
 * implications.
```

```
*
* Revision 1.8  1995/08/30 15:10:23  mib
* (hurd_condition_wait): Provide declaration.
*
* Revision 1.7  1995/07/18 17:15:51  mib
* Reverse previous change.
*
* Revision 1.5  1995/04/04 21:06:16  roland
* (mutex_lock, mutex_unlock): Use __ names for *_solid.
*
* Revision 1.4  1994/05/05  10:52:06  roland
* entered into RCS
*
* Revision 2.12  92/05/22  18:38:36  jfriedl
*  From Mike Kupfer <kupfer@sprite.Berkeley.EDU>:
*  Add declaration for cthread_wire().
*  Merge in Jonathan Chew's changes for thread-local data.
*  Use MACRO_BEGIN and MACRO_END.
*
* Revision 1.8  91/03/25  14:14:49  jjc
*  For compatibility with cthread_data:
*  1) Added private_data field to cthread structure
*     for use by POSIX thread specific data routines.
*  2) Conditionalized old data field used by cthread_data
*     under CTHREAD_DATA for binary compatibility.
*  3) Changed macros, cthread_set_data and cthread_data,
*     into routines which use the POSIX routines for
*     source compatibility.
*     Also, conditionalized under CTHREAD_DATA.
*  [91/03/18            jjc]
*  Added support for multiplexing the thread specific global
*  variable, cthread_data, using the POSIX threads interface
*  for thread private data.
*  [91/03/14            jjc]
*
* Revision 2.11  91/08/03  18:20:15  jsb
*  Removed the infamous line 122.
*  [91/08/01  22:40:24  jsb]
*
* Revision 2.10  91/07/31  18:35:42  dbg
*  Fix the standard-C conditional: it's __STDC__.
*
*  Allow for macro-redefinition of cthread_sp, spin_try_lock,
*  spin_unlock (from machine/cthreads.h).
*  [91/07/30  17:34:28  dbg]
*
* Revision 2.9  91/05/14  17:56:42  mrt
*  Correcting copyright
*
* Revision 2.8  91/02/14  14:19:52  mrt
*  Added new Mach copyright
*  [91/02/13  12:41:15  mrt]
*
* Revision 2.7  90/11/05  14:37:12  rpd
*  Include machine/cthreads.h.  Added spin_lock_t.
*  [90/10/31            rwd]
*
* Revision 2.6  90/10/12  13:07:24  rpd
*  Channge to allow for positive stack growth.
*  [90/10/10            rwd]
*
* Revision 2.5  90/09/09  14:34:56  rpd
*  Remove mutex_special and debug_mutex.
*  [90/08/24            rwd]
*
* Revision 2.4  90/08/07  14:31:14  rpd
*  Removed RCS keyword nonsense.
*
* Revision 2.3  90/01/19  14:37:18  rwd
*  Add back pointer to cthread structure.
*  [90/01/03            rwd]
```

```
 *  Change definition of cthread_init and change ur_cthread_self macro
 *  to reflect movement of self pointer on stack.
 *  [89/12/18  19:18:34  rwd]
 *
 * Revision 2.2  89/12/08  19:53:49  rwd
 *  Change spin_try_lock to int.
 *  [89/11/30            rwd]
 *  Changed mutex macros to deal with special mutexs
 *  [89/11/26            rwd]
 *  Make mutex_{set,clear}_special routines instead of macros.
 *  [89/11/25            rwd]
 *  Added mutex_special to specify a need to context switch on this
 *  mutex.
 *  [89/11/21            rwd]
 *
 *  Made mutex_lock a macro trying to grab the spin_lock first.
 *  [89/11/13            rwd]
 *  Removed conditionals.  Mutexes are more like conditions now.
 *  Changed for limited kernel thread version.
 *  [89/10/23            rwd]
 *
 * Revision 2.1  89/08/03  17:09:40  rwd
 * Created.
 *
 *
 * 28-Oct-88  Eric Cooper (ecc) at Carnegie Mellon University
 * Implemented spin_lock() as test and test-and-set logic
 * (using mutex_try_lock()) in sync.c.  Changed ((char *) 0)
 * to 0, at Mike Jones's suggestion, and turned on ANSI-style
 * declarations in either C++ or _STDC_.
 *
 * 29-Sep-88  Eric Cooper (ecc) at Carnegie Mellon University
 * Changed NULL to ((char *) 0) to avoid dependency on <stdio.h>,
 * at Alessandro Forin's suggestion.
 *
 * 08-Sep-88  Alessandro Forin (af) at Carnegie Mellon University
 * Changed queue_t to cthread_queue_t and string_t to char *
 * to avoid conflicts.
 *
 * 01-Apr-88  Eric Cooper (ecc) at Carnegie Mellon University
 * Changed compound statement macros to use the
 * do { ... } while (0) trick, so that they work
 * in all statement contexts.
 *
 * 19-Feb-88  Eric Cooper (ecc) at Carnegie Mellon University
 * Made spin_unlock() and mutex_unlock() into procedure calls
 * rather than macros, so that even smart compilers can't reorder
 * the clearing of the lock.  Suggested by Jeff Eppinger.
 * Removed the now empty <machine>/cthreads.h.
 *
 * 01-Dec-87  Eric Cooper (ecc) at Carnegie Mellon University
 * Changed cthread_self() to mask the current SP to find
 * the self pointer stored at the base of the stack.
 *
 * 22-Jul-87  Eric Cooper (ecc) at Carnegie Mellon University
 * Fixed bugs in mutex_set_name and condition_set_name
 * due to bad choice of macro formal parameter name.
 *
 * 21-Jul-87  Eric Cooper (ecc) at Carnegie Mellon University
 * Moved #include <machine/cthreads.h> to avoid referring
 * to types before they are declared (required by C++).
 *
 *  9-Jul-87  Michael Jones (mbj) at Carnegie Mellon University
 * Added conditional type declarations for C++.
 * Added _cthread_init_routine and _cthread_exit_routine variables
 * for automatic initialization and finalization by crt0.
 */
/*
 *  File:  cthreads.h
 * Author: Eric Cooper, Carnegie Mellon University
 * Date: Jul, 1987
```

```
 *
 *  Definitions for the C Threads package.
 *
 */

#ifndef _CTHREADS_
#define _CTHREADS_ 1

/* MIB XXX */
#define CTHREAD_DATA

#if 0
/* This is CMU's machine-dependent file.  In GNU all of the machine
   dependencies are dealt with in libc.  */
#include <machine/cthreads.h>
#else
#include <machine-sp.h>
/*#define cthread_sp() ((int) __thread_stack_pointer ()) *****
 * Kinalus: This causes cthread_sel() to fail compilation*/
#endif

#if c_plusplus || __STDC__

#ifndef C_ARG_DECLS
#define C_ARG_DECLS(arglist) arglist
#endif                          /* not C_ARG_DECLS */

typedef void *any_t;

#else                           /* not (c_plusplus || __STDC__) */

#ifndef C_ARG_DECLS
#define C_ARG_DECLS(arglist) ()
#endif                          /* not C_ARG_DECLS */

typedef char *any_t;

#endif                          /* not (c_plusplus || __STDC__) */

#include <mach/mach.h>
#include <mach/machine/vm_param.h>

#ifndef TRUE
#define TRUE 1
#define FALSE 0
#endif                          /* TRUE */


#undef MACRO_BEGIN
#undef MACRO_END
#define MACRO_BEGIN __extension__ ({
#define MACRO_END 0; })


/*
 * C Threads package initialization.
 */

extern int cthread_init C_ARG_DECLS((void));
#if 0
/* This prototype is broken for GNU.  */
extern any_t calloc C_ARG_DECLS((unsigned n, unsigned size));
#else
#include <stdlib.h>
#endif

/*
 * Queues.
 */
typedef struct cthread_queue {
    struct cthread_queue_item *head;
    struct cthread_queue_item *tail;
} *cthread_queue_t;

typedef struct cthread_queue_item {
    struct cthread_queue_item *next;
} *cthread_queue_item_t;
```

```
#define NO_QUEUE_ITEM ((cthread_queue_item_t) 0)

#define QUEUE_INITIALIZER { NO_QUEUE_ITEM, NO_QUEUE_ITEM }

#define cthread_queue_alloc() ((cthread_queue_t) calloc(1, sizeof(struct cthread_queue)))
#define cthread_queue_init(q) ((q)->head = (q)->tail = 0)
#define cthread_queue_free(q) free((any_t) (q))

#define cthread_queue_enq(q, x) \
MACRO_BEGIN \
(x)->next = 0; \
if ((q)->tail == 0) \
(q)->head = (cthread_queue_item_t) (x); \
else \
(q)->tail->next = (cthread_queue_item_t) (x); \
(q)->tail = (cthread_queue_item_t) (x); \
MACRO_END

#define cthread_queue_preq(q, x) \
MACRO_BEGIN \
if ((q)->tail == 0) \
(q)->tail = (cthread_queue_item_t) (x); \
((cthread_queue_item_t) (x))->next = (q)->head; \
(q)->head = (cthread_queue_item_t) (x); \
MACRO_END

#define cthread_queue_head(q, t) ((t) ((q)->head))

#define cthread_queue_deq(q, t, x) \
MACRO_BEGIN \
if (((x) = (t) ((q)->head)) != 0 && \
    ((q)->head = (cthread_queue_item_t) ((x)->next)) == 0) \
(q)->tail = 0; \
MACRO_END

#define cthread_queue_map(q, t, f) \
MACRO_BEGIN \
register cthread_queue_item_t x, next; \
for (x = (cthread_queue_item_t) ((q)->head); x != 0; x = next) { \
next = x->next; \
(*(f))((t) x); \
} \
MACRO_END

#if 1

/* In GNU, spin locks are implemented in libc.
   Just include its header file.  */
#include <spin-lock.h>

#else                           /* Unused CMU code.  */

/*
 * Spin locks.
 */
extern void
spin_lock_solid C_ARG_DECLS((spin_lock_t * p));

#ifndef spin_unlock
extern void
spin_unlock C_ARG_DECLS((spin_lock_t * p));
#endif

#ifndef spin_try_lock
extern int
spin_try_lock C_ARG_DECLS((spin_lock_t * p));
#endif

#define spin_lock(p) ({if (!spin_try_lock(p)) spin_lock_solid(p);})

#endif                          /* End unused CMU code.  */

/*
 * Mutex objects.
```

```
 */
typedef struct mutex {
    /* The ‘held’ member must be first in GNU.  The GNU C library relies
       on being able to cast a ‘struct mutex *’ to a ‘spin_lock_t *’
       (which is kosher if it is the first member) and spin_try_lock that
       address to see if it gets the mutex.  */
    spin_lock_t held;
    spin_lock_t lock;
    char *name;
    struct cthread_queue queue;
} *mutex_t;

/* Rearranged accordingly for GNU: */
#define MUTEX_INITIALIZER { SPIN_LOCK_INITIALIZER, SPIN_LOCK_INITIALIZER, 0, QUEUE_INITIALIZER }

#define mutex_alloc() ((mutex_t) calloc(1, sizeof(struct mutex)))
#define mutex_init(m) \
MACRO_BEGIN \
spin_lock_init(&(m)->lock); \
cthread_queue_init(&(m)->queue); \
spin_lock_init(&(m)->held); \
MACRO_END
#define mutex_set_name(m, x) ((m)->name = (x))
#define mutex_name(m) ((m)->name != 0 ? (m)->name : "?")
#define mutex_clear(m) mutex_init(m)
#define mutex_free(m) free((any_t) (m))

extern void __mutex_lock_solid(void *mutex);    /* blocking -- roland@gnu */
extern void __mutex_unlock_solid(void *mutex);  /* roland@gnu */

#define mutex_try_lock(m) spin_try_lock(&(m)->held)
#define mutex_lock(m) \
MACRO_BEGIN \
if (!spin_try_lock(&(m)->held)) { \
__mutex_lock_solid(m); \
} \
MACRO_END
#define mutex_unlock(m) \
MACRO_BEGIN \
if (spin_unlock(&(m)->held), \
    cthread_queue_head(&(m)->queue, int) != 0) { \
__mutex_unlock_solid(m); \
} \
MACRO_END

/*
 * Condition variables.
 */
typedef struct condition {
    spin_lock_t lock;
    struct cthread_queue queue;
    char *name;
    struct cond_imp *implications;
} *condition_t;

struct cond_imp {
    struct condition *implicatand;
    struct cond_imp *next;
};

#define CONDITION_INITIALIZER { SPIN_LOCK_INITIALIZER, \
QUEUE_INITIALIZER, 0, 0 }

#define condition_alloc() ((condition_t) calloc(1, \
sizeof(struct condition)))
#define condition_init(c) \
MACRO_BEGIN \
spin_lock_init(&(c)->lock); \
cthread_queue_init(&(c)->queue); \
(c)->name = 0; \
```

```
(c)->implications = 0; \
MACRO_END
#define condition_set_name(c, x) ((c)->name = (x))
#define condition_name(c) ((c)->name != 0 ? (c)->name : "?")
#define condition_clear(c) \
MACRO_BEGIN \
condition_broadcast(c); \
spin_lock(&(c)->lock); \
MACRO_END
#define condition_free(c) \
MACRO_BEGIN \
condition_clear(c); \
free((any_t) (c)); \
MACRO_END

#define condition_signal(c) \
MACRO_BEGIN \
if ((c)->queue.head || (c)->implications) { \
cond_signal(c); \
} \
MACRO_END

#define condition_broadcast(c) \
MACRO_BEGIN \
if ((c)->queue.head || (c)->implications) { \
cond_broadcast(c); \
} \
MACRO_END

extern int
cond_signal C_ARG_DECLS((condition_t c));

extern void
cond_broadcast C_ARG_DECLS((condition_t c));

extern void
condition_wait C_ARG_DECLS((condition_t c, mutex_t m));

extern int
hurd_condition_wait C_ARG_DECLS((condition_t c, mutex_t m));

extern void
    condition_implies
C_ARG_DECLS((condition_t implicator, condition_t implicatand));

extern void
 condition_unimplies
C_ARG_DECLS((condition_t implicator, condition_t implicatand));

/*
 * Threads.
 */

typedef any_t(*cthread_fn_t) C_ARG_DECLS((any_t arg));

#include <setjmp.h>

typedef struct cthread {
    struct cthread *next;
    struct mutex lock;
    struct condition done;
    int state;
    jmp_buf catch;
    cthread_fn_t func;
    any_t arg;
    any_t result;
    char *name;
#ifdef CTHREAD_DATA
    any_t data;
#endif                          /* CTHREAD_DATA */
    any_t private_data;
    struct ur_cthread *ur;
#if 0
        /******Kinalis: data could be stored here for use by
 * scheduling algorithms currently unused**/
```

```
    any_t sched_data;
        /******************/
#endif
} *cthread_t;

#define NO_CTHREAD ((cthread_t) 0)

extern cthread_t
    cthread_fork C_ARG_DECLS((cthread_fn_t func, any_t arg));

extern cthread_t
    cthread_spawn C_ARG_DECLS((cthread_fn_t func, any_t arg));

extern void
cthread_detach C_ARG_DECLS((cthread_t t));

extern any_t cthread_join C_ARG_DECLS((cthread_t t));

extern void
cthread_yield C_ARG_DECLS((void));

extern void
cthread_exit C_ARG_DECLS((any_t result));

/*
 * This structure must agree with struct cproc in cthread_internals.h
 */
typedef struct ur_cthread {
    struct ur_cthread *next;
    cthread_t incarnation;
} *ur_cthread_t;

#ifndef cthread_sp
extern int
cthread_sp C_ARG_DECLS((void));
#endif

extern int cthread_stack_mask;

#ifdef STACK_GROWTH_UP
#define ur_cthread_ptr(sp) \
(* (ur_cthread_t *) ((sp) & cthread_stack_mask))
#else                          /* STACK_GROWTH_UP */
#define ur_cthread_ptr(sp) \
(* (ur_cthread_t *) ( ((sp) | cthread_stack_mask) + 1 \
      - sizeof(ur_cthread_t *)) )
#endif                         /* STACK_GROWTH_UP */

#define ur_cthread_self() (ur_cthread_ptr(cthread_sp()))

#define cthread_assoc(id, t) ((((ur_cthread_t) (id))->incarnation = (t)), \
((t) ? ((t)->ur = (ur_cthread_t)(id)) : 0))
#define cthread_self() (ur_cthread_self()->incarnation)

extern void
cthread_set_name C_ARG_DECLS((cthread_t t, char *name));

extern char *cthread_name C_ARG_DECLS((cthread_t t));

extern int
cthread_count C_ARG_DECLS((void));

extern void
cthread_set_limit C_ARG_DECLS((int n));

extern int
cthread_limit C_ARG_DECLS((void));

extern void
cthread_wire C_ARG_DECLS((void));

#ifdef CTHREAD_DATA
/*
 * Set or get thread specific "global" variable
 *
 * The thread given must be the calling thread (ie. thread_self).
 * XXX This is for compatibility with the old cthread_data. XXX
 */
extern int
```

```
cthread_set_data C_ARG_DECLS((cthread_t t, any_t x));

extern any_t cthread_data C_ARG_DECLS((cthread_t t));
#endif                          /* CTHREAD_DATA */

/*
 * Support for POSIX thread specific data
 *
 * Multiplexes a thread specific "global" variable
 * into many thread specific "global" variables.
 */
#define CTHREAD_DATA_VALUE_NULL (any_t)0
#define CTHREAD_KEY_INVALID (cthread_key_t)-1

typedef int cthread_key_t;

/*
 * Create key to private data visible to all threads in task.
 * Different threads may use same key, but the values bound to the key are
 * maintained on a thread specific basis.
 */
extern int
cthread_keycreate C_ARG_DECLS((cthread_key_t * key));

/*
 * Get value currently bound to key for calling thread
 */
extern int
cthread_getspecific
C_ARG_DECLS((cthread_key_t key, any_t * value));

/*
 * Bind value to given key for calling thread
 */
extern int
cthread_setspecific
C_ARG_DECLS((cthread_key_t key, any_t value));

/*
 * Debugging support.
 */
#ifdef DEBUG

#ifndef ASSERT
/*
 * Assertion macro, similar to <assert.h>
 */
#include <stdio.h>
#define ASSERT(p) \
MACRO_BEGIN \
if (!(p)) { \
fprintf(stderr, \
"File %s, line %d: assertion p failed.\n", \
__FILE__, __LINE__); \
abort(); \
} \
MACRO_END

#endif                          /* ASSERT */

#define SHOULDNT_HAPPEN 0

extern int cthread_debug;

#else                           /* DEBUG */

#ifndef ASSERT
#define ASSERT(p)
#endif                          /* ASSERT */

#endif                          /* DEBUG */


/***********Scheduler related code**************************/
```

```
extern int ready_queue_counter;
#define sched_queue_enq(q, x) \
MACRO_BEGIN \
(x)->sched_next = 0; \
if ((q)->tail == 0) \
(q)->head = (cthread_queue_item_t) (x); \
else \
((cproc_t)(q)->tail)->sched_next =  (x); \
(q)->tail = (cthread_queue_item_t) (x); \
ready_queue_counter++; \
MACRO_END

#define sched_queue_head(q) ((cproc_t)(q)->head)

#define sched_queue_deq(q, x) \
MACRO_BEGIN \
if (((x) = (cproc_t) ((q)->head)) != 0 && \
    ((q)->head = (cthread_queue_item_t) ((x)->sched_next)) == 0) \
(q)->tail = 0; \
if ((x) != 0) \
ready_queue_counter--; \
MACRO_END


#define SMAS_SCHED_ROUNDROBIN (any_t)1
#define SMAS_SCHED_FIRSTCOME (any_t)2
#define SMAS_SCHED_SHORTESTJOB (any_t)4

typedef any_t(*sched_func) (any_t);

extern struct cthread_queue ready_queue;

extern any_t sched(any_t);
extern int set_sched(any_t, any_t);
extern int retcounter();


#endif                          /* _CTHREADS_ */
```