Ατομική Διπλωματική Εργασία

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ ΑΤΟΜΙΚΩΝ ΑΝΤΙΚΕΙΜΕΝΩΝ ΓΡΑΦΗΣ/ΑΝΑΓΝΩΣΗΣ CWFR

Μαρία Στυλιανού

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάρτιος 2011

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ ΑΤΟΜΙΚΩΝ ΑΝΤΙΚΕΙΜΕΝΩΝ ΓΡΑΦΗΣ/ΑΝΑΓΝΩΣΗΣ CWFR

Μαρία Στυλιανού

Επιβλέπων Καθηγητής

Δρ. Χρύσης Γεωργίου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2011

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου Δρ. Χρύση Γεωργίου που μου έδωσε την ευκαιρία να εργαστώ πάνω στο αντικείμενο της παρούσας εργασίας. Με την καθοδήγηση και τη πολύτιμη βοήθεια του κατάφερα να ολοκληρώσω την εργασία αυτή.

Θα ήθελα επίσης να ευχαριστήσω τον Νικόλα Νικολάου για την άριστη συνεργασία μας και τη μεγάλη του βοήθεια που μου προσέφερε, μέσω πολλών συμβουλών αλλά και μέσω ενός προγράμματος που μου έδωσε του οποίου τα αποτελέσματα χρησιμοποιούνται στο δικό μου πρόγραμμα.

Περίληψη

Αυτή η διπλωματική εργασία κινείται στο χώρο των κατανεμημένων συστημάτων. Στην εποχή μας, τα κατανεμημένα συστήματα αποθήκευσης διαδίδονται ευρέως, λόγω της ανοχής σφαλμάτων, της επεκτασιμότητας και της υψηλής επίδοσης. Η ασυγχρονία και η ταυτοχρονία είναι κάποιες δυνατότητες που καταστούν αυτά τα συστήματα αναγκαία στις μέρες μας. Απαραίτητες προϋποθέσεις για την ύπαρξη και τη σωστή λειτουργία τέτοιων συστημάτων είναι η διατήρηση της ατομικότητας των δεδομένων καθώς και η συνέπεια μνήμης ανάμεσα στις διεργασίες, ακόμη και στην παρουσία σφαλμάτων.

Σε πρόσφατη έρευνα έχει παρουσιαστεί ο αλγόριθμος CWFR [1] ο οποίος επιτρέπει την ανταλλαγή μηνυμάτων εγγραφής/ανάγνωσης ατομικών αντικειμένων μεταξύ βιώσιμων ατομικών καταχωρητών. Σε αυτόν τον αλγόριθμο εισάγεται η δυνατότητα ολοκλήρωσης μιας ανάγνωσης σε ένα γύρο επικοινωνίας, σε αντίθεση με τον απλούστερο αλγόριθμο που ολοκληρώνει όλα τα είδη μηνυμάτων σε δύο γύρους επικοινωνίας.

Στόχος της παρούσας διπλωματικής εργασίας είναι να υλοποιηθεί ο απλούστερος αλγόριθμος και ο αλγόριθμος CWFR, να γίνει μια σύγκριση μεταξύ των δύο αλγορίθμων για να φανεί κατά πόσο ο CWFR είναι αποδοτικότερος σε ένα ρεαλιστικό ασύγχρονο δίκτυο. Οι δύο αλγόριθμοι υλοποιήθηκαν στη γλώσσα προγραμματισμού C και η πειραματική αξιολόγηση τους έχει γίνει στο ασύγχρονο και ρεαλιστικό δίκτυο PlanetLab.

Περιεχόμενα

Κεφάλαιο 1

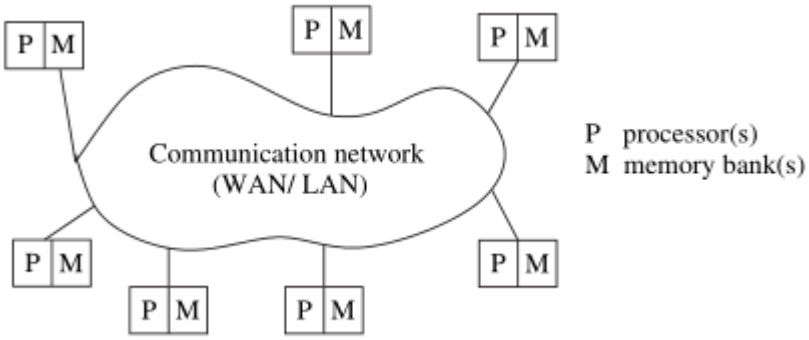
Εισαγωγή

1.1 Θεωρητικό Υπόβαθρο

Για τη καλύτερη κατανόηση της διπλωματικής εργασίας δίνονται κάποιοι ορισμοί και έννοιες που θα χρησιμοποιηθούν στη πορεία.

1.1.1 Κατανεμημένα Συστήματα

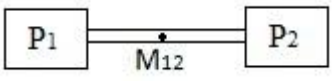
Ένα κατανεμημένο σύστημα [8] είναι μια συλλογή ανεξάρτητων και αυτόνομων οντοτήτων οι οποίες συνεργάζονται για την επίλυση ενός προβλήματος, το οποίο δεν μπορεί να λυθεί ατομικά. Συγκεκριμένα, ως οντότητες χαρακτηρίζονται οι επεξεργαστές οι οποίοι επικοινωνούν ανταλλάζοντας μηνύματα μέσω ενός δικτύου, αλλά παρουσιάζονται στους χρήστες ως μια μοναδική οντότητα. Δεν μοιράζονται κοινή μνήμη, έχουν ασυγχρονία, αυτονομία και υπάρχει γεωγραφική απόσταση μεταξύ τους. Στο Σχήμα 1.1 παρουσιάζεται ένα απλό κατανεμημένο σύστημα, στο οποίο ένα δίκτυο επικοινωνίας ενώνει όλους τους κόμβους, και ο κάθε ένας έχει τη δική του μνήμη και επεξεργαστή.



Σχήμα 1.1: Απλό Κατανεμημένο Δίκτυο αποτελούμενο από κόμβους ενωμένοι μέσω ενός δικτύου [8].

Τα κατανεμημένα συστήματα έχουν πολλά πλεονεκτήματα που λειτουργούν ως κίνητρα για να τα χρησιμοποιεί κανείς. Αρχικά, οι υπολογισμοί κατανέμονται μεταξύ των κόμβων με αποτέλεσμα η απόδοση να αυξάνεται. Επίσης, υπάρχουν αντίγραφα δεδομένων και πηγών σε όλο το δίκτυο και κατά συνέπεια παρατηρείται περισσότερη αξιοπιστία σε τέτοιου είδους συστήματα. Αυτό οφείλεται στη διαθεσιμότητα των δεδομένων και των πόρων από διαφορετικούς κόμβους, στην ακεραιότητα τους – κυρίως στις περιπτώσεις ταυτόχρονης πρόσβασης σε αυτά – και στη ανοχή σφαλμάτων, δηλαδή στην ικανότητα να επιβιώνουν ακόμη και στη παρουσία καταρρεύσεων κάποιων κόμβων. Τέλος, να σημειωθεί πως αυτά τα δίκτυα είναι επεκτάσιμα με το μεγάλο πλεονέκτημα πως όσο περισσότεροι κόμβοι υπάρχουν σε ένα κατανεμημένο σύστημα, τόσο πιο εύρωστο γίνεται.

Ένα κατανεμημένο πρόγραμμα αποτελείται από ένα σύνολο διεργασιών P_1, P_2, …, P_N, οι οποίες βρίσκονται σε ένα δίκτυο και επικοινωνούν μεταξύ τους με ανταλλαγή μηνυμάτων. Όπως φαίνεται στο Σχήμα 1.2, οι διεργασίες ανταλλάζουν μηνύματα μέσω κάποιων καναλιών. Το $M_{\chi\psi}$ είναι το μήνυμα που στέλνεται από τη διεργασία P_χ στη διεργασία P_ψ. Λόγω της ασυγχρονίας, υπάρχει η δυνατότητα να σταλούν μηνύματα πριν ακόμη παραδοθούν τα προηγούμενα. Επιπλέον, η καθυστέρηση στην επικοινωνία είναι μη προβλέψιμη αλλά περιορισμένη.

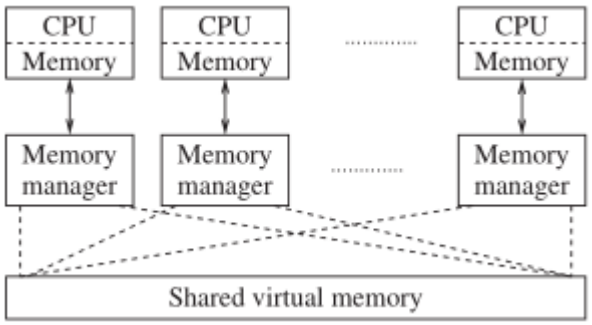


Σχήμα 1.2: Απλό Κατανεμημένο Πρόγραμμα, όπου η P_1 στέλνει μήνυμα M_{12} στη P_2.

Ένας κατανεμημένος υπολογισμός είναι η επίλυση υπολογιστικών προβλημάτων με τη χρήση κατανεμημένων συστημάτων. Συγκεκριμένα, τα προβλήματα αυτά μοιράζονται σε υποπροβλήματα, τα οποία επιλύονται από ένα ή περισσότερους επεξεργαστές του κατανεμημένου συστήματος.

1.1.2 Κατανεμημένη Κοινή Μνήμη

Η κατανεμημένη κοινή μνήμη (Distributed Shared Memory) [8] είναι μία αφηρημένη έννοια. Φαινομενικά, μοιάζει να είναι μια μοναδική μνήμη με ένα χώρο διευθύνσεων. Όμως, στη πραγματικότητα – όπως φαίνεται και στο Σχήμα 1.3 – κάθε κόμβος στο κατανεμημένο σύστημα προσφέρει ένα μέρος της μνήμης του για την κοινή μνήμη η οποία απεικονίζεται ως μια κοινή εικονική μνήμη ενός χώρου διευθύνσεων. Κάθε κόμβος έχει πρόσβαση στη δική του προσωπική μνήμη και στη κοινή μνήμη με τη χρήση των συναρτήσεων read και write.



Σχήμα 1.3: Απεικόνιση της Κατανεμημένης Κοινής Μνήμης [8].

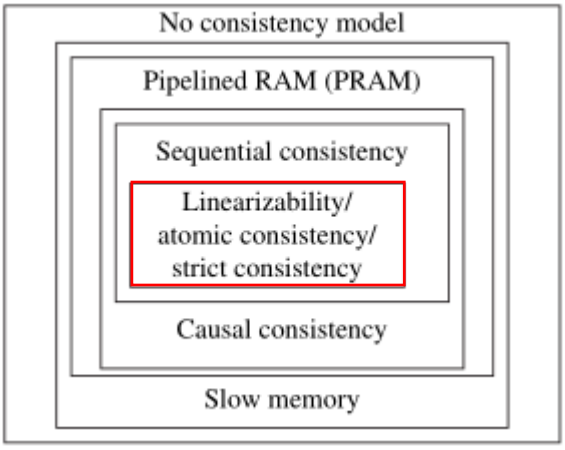
Όπως αναφέραμε προηγουμένως, η ταυτόχρονη πρόσβαση στη μνήμη είναι πιθανή σε ένα κατανεμημένο σύστημα. Ως εκ τούτου, κάποιος μηχανισμός ταυτοχρονίας είναι αναγκαίος για το σωστό χειρισμό των ταυτόχρονων αυτών προσβάσεων.

1.1.3 Συνέπεια Μνήμης

Συνέπεια μνήμης [8], ή αλλιώς συνέπεια μνήμης, είναι η ικανότητα του συστήματος να εκτελεί ορθά λειτουργίες μνήμης. Έστω N ο αριθμός των διεργασιών και si ο αριθμός των λειτουργιών μνήμης για τη διεργασία Pi. Προφανώς, ο αριθμός των διαφορετικών συνδυασμών εκτέλεσης αυτών των λειτουργιών είναι πολύ μεγάλος. Για να υπάρχει

συνέπεια μνήμης χρειάζεται να οριστεί η ορθότητα, ούτως ώστε να αφαιρεθούν οι συνδυασμοί που τη παραβιάζουν. Ένας κλασικός ορισμός της ορθότητας λέει πως μια ορθή εκτέλεση στη μνήμη είναι αυτή στην οποία κάθε λειτουργία ανάγνωσης επιστρέφει την τιμή που αποθηκεύτηκε στην πιο πρόσφατη λειτουργία εγγραφής. Με την παρουσία όμως ταυτόχρονων προσβάσεων στη μνήμη και πολλαπλών αντιγραφών του αντικειμένου, μπορεί να υπάρξουν περισσότερες από μια πρόσφατες λειτουργίες εγγραφής. Επομένως, χρειάζεται να προσαρμοστεί ο ορισμός της ορθότητας στο συγκεκριμένο σύστημα. Στόχος είναι να απορρίπτονται κάποιες ταυτόχρονες προσβάσεις, χωρίς να γίνεται το σύστημα υπερβολικά περιοριστικό ως προς τη συγχρονία.

Το μοντέλο συνέπειας μνήμης ορίζει το σύνολο των επιτρεπτών διατάξεων πρόσβασης στη μνήμη. Στο Σχήμα 1.4 μπορούμε να δούμε τα 6 διαφορετικά μοντέλα συνέπειας μνήμης. Στη παρούσα διπλωματική εργασία θα ασχοληθούμε μόνο με μοντέλο της ατομικής ή γραμμικής μνήμης (Linearizability/atomic consistency/strict constistency).



Σχήμα 1.4: Ιεραρχία των μοντέλων συνέπειας μνήμης [8].

1.1.4 Συστήματα Απαρτίας

Ένα σύστημα απαρτίας (Quorum System) [11,12] είναι ένα εργαλείο που χρησιμοποιείται για τη διασφάλιση της συνέπειας και της διαθεσιμότητας των αντίγραφων των δεδομένων, ακόμη και στη παρουσία σφαλμάτων. Ορίζεται ως μια συλλογή από σύνολα $Q = \{Q_1,\ldots,Q_m\}$, γνωστά ως quorums, από τα οποία το κάθε ένα

περιέχει ένα σύνολο εξυπηρετητών S = {S1,…,Si} για κάθε σύνολο Qi. Το σύστημα απαρτίας, $Q \subset 2^S$ είναι ένα υποσύνολο του S, ούτως ώστε κάθε δύο υποσύνολα του Q να τέμνονται μεταξύ τους. Ένα σύστημα απαρτίας καλείται n-wise, αν κάθε quorum που ανήκει σε αυτό το σύστημα τέμνεται το λιγότερο με n άλλα quorums. Κάθε quorum λειτουργεί και αντιπροσωπεύει το σύστημα. Επομένως, αν όλοι οι εξυπηρετητές ενός quorum λάβουν ένα μήνυμα και απαντήσουν, θεωρείται πως η λειτουργία έχει ολοκληρωθεί, γιατί σε κάθε τομή του quorum με τα υπόλοιπα υπάρχει το πιο πρόσφατο αντίγραφο.

Λέμε πως ένα σύστημα έχει ευρωστία όταν έχει μεγάλη ανοχή σφαλμάτων και όταν είναι αποδοτικό, δηλαδή γρήγορο. Τα συστήματα με κοινή τομή μεταξύ όλων των quorums δεν είναι εύρωστα αφού τη κατάρρευση ενός κόμβου από την κοινή τομή είναι αρκετή για να καταρρεύσει ολόκληρο το σύστημα.

Η ποιότητα ενός συστήματος απαρτίας μπορεί να μετρηθεί από 3 βασικά κριτήρια [13]. Αρχικό κριτήριο είναι το μέγεθος του συστήματος. Όσο μικρότερο είναι ένα σύστημα τόσο μικρότερη είναι και η πολυπλοκότητα του, εφόσον φυλάγονται λιγότερα αντίγραφα του αντικειμένου και συνεπώς η επικοινωνία μεταξύ των εξυπηρετητών είναι συντομότερη. Δεύτερο κριτήριο μπορεί να θεωρηθεί η διαθεσιμότητα ενός ολόκληρου quorum. Ας υποθέσουμε πως κάθε εξυπηρετητής μπορεί να καταρρεύσει με πιθανότητα p. Υπάρχει μια πιθανότητα Fp, να καταρρεύσουν αρκετοί εξυπηρετητές με αποτέλεσμα να μην υπάρχει ένα ολόκληρο λειτουργικό quorum στο σύστημα. Προφανώς, αυτή η πιθανότητα μετρά την ευρωστία του συστήματος, και για αυτό επιθυμητή πιθανότητα είναι η μικρότερη. Τέλος, σημαντικό κριτήριο είναι το φόρτος κάθε εξυπηρετητή, δηλαδή το πόσο συχνά χρησιμοποιείται. Σε όσο περισσότερα quorums ανήκει, τόσο περισσότερο χρησιμοποιείται και συνεπώς το φόρτος είναι μεγαλύτερο. Επιθυμητό φόρτος είναι το μικρότερο, αφού σε τέτοια περίπτωση οι εξυπηρετητές θα μπορούν να εκτελούν κι άλλες εργασίες.

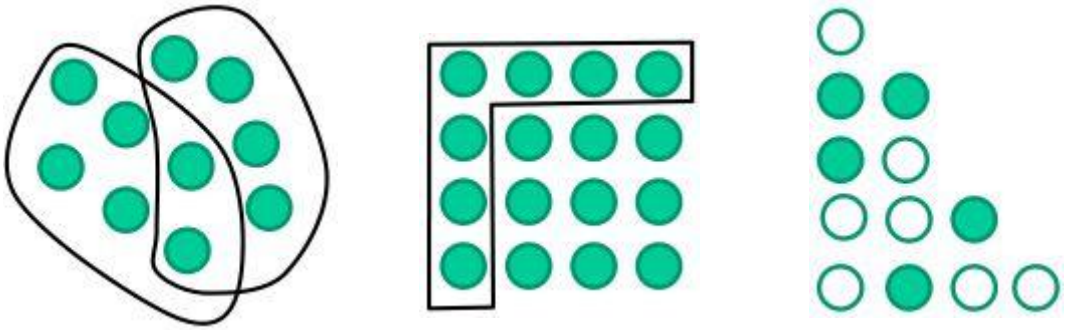
Παρατηρούμε πως τα κριτήρια είναι αντικρουόμενα. Στο πρώτο κριτήριο απαιτείται μικρός αριθμός εξυπηρετητών, ενώ στο δεύτερο κριτήριο απαιτείται μεγαλύτερος. Ως εκ τούτου, δεν υπάρχει κάποιο σύστημα απαρτίας που να είναι βέλτιστο ως προς όλα τα κριτήρια. Υπάρχουν, όμως, ποικίλα είδη συστημάτων απαρτίας, ανάλογα με τις ανάγκες

του χρήστη. Για τους σκοπούς αυτής της διπλωματικής εργασίας, θα ασχοληθούμε με τα εξής τρία είδη: (α) Το majority [13,16], (β) το matrix [13, 16] και (γ) το crumbling walls [13].

Στα majority συστήματα απαρτίας τα quorums αποτελούνται από τουλάχιστον $\lceil (n + 1) / 2 \rceil$ εξυπηρετητές. Όπως φαίνεται και στο Σχήμα 1.5 (α), υπάρχουν δύο σύνολα για τα οποία ισχύει ο πιο πάνω περιορισμός.

Στα matrix συστήματα απαρτίας (Σχήμα 1.5 (β)), έχουμε ένα ορθογώνιο πλέγμα αποτελούμενο από τους εξυπηρετητές. Κάθε quorum αποτελείται από μια γραμμή και μια στήλη του πλέγματος και έχει μέγεθος $2\sqrt{n-1}$ εξυπηρετητές.

Τα crumbling walls συστήματα απαρτίας (Σχήμα 1.5 (γ)) σχηματίζονται με παρόμοιο τρόπο με τη διαφορά πως οι γραμμές και οι στήλες έχουν διαφορετικό μέγεθος. Ένα quorum σχηματίζεται από μια ολόκληρη γραμμή και ένα στοιχείο από κάθε γραμμή που βρίσκεται κάτω από αυτή. Για παράδειγμα, το quorum στο Σχήμα 1.5 (γ) αποτελείται από τη δεύτερη γραμμή και τα σκουρόχρωμα στοιχεία που βρίσκονται κάτω από αυτή.



(α) (β) (γ)

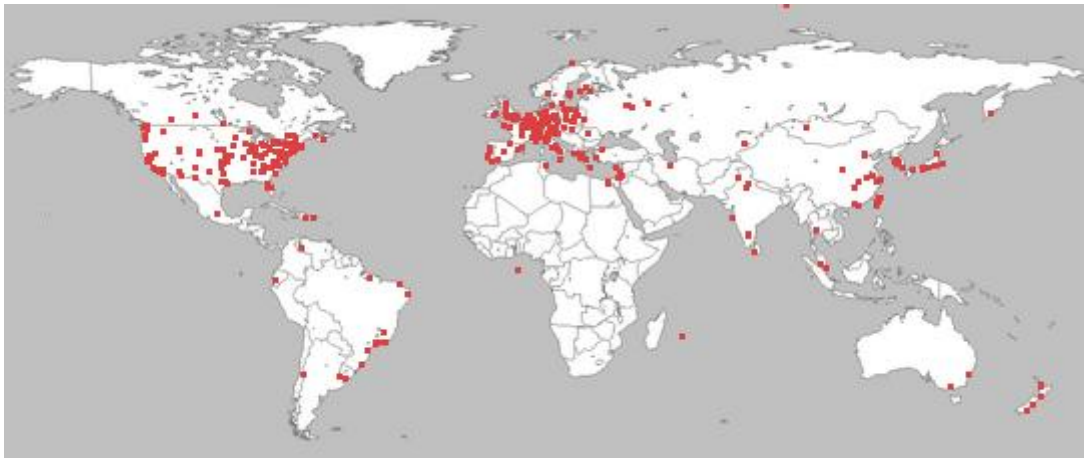
Σχήμα 1.5: Είδη Συστημάτων Απαρτίας [13,16]
(α) majority, (β) matrix, (γ) crumbling walls

1.1.5 Κατανεμημένο Δίκτυο PlanetLab

Το PlanetLab [17] είναι ένα κατανεμημένο δίκτυο το οποίο λειτουργεί ως μια πλατφόρμα ανάπτυξης και δοκιμής νέων δικτυακών εφαρμογών και υπηρεσιών

παγκοσμίου κλίμακας. Μέχρι στιγμής, στο PlanetLab συμμετέχουν ακαδημαϊκά, βιομηχανικά και κυβερνητικά ιδρύματα με 1064 κόμβους από 562 διαφορετικές τοποθεσίες παγκοσμίως (Σχήμα 1.6). Προϋπόθεση για ένα ίδρυμα να γίνει μέλος είναι να προσφέρει 2 ή περισσότερους υπολογιστές εις χρήση μέσα στο δίκτυο. Αυτοί οι υπολογιστές είναι που σχηματίζουν το overlay δίκτυο στο οποίο έχουν όλα τα μέλη πρόσβαση. Συγκεκριμένα οι πόροι κάθε κόμβου μοιράζονται σε μερίσματα (slices), το κάθε ένα από τα οποία παρουσιάζεται ως ένα δίκτυο με εικονικές μηχανές. Κάθε μέρισμα μπορεί να χρησιμοποιήσει μέχρι και 32 κόμβους. Από τη μέρα δημιουργίας του, το μέρισμα έχει διάρκεια ζωής ένα μήνα και αν δεν υπάρξει κάποια ανανέωση, τότε οι πόροι αυτού του μερίσματος αποδεσμεύονται και όλα τα δεδομένα χάνονται. Ωστόσο, επιτρέπονται απεριόριστες ανανεώσεις.

Η πρόσβαση στο δίκτυο είναι εφικτή μέσω SSH, το οποίο προσφέρει ασφαλή και κρυπτογραφημένη επικοινωνία. Επίσης, οι σταθμοί μπορούν να επανεγκατασταθούν ή και να επανεκκινήσουν μετατρέποντας το δίσκο σε μια μη-αξιόπιστη, προσωρινή μορφή αποθήκευσης. Το Πανεπιστήμιο Κύπρου συμμετέχει στο PlanetLab προσφέροντας δύο υπολογιστές. Για σκοπούς αυτής της διπλωματικής και για την πειραματική αξιολόγηση των αλγορίθμων έχει χρησιμοποιηθεί το PlanetLab ως ένα ρεαλιστικό δίκτυο.



Σχήμα 1.6: Το εύρος του δικτύου PlanetLab [17]

1.2 Προηγούμενες Εργασίες και Κίνητρα

Η αποδοτικότητα των λειτουργιών γραφής και ανάγνωσης μετριέται από τον αριθμό των γύρων επικοινωνίας μεταξύ των διεργασιών του συστήματος, οι οποίες κατατάσσονται στους αναγνώστες (readers), στους εγγραφείς (writers) και στους εξυπηρετητές (servers). Ένας γύρος επικοινωνίας (RTT) ξεκινά όταν ένας αναγνώστης ή εγγραφέας στείλει μια αίτηση για λειτουργία και ολοκληρώνεται όταν λάβει απαντήσεις από ένα σύνολο (όπως έχει οριστεί στο υποκεφάλαιο 1.1.4) των εξυπηρετητών.

Η πρώτη υλοποίηση ενός μοντέλου ατομικής μνήμης παρουσιάζεται στο [1], όπου δίνεται μια λύση με ένα εγγραφέα και πολλούς αναγνώστες (SWMR). Στο συγκεκριμένο μοντέλο n εξυπηρετητές λαμβάνουν αντίγραφο του αντικειμένου μέσω μηνυμάτων. Χρησιμοποιείται το σύστημα απαρτίας majority και ως εκ τούτου οι λειτουργίες ολοκληρώνονται με επιτυχία εφόσον ο αριθμός των εξυπηρετητών που κατέρρευσαν δεν φτάνει το $n/2$. Η λειτουργία εγγραφής χρειάζεται ένα (1) γύρο επικοινωνίας – και γι' αυτό ονομάζεται "fast" – ενώ η λειτουργία ανάγνωσης χρειάζεται δύο (2) γύρους επικοινωνίας – παίρνοντας το όνομα "slow" – με το δεύτερο γύρο να γράφει τη τιμή που πήρε στο πρώτο στάδιο. Ακολουθώντας αυτή την ανάπτυξη επικράτησε ο όρος πως «η ανάγνωση πρέπει να γράφει» σε περιβάλλοντα με πολλαπλούς αναγνώστες.

Ωστόσο, οι ερευνητές στο [2] μετατρέπουν την λειτουργία ανάγνωσης σε "fast" προσθέτοντας ένα περιορισμό ως προς τον αριθμό των αναγνωστών. Συγκεκριμένα λένε πως αν R ο αριθμός των αναγνωστών, S ο αριθμός των αντιγράφων και t ο αριθμός των καταρρεύσεων στο σύστημα, τότε πρέπει να ισχύει: $R < S/t - 2$. Οι υλοποιήσεις στις οποίες και οι δύο λειτουργίες είναι γρήγορες, όπως στο [2] ονομάζονται *γρήγορες*.

Στο [5], αφαιρείται ο περιορισμός του [2], επιτρέποντας απεριόριστο αριθμό αναγνωστών, με κόστος μια αργή λειτουργία ανάγνωσης για κάθε λειτουργία εγγραφής. Ο αλγόριθμος δοκιμάστηκε κάτω από ρεαλιστικές συνθήκες [4] αποδεικνύοντας πως οι περισσότερες λειτουργίες ανάγνωσης είναι γρήγορες. Εδώ εισάχθηκαν οι νοητοί κόμβοι (virtual nodes) μέσα στους οποίους ομαδοποιούνται οι αναγνώστες. Επισημάνθηκε πως η ημιγρήγορη συμπεριφορά διατηρείται όσο ο αριθμός των νοητών κόμβων δεν φτάνει

το S/t -2. Οι υλοποιήσεις στις οποίες συνυπάρχουν αργές και γρήγορες λειτουργίες, όπως στο [5], ονομάζονται *ημιγρήγορες* (semifast).

Χρησιμοποιώντας ως βάση το [1], οι ερευνητές παρουσίασαν στο [10] το μοντέλο με πολλούς εγγραφείς και πολλούς αναγνώστες (MWMR), στο οποίο το σύστημα απαρτίας είναι το majority και οι δύο λειτουργίες γραφής και ανάγνωσης είναι αργές. Μια παραλλαγή του αλγορίθμου αυτού υλοποιείται στα πλαίσια αυτής της διπλωματικής και έχει το όνομα SIMPLE [16].

Έχει αποδειχθεί στο [6] πως για την ύπαρξη γρήγορων και ημιγρήγορων υλοποιήσεων βασισμένων σε συστήματα απαρτίας, είναι απαραίτητη η ιδιότητα της τομής μεταξύ όλων των συνόλων (quorums). Μάλιστα, η ιδιότητα αυτή δεν χρειάζεται την ύπαρξη του περιορισμού στον αριθμό των αναγνωστών, όπως γίνεται στο [2]. Παρόλα αυτά, με την ικανοποίηση αυτής της ιδιότητας, το σύστημα δεν είναι εύρωστο. Αυτό οφείλεται στην ύπαρξη ενός μόνο σημείου σφάλματος (single point of failure), το οποίο μειώνει κατά πολύ την ανοχή σφαλμάτων.

Για την επίλυση αυτού του προβλήματος, οι ερευνητές στο [6] παρουσίασαν τον Semifast Like Implementation for Quorum systems (SLIQ) αλγόριθμο για την υλοποίηση ενός μοντέλου με ένα εγγραφέα και πολλούς αναγνώστες (SWMR). Σε αυτό το μοντέλο η ευρωστία και η ανοχή στα σφάλματα επιτυγχάνονται με αντάλλαγμα ένα μέρος της ταχύτητας της υλοποίησης. Η λειτουργία γραφής είναι πάντα γρήγορη ενώ η αποδοτικότητα της λειτουργίας ανάγνωσης βασίζεται πλέον στα Quorum Views – μια έννοια που θα μελετηθεί σε επόμενο κεφάλαιο – τα οποία προσφέρουν κάποια γνώση στους αναγνώστες μετά τον πρώτο γύρο επικοινωνίας και καθορίζουν αν θα χρειαστεί δεύτερος γύρος ή όχι. Οι υλοποιήσεις που βασίζονται σε συστήματα απαρτίας και χρησιμοποιούν τα Quorum Views, ονομάζονται *ασθενές-ημιγρήγορες* (weak-semifast).

Με βάση το [6], έγινε περαιτέρω έρευνα στο [3] παρουσιάζοντας δύο καινούργιους αλγορίθμους, τους CWFR και SFW. Στον CWFR η λειτουργία γραφής χρειάζεται 2 γύρους επικοινωνίας, ενώ η λειτουργία ανάγνωσης χρειάζεται 1 ή 2 γύρους, ανάλογα με τα quorum views. Ο SFW αξιοποιεί μια τεχνική που λέγεται Server Side Ordering

(SSO) η οποία επιτρέπει γρήγορες λειτουργίες γραφής και ανάγνωσης σε συγκεκριμένες περιπτώσεις.

1.3 Σκοπός Διπλωματικής Εργασίας

Σκοπός της παρούσας διπλωματικής εργασίας είναι η υλοποίηση και η πειραματική αξιολόγηση των δύο αλγορίθμων ατομικών αντικειμένων γραφής/ανάγνωσης SIMPLE και CWFR [3] με πολλαπλούς εγγραφείς και πολλαπλούς αναγνώστες. Απώτερος στόχος είναι να συγκριθούν τα αποτελέσματα των δύο αυτών αλγορίθμων ως προς την απόδοση τους σ' ένα ασύγχρονο δίκτυο με αυθαίρετα σφάλματα και να εκτιμηθεί κατά πόσο ο αλγόριθμος CWFR είναι πιο αποδοτικός από τον SIMPLE.

1.4 Μεθοδολογία Υλοποίησης

Η μεθοδολογία που ακολουθήθηκε για την υλοποίηση της διπλωματικής εργασίας αυτής είναι η ακόλουθη: Αρχικά, έγινε μελέτη άρθρων και προηγούμενων εργασιών που αφορούσαν αλγορίθμους γραφής και ανάγνωσης ατομικών αντικειμένων, δίνοντας περισσότερη έμφαση στο υλικό που αφορούσε καθαρά τους αλγορίθμους SIMPLE και CWFR. Αναγκαία ήταν η απόκτηση θεωρητικού υποβάθρου για τον κατανεμημένο υπολογισμό και την κατανεμημένη μνήμη.

Έπειτα, έγινε εξοικείωση στο προγραμματισμό με υποδοχές ροής που εξυπηρετεί στην επικοινωνία των διεργασιών στο κατανεμημένο σύστημα, όπως προϋποθέτουν οι αλγόριθμοι. Η αρχική υλοποίηση απλών υπολογιστικών παραδειγμάτων βοήθησε στη καλύτερη κατανόηση του προγραμματισμού με υποδοχές και υπήρξε η βάση για τη μετέπειτα υλοποίηση των αλγορίθμων.

Ακολούθησε η υλοποίηση των αλγορίθμων SIMPLE και CWFR όπως περιγράφονται στα [10] και [3] αντίστοιχα, στη γλώσσα προγραμματισμού C. Η αποσφαλμάτωση έγινε αρχικά στο τοπικό δίκτυο του Πανεπιστημίου Κύπρου και αργότερα έγινε η πειραματική αξιολόγηση τους στο κατανεμημένο δίκτυο PlanetLab.

Προτού αρχίσει η πειραματική αξιολόγηση, προηγήθηκε η εκμάθηση του τρόπου λειτουργίας του PlanetLab, καθώς επίσης η υλοποίηση κάποιων αρχείων εντολών (scripts) – με βάση τις γνώσεις που αποκόμισα από το [19] – για την προετοιμασία των σεναρίων και την αυτοματοποίηση των εντολών που επρόκειτο να εκτελεστούν. Αφού δοκιμάστηκε ο αλγόριθμος κάτω από διαφορετικά σενάρια, υλοποιήθηκε ακόμη ένα αρχείο εντολών για την αποκόμιση των αποτελεσμάτων. Στο τέλος, έγινε μια σύγκριση των δύο αλγορίθμων και η εξαγωγή συμπερασμάτων.

1.5 Δομή Διπλωματικής Εργασίας

Το έγγραφο αυτό περιλαμβάνει έξι κεφάλαια. Στο Κεφάλαιο 1 δίνονται κάποιες βασικές έννοιες, τα κίνητρα και η μεθοδολογία της εργασίας. Το Κεφάλαιο 2 που ακολουθεί περιέχει την περιγραφή των δύο αλγορίθμων, ενώ το Κεφάλαιο 3 εξηγεί την υλοποίηση τους. Έπειτα, στο Κεφάλαιο 4 δίνεται μια επεξήγηση των ρυθμίσεων στο PlanetLab που προηγήθηκαν της πειραματικής αξιολόγησης και στο Κεφάλαιο 5 περιγράφεται η πειραματική αξιολόγηση των αλγορίθμων και η εξαγωγή συμπερασμάτων. Στο τελευταίο κεφάλαιο, δηλαδή το Κεφάλαιο 6, αναφέρονται τα συμπεράσματα τα οποία εξάγονται από την δουλειά που έγινε για την ανάπτυξη και ολοκλήρωση της εργασίας αυτής.

Κεφάλαιο 2

Περιγραφή Αλγορίθμων

2.1 Κοινά Χαρακτηριστικά Αλγορίθμων

Προτού δοθούν οι περιγραφές των δύο αλγορίθμων, κρίνεται αναγκαίος ο ορισμός και η επεξήγηση κάποιων κοινών εννοιών και λειτουργιών των αλγορίθμων

2.1.1 Ατομικότητα

Ένα ατομικό αντικείμενο είναι μια αφηρημένη έννοια που παρουσιάζει μια δομή δεδομένων η οποία ορίζεται από ένα σύνολο πιθανών τιμών και υποστηρίζεται από λειτουργίες ανάγνωσης (read) και γραφής (write). Η ταυτόχρονη πρόσβαση σε τέτοιου είδους αντικείμενα είναι εφικτή προσφέροντας την αίσθηση πως οι λειτουργίες

αναλαμβάνονται σε μια διαδοχική σειρά. Γι' αυτό και κάποιες φορές ονομάζονται γραμμικά αντικείμενα [9].

Όπως έχει ειπωθεί μια λειτουργία ανάγνωσης ή γραφής ξεκινά τη στιγμή που η διεργασία πελάτη στέλνει αίτηση στους εξυπηρετητές του συστήματος και ολοκληρώνεται όταν λάβει απαντήσεις από ένα σύνολο εξυπηρετητών, δηλαδή από ένα quorum. Το βήμα αποστολής της αίτησης λέγεται βήμα επίκλησης (invocation step) και το βήμα λήψης των απαντήσεων λέγεται βήμα απόκρισης ή επιστροφής (response step). Η εκτέλεση και των δύο βημάτων είναι αναγκαία για την ολοκλήρωση μιας λειτουργίας. Είναι σημαντικό να ξεκαθαριστεί πως μια διεργασία δεν μπορεί να αιτήσει μια νέα λειτουργία εάν δεν έχει ολοκληρωθεί η προηγούμενη λειτουργία που έχει αιτήσει.

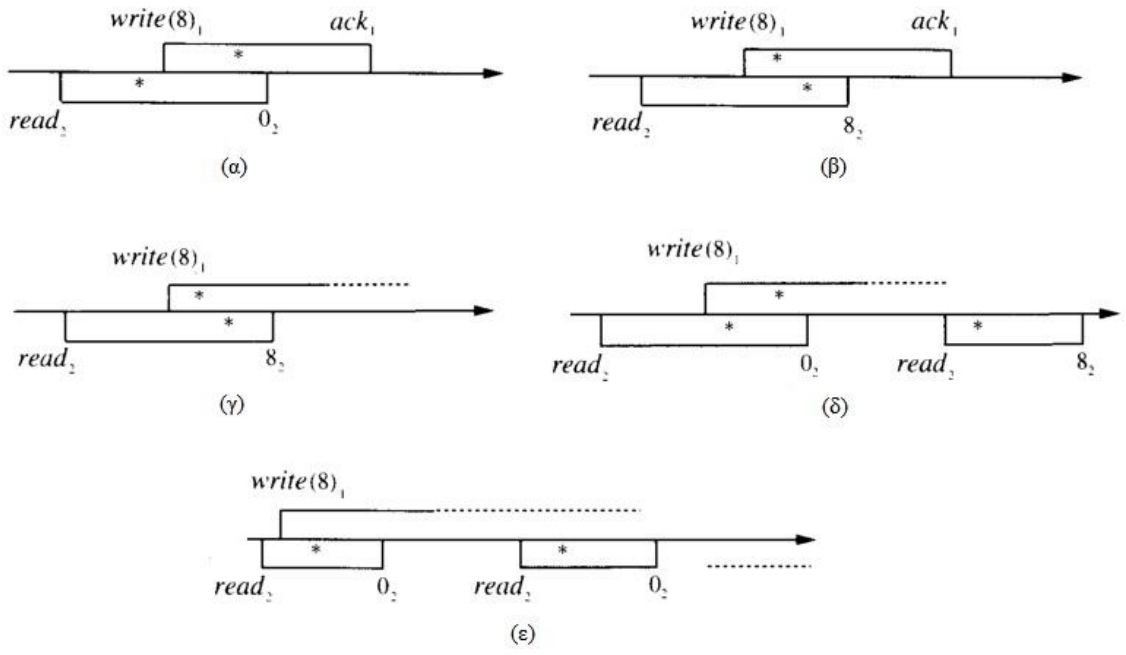
Μια λειτουργία λ1 προηγείται μιας λειτουργίας λ2, αν το βήμα απόκρισης της λ1 ολοκληρωθεί πριν το βήμα επίκλησης της λ2 και συμβολίζεται ως λ1→λ2. Οι λειτουργίες λ1 και λ2 είναι ταυτόχρονες (concurrent) αν η λ1 δεν προηγείται της λ2, δηλαδή λ1 ↛ λ2, και η λ2 δεν προηγείται της λ1, δηλαδή λ2 ↛ λ1.

Ωστόσο, κάθε λειτουργία συμβαίνει σε κάποια συγκεκριμένη στιγμή μεταξύ των βημάτων επίκλησης και επιστροφής. Επομένως, η ατομικότητα ορίζεται ως εξής:

1. Μια λειτουργία ανάγνωσης r ενός αντικειμένου x επιστρέφει είτε τη τιμή της τελευταίας ολοκληρωμένης γραφής w πάνω στο x είτε τη τιμή μιας γραφής w που είναι ταυτόχρονη με την r.
2. Εάν μια λειτουργία ανάγνωσης r1 επιστρέφει τη τιμή που έγραψε μια λειτουργία γραφής w1 και μια λειτουργία ανάγνωσης r2 επιστρέφει τη τιμή που έγραψε μια λειτουργία γραφής w2 και η r1 προηγείται της r2, τότε η w2 δεν προηγείται της w1.
3. Όλες οι λειτουργίες γραφής είναι αυστηρά διατεταγμένες.

Στο Σχήμα 2.1 φαίνονται κάποια παραδείγματα ατομικότητας με ταυτόχρονες λειτουργίες γραφής και ανάγνωσης, όπου το ατομικό αντικείμενο έχει αρχική τιμή μηδέν (0). Με αστερίσκο (*) παρουσιάζεται η συγκεκριμένη στιγμή που γίνεται η γραφή ή η ανάγνωση. Στα παραδείγματα (α) και (β) του σχήματος φαίνονται δύο λειτουργίες γραφής και ανάγνωσης οι οποίες έχουν ολοκληρωθεί. Στο (α) η στιγμή της

ανάγνωσης γίνεται πριν τη στιγμή της γραφής για αυτό επιστρέφεται η αρχική τιμή του αντικείμενου, το μηδέν (0). Στο (β) η ανάγνωση γίνεται μετά τη γραφή με αποτέλεσμα να πάρει τη νέα τιμή της γραφής, το οκτώ (8). Στα παραδείγματα (γ), (δ) και (ε) η λειτουργία της γραφής δεν έχει ολοκληρωθεί. Παρ' όλα αυτά, στο (γ) η στιγμή που γράφεται η νέα τιμή στο αντικείμενο συμβαίνει πριν την ανάγνωση, με αποτέλεσμα η ανάγνωση που γίνεται αργότερα να παίρνει τη νέα τιμή, το οκτώ. Στο (δ) φαίνεται ακόμη πιο ξεκάθαρα η έννοια της ατομικότητας. Συμβαίνουν δύο αναγνώσεις, όπου η μία προηγείται της άλλης. Η λειτουργία της γραφής είναι ταυτόχρονη με τις λειτουργίες ανάγνωσης, αλλά η στιγμή της γραφής συμβαίνει ενδιάμεσα των δύο αναγνώσεων. Ως εκ τούτου, η πρώτη λειτουργία ανάγνωσης επιστρέφει την αρχική τιμή του αντικειμένου, το μηδέν (0) – γιατί ακόμη να γραφεί η νέα, ενώ η δεύτερη λειτουργία ανάγνωσης επιστρέφει τη πιο πρόσφατη, το οκτώ (8). Στο τελευταίο (ε) παράδειγμα του Σχήματος 2.1, η γραφή δεν έχει συμβεί, με αποτέλεσμα οι δύο λειτουργίες ανάγνωσης να επιστρέφουν τη ίδια τιμή, το μηδέν.

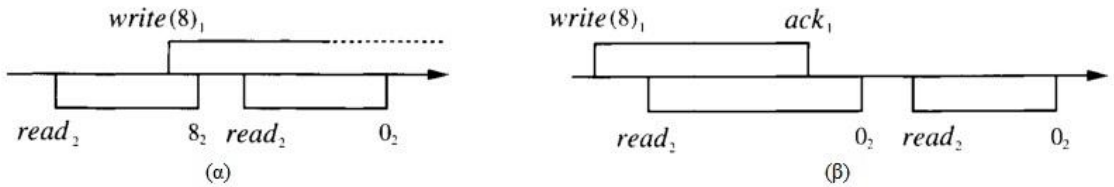


Σχήμα 2.1: Παραδείγματα Ατομικότητας [9].

Αντίθετα στο Σχήμα 2.2 βλέπουμε κάποια παραδείγματα που παραβιάζουν την ατομικότητα. Στο (α) παρουσιάζεται μια λειτουργία γραφής η οποία δεν έχει ολοκληρωθεί και δύο ολοκληρωμένες λειτουργίες ανάγνωσης, όπου η μια προηγείται

της άλλης. Η παραβίαση έγκειται στο ότι η πρώτη λειτουργία ανάγνωσης επιστρέφει τη νέα τιμή που πήρε το αντικείμενο από τη λειτουργία γραφής, ενώ η δεύτερη λειτουργία ανάγνωσης επιστρέφει την αρχική τιμή. Αυτό είναι αδύνατο λόγω της φύσεως του ατομικού αντικειμένου, η οποία εγγυάται πως όταν κάποια διεργασία διαβάσει τη τιμή του αντικειμένου, τότε αν δεν αλλάξει από κάποια λειτουργία γραφής, οποιαδήποτε μελλοντική λειτουργία ανάγνωσης θα επιστρέψει σίγουρα την ίδια τιμή.

Στο (β) παράδειγμα του Σχήματος 2.2, υπάρχουν τρεις λειτουργίες ολοκληρωμένες, εκ των οποίων δύο είναι αναγνώσεις και μία γραφή. Η γραφή είναι ταυτόχρονη με τη πρώτη λειτουργία ανάγνωσης. Παρατηρούμε πως η πρώτη λειτουργία ανάγνωσης επιστρέφει την αρχική τιμή του αντικειμένου, υποδηλώνοντας πως η γραφή δεν έχει ακόμη συμβεί. Η λειτουργία γραφής, όμως, ολοκληρώνεται πριν ξεκινήσει η δεύτερη λειτουργία ανάγνωσης. Παρόλα αυτά, η δεύτερη λειτουργία ανάγνωσης επιστρέφει επίσης την αρχική τιμή, παραβιάζοντας αυτόματα την ατομικότητα.



Σχήμα 2.2: Παραδείγματα Παραβίασης Ατομικότητας [9].

2.1.2 Συστήματα Απαρτίας

Οι δύο αλγόριθμοι που υλοποιούνται σε αυτή τη διπλωματική εργασία βασίζονται στα συστήματα απαρτίας, όπου οι εξυπηρετητές κρατούν αντίγραφα του αντικειμένου στη τοπική τους μνήμη. Οι διεργασίες του εγγραφέα και του αναγνώστη είναι ενημερωμένες για την ύπαρξη και δομή του συστήματος απαρτίας, τα οποία είναι στατικά και παραμένουν τα ίδια καθ' όλη τη διάρκεια των αλγορίθμων. Όπως έχουμε αναφέρει, οι δύο αλγόριθμοι θα δοκιμαστούν σε τρία διαφορετικά συστήματα απαρτίας: το matrix, το majority και το crumbling walls.

Η κατάρρευση κάποιας διεργασίας είναι δυνατή και πιθανή σε οποιαδήποτε στιγμή κατά τη διάρκεια των αλγορίθμων. Αν καταρρεύσει ονομάζεται επιρρεπής σε σφάλματα

(faulty). Από τη στιγμή που κάποια διεργασία είναι επιρρεπής σε σφάλματα, αυτόματα και τα quorums στα οποία ανήκει καλούνται επιρρεπή σε σφάλματα. Ωστόσο, προϋπόθεση και των δύο αλγορίθμων – ομοίως με την προϋπόθεση της ιδιότητας της τομής – είναι να υπάρχει τουλάχιστον ένα quorum που δεν είναι επιρρεπές σε σφάλματα, για να είναι εφικτή η λειτουργία τους.

2.1.3 Λειτουργία

Οι δύο αλγόριθμοι SIMPLE και CWFR έχουν πολλά κοινά χαρακτηριστικά στη λειτουργία τους. Καταρχήν, αποτελούνται από τρία σύνολα διεργασιών: τους εξυπηρετητές $S=\{S_1,\ldots,S_s\}$, τους γραφείς $W=\{W_1,\ldots,W_w\}$ και τους αναγνώστες $R=\{R_1,\ldots,R_r\}$, όπου s, w και r είναι τα πλήθη των συνόλων αντίστοιχα. Κάθε διεργασία έχει μοναδικό αναγνωριστικό και η επικοινωνία μεταξύ τους είναι περιορισμένη. Συγκεκριμένα, οι εξυπηρετητές δεν μπορούν να επικοινωνήσουν μεταξύ τους, παρά μόνο με τους πελάτες με ανταλλαγή μηνυμάτων. Παρομοίως, οι πελάτες μπορούν να επικοινωνήσουν μόνο με τους εξυπηρετητές. Το δίκτυο στο οποίο επικοινωνούν είναι ασύγχρονο και κάθε διεργασία εκτελεί κάποια λειτουργία με διαφορετική ταχύτητα. Ακόμη, κάθε διεργασία μπορεί να καταρρεύσει οποιαδήποτε στιγμή χωρίς προηγούμενη προειδοποίηση. Επομένως, δεν μπορεί να ελεγχθεί αν κάποια διεργασία έχει καταρρεύσει ή λειτουργεί πολύ αργά.

Τα μηνύματα που αποστέλλονται μεταξύ των διεργασιών αποτελούνται από το τύπο του μηνύματος, ένα ζεύγος τιμών < tag, value > και τον αριθμό αίτησης του εγγραφέα/αναγνώστη. Ο τύπος του μηνύματος μπορεί να είναι είτε READ και WRITE αν προέρχεται από κάποιο εγγραφέα ή αναγνώστη είτε READACK και WRITEACK αν προέρχεται από κάποιο εξυπηρετητή.

Το ζεύγος < tag, value > χρησιμοποιείται από τους αλγορίθμους για να διατάξουν τις ταυτόχρονες γραφές. Το tag αποτελείται από μια εγγραφή με δύο πεδία < ts, wid >, όπου ts είναι η χρονοσφραγίδα του αντικειμένου και wid η ταυτότητα του εγγραφέα, ενώ το value είναι η τιμή που θέλει να γράψει ο εγγραφέας. Αρχικά το tag είναι αρχικοποιημένο ως < 0, 0 > σε όλες τις διεργασίες και η χρονοσφραγίδα του αυξάνεται κατά ένα (1) σε κάθε λειτουργία WRITE που προέρχεται από διεργασία του εγγραφέα.

Η σύγκριση μεταξύ των tags γίνεται αλφαριθμητικά. Ας υποθέσουμε δύο μηνύματα m1 και m2 με tags τα tag1 και tag2 αντίστοιχα. Το m1 είναι πιο πρόσφατο από το m2, όταν το tag1 είναι μεγαλύτερο από το tag2. Αυτό ισχύει (tag1 > tag2), εάν είτε η χρονοσφραγίδα του tag1 είναι μεγαλύτερη από αυτή του tag2 (tag1.ts > tag2.ts) είτε οι δύο χρονοσφραγίδες ισούνται – που σημαίνει πως οι δύο γραφές είναι ταυτόχρονες – και η ταυτότητα του tag1 είναι μεγαλύτερη από αυτή του tag2 (tag1.ts == tag2.ts && tag1.wid > tag2.wid).

Ο αριθμός αίτησης είναι ένας μετρητής που αριθμεί τις αιτήσεις που έκανε ο κάθε εγγραφέας/αναγνώστης. Προτού κάποια διεργασία πελάτη στείλει κάποιο μήνυμα, αυξάνει το μετρητή αιτήσεων κατά ένα (1). Αργότερα, αφότου λάβει ένα μήνυμα, ελέγχει αν ο αριθμός αίτησης του μηνύματος είναι ίσος με το δικό του μετρητή. Αυτός ο έλεγχος είναι αναγκαίος γιατί είναι πιθανή η λήψη παλαιότερων μηνυμάτων, τα οποία ο εγγραφέας αγνοεί.

Συμπληρωματικά, να υπενθυμίσουμε πως μια διεργασία P εκτελεί ένα γύρο επικοινωνίας για κάποια λειτουργία λ εάν:

1. Η P στείλει ένα μήνυμα m για τη σχετική λειτουργία σε ένα υποσύνολο διεργασιών (που έχουν το ρόλο των εξυπηρετητών).
2. Κάθε διεργασία – εξυπηρετητής που λαμβάνει το m απαντά με acknowledgement στην P.
3. Η P συλλέγει απαντήσεις από ένα quorum.

Στη συνέχεια θα δούμε τη λειτουργία του εγγραφέα και του εξυπηρετητή, η οποία είναι ίδια και για τους δύο αλγορίθμους.

2.1.4 Εγγραφέας

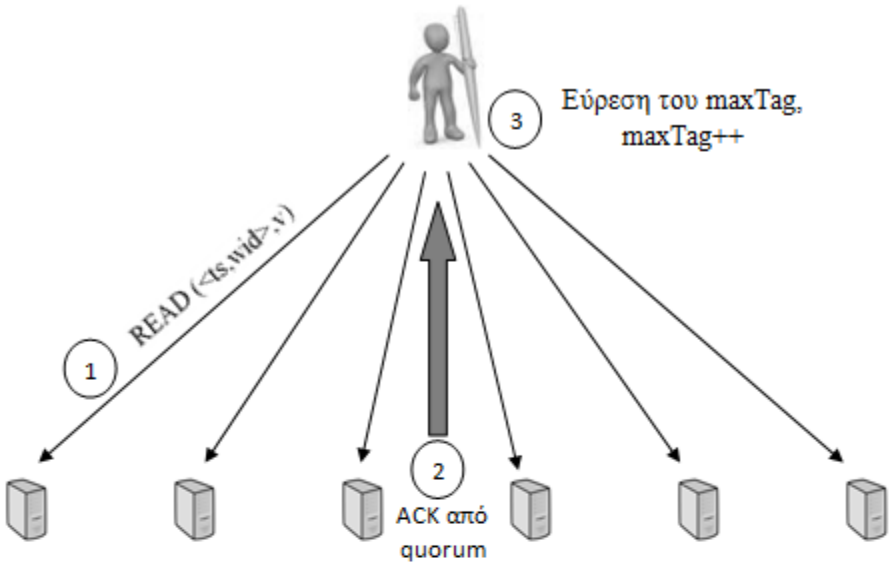
Στο Σχήμα 2.3 παρουσιάζεται το πρωτόκολλο του εγγραφέα. Ο εγγραφέας (writer) είναι η μόνη διεργασία που μπορεί να δώσει μια νέα τιμή στο αντικείμενο και χρειάζεται δύο γύρους επικοινωνίας για να ολοκληρώσει μια λειτουργία. Όταν θέλει να εκτελέσει μια λειτουργία γραφής, αυξάνει τον αριθμό αιτήσεων του κατά ένα και στέλνει στους εξυπηρετητές του συστήματος ένα μήνυμα READ για να μάθει το tag του αντικειμένου.

Μόλις λάβει έγκυρες απαντήσεις READACK από ένα ολόκληρο quorum, τότε ολοκληρώνεται ο πρώτος γύρος επικοινωνίας. Με τον όρο έγκυρες εννοούμε τις απαντήσεις που περνούν τον έλεγχο σχετικά με τον αριθμό αίτησης. Έπειτα, ο εγγραφέας εντοπίζει τη πιο πρόσφατη τιμή του αντικειμένου ανάμεσα στις απαντήσεις και αποθηκεύει τοπικά τη χρονοσφραγίδα από το tag του μηνύματος. Προτού αρχίσει ο δεύτερος γύρος επικοινωνίας, αυξάνει τον αριθμό αιτήσεων του και τη χρονοσφραγίδα που μόλις φύλαξε κατά ένα και αποθηκεύει στο μήνυμα που θα στείλει τη νέα τιμή του αντικειμένου. Τότε στέλνει στους εξυπηρετητές ένα μήνυμα WRITE με τα πιο πάνω στοιχεία και μόλις λάβει έγκυρες απαντήσεις WRITEACK από ένα quorum, ολοκληρώνεται ο δεύτερος γύρος επικοινωνίας και ως αποτέλεσμα ολόκληρη η λειτουργία.

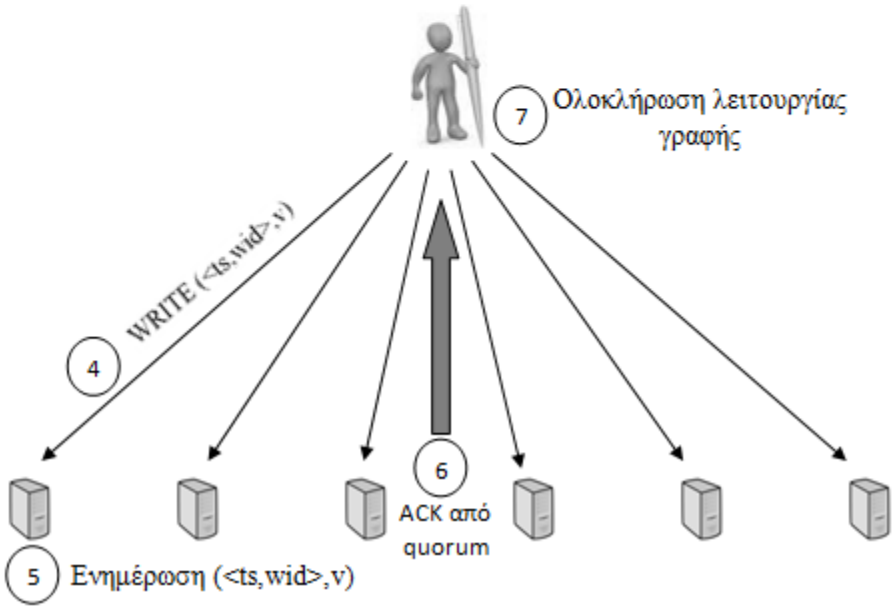
2.1.5 Εξυπηρετητής

Ο εξυπηρετητής (server) έχει ένα παθητικό ρόλο στο σύστημα. Τοπικά φυλάει ένα αντίγραφο του αντικειμένου το οποίο ανανεώνεται βάση των μηνυμάτων που παίρνει. Λαμβάνει μηνύματα WRITE και READ με κάποια στοιχεία για το αντικείμενο και απαντά με μηνύματα WRITEACK και READACK αντίστοιχα. Ο εξυπηρετητής φυλάει, επίσης, στη τοπική του μνήμη τον αριθμό αιτήσεων για τη κάθε διεργασία πελάτη. Έτσι, κάθε φορά που λαμβάνει κάποιο μήνυμα, ελέγχει αν ο αριθμός αιτήσεως του μηνύματος για το συγκεκριμένο πελάτη, είναι μεγαλύτερος από αυτόν που έχει τοπικά. Αν όχι, τότε το αγνοεί γιατί σημαίνει πως είναι κάποιο παλιό καθυστερημένο μήνυμα. Με τον τρόπο αυτό, μειώνεται τυχόν αχρείαστη συμφόρηση του δικτύου. Στη περίπτωση έγκυρου μηνύματος, αν το μήνυμα που έλαβε είναι READ, επιστρέφει τα στοιχεία του αντικειμένου που έχει στη τοπική του μνήμη. Αν το μήνυμα που έλαβε είναι WRITE, ανανεώνει τα στοιχεία του αντίγραφου του αντικειμένου που κρατά τοπικά.

1ος γύρος επικοινωνίας: Αποστολή READ και εύρεση του maxTag <ts,wid>



2ος γύρος επικοινωνίας: Αποστολή WRITE και διάδοση του maxTag <ts,wid>



Σχήμα 2.3: Πρωτόκολλο Εγγραφέα

2.2 Αλγόριθμος SIMPLE

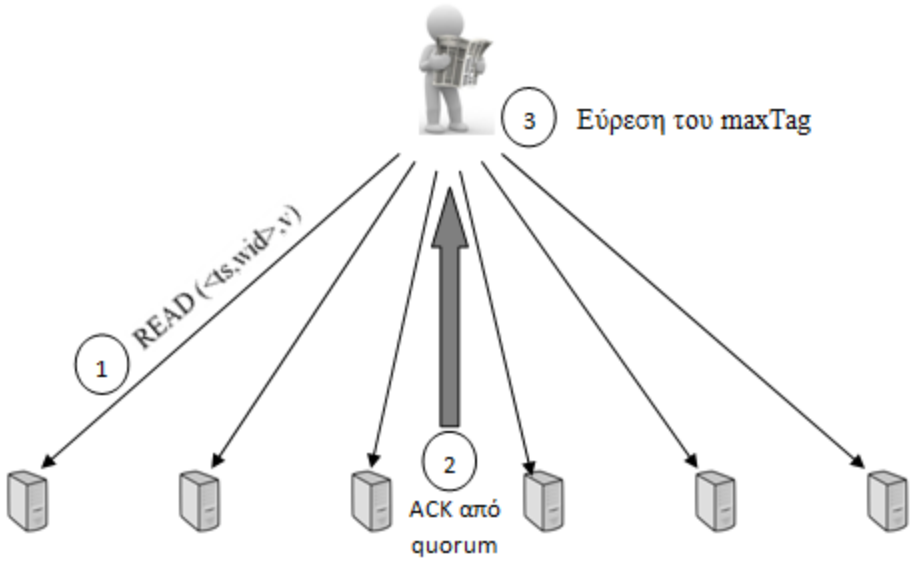
2.2.1 Περιγραφή Αλγορίθμου

Το πρώτο μοντέλο που υλοποιείται είναι ο απλός αλγόριθμος ατομικών αντικειμένων γραφής και ανάγνωσης με πολλαπλούς εγγραφείς και αναγνώστες. Στα πλαίσια αυτής της διπλωματικής εργασίας, τον ονομάζουμε SIMPLE γιατί είναι το πιο απλό MWMR μοντέλο που μπορεί να υπάρξει. Οι λειτουργίες γραφής και ανάγνωσης είναι αργές, δηλαδή χρειάζονται δύο (2) γύρους επικοινωνίας για να ολοκληρωθούν.

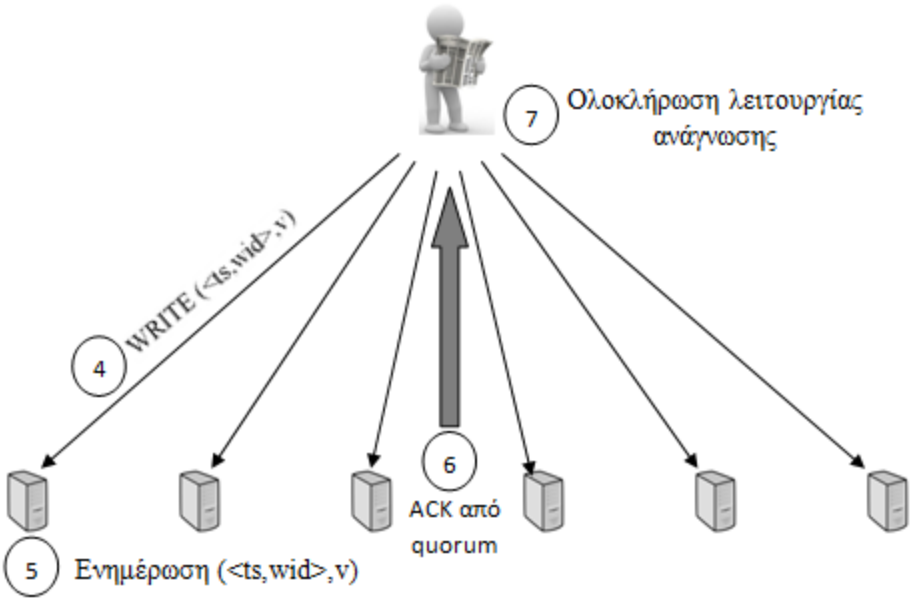
2.2.2 Αναγνώστης

Στο Σχήμα 2.4 παρουσιάζεται το πρωτόκολλο του αναγνώστη, το οποίο μοιάζει πολύ με αυτό του εγγραφέα. Ο αναγνώστης (reader) εκτελεί τις λειτουργίες ανάγνωσης κατά παρόμοιο τρόπο με τον εγγραφέα. Όταν θέλει να εκτελέσει μια λειτουργία ανάγνωσης, αυξάνει τον αριθμό αιτήσεων του κατά ένα και στέλνει στους εξυπηρετητές του συστήματος ένα μήνυμα READ για να μάθει τη τιμή του αντικειμένου. Μόλις λάβει έγκυρες απαντήσεις READACK από ένα ολόκληρο quorum, τότε ολοκληρώνεται ο πρώτος γύρος επικοινωνίας. Έπειτα, ο αναγνώστης εντοπίζει τη πιο πρόσφατη τιμή του αντικειμένου ανάμεσα στις απαντήσεις και αποθηκεύει τοπικά το tag του μηνύματος. Τότε χρειάζεται ένας δεύτερος γύρος επικοινωνίας, ούτως ώστε να ενημερώσει τους εξυπηρετητές που κρατούσαν παλαιότερη τιμή, με τη νέα τιμή του αντικειμένου. Έτσι, ο αναγνώστης αυξάνει τον αριθμό αιτήσεων του κατά ένα και στέλνει στους εξυπηρετητές ένα μήνυμα WRITE με τα στοιχεία του πιο πρόσφατου αντικειμένου. Μόλις λάβει έγκυρες απαντήσεις WRITEACK από ένα quorum, ολοκληρώνεται και ο δεύτερος γύρος επικοινωνίας και ως αποτέλεσμα ολόκληρη η λειτουργία. Με αυτό τον τρόπο διαδίδεται η νέα τιμή σε όλο το σύστημα απαρτίας.

1ος γύρος επικοινωνίας: Αποστολή READ και εύρεση του maxTag <ts,wid>

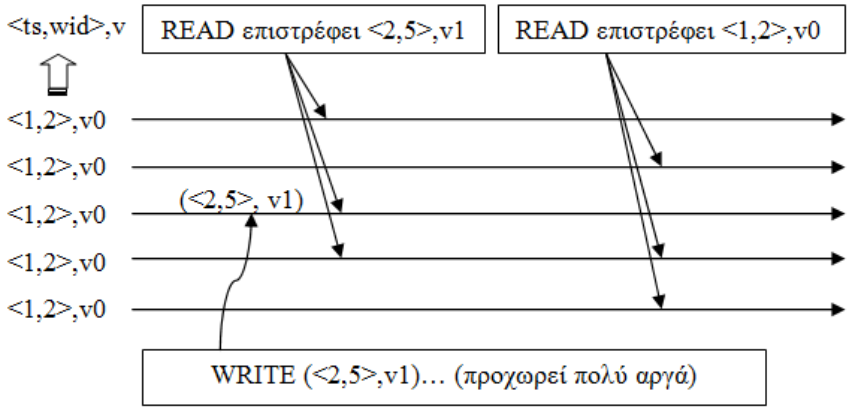


2ος γύρος επικοινωνίας: Αποστολή WRITE και διάδοση του maxTag <ts,wid>



Σχήμα 2.4: Πρωτόκολλο Αναγνώστη

Στο Σχήμα 2.5 δίνεται ένα παράδειγμα όπου η ατομικότητα παραβιάζεται αν η λειτουργία ανάγνωσης ολοκληρώνεται στο $1^{ο}$ γύρο επικοινωνίας. Ας υποθέσουμε ότι έχουμε σύστημα απαρτίας majority και μια λειτουργία ολοκληρώνεται όταν απαντήσει η πλειοψηφία των εξυπηρετητών. Μια αργή λειτουργία γραφής γράφει το νέο tag <2,v1>. Όσο είναι σε εξέλιξη, μια λειτουργία ανάγνωσης διαβάζει από ένα σύνολο εξυπηρετητών, όπου υπάρχει το νέο tag. Εφόσον δεν γίνεται $2^{ος}$ γύρος επικοινωνίας για την ενημέρωση των υπολοίπων εξυπηρετητών με τη νέα τιμή, ολοκληρώνεται σε αυτό το σημείο κρατώντας το <2,v1> ως τελικό tag. Τότε, ένας δεύτερος αναγνώστης διαβάζει από ένα σύνολο εξυπηρετητών, όπου δεν υπάρχει το νέο tag, γιατί η λειτουργία γραφής βρίσκεται ακόμη σε εξέλιξη. Έτσι, επιστρέφει το tag <1,v0>, παραβιάζοντας την ατομικότητα, όπως έχει προαναφερθεί στο υποκεφάλαιο 2.1.1.



Σχήμα 2.5: Παραβίαση ατομικότητας στην ολοκλήρωση της λειτουργίας ανάγνωσης σε ένα γύρο επικοινωνίας.

2.3 Αλγόριθμος CWFR

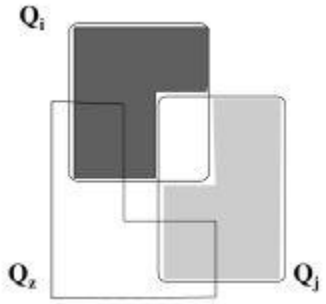
Στον αλγόριθμο CWFR εισάγεται η έννοια των Quorum Views, τα οποία δίνουν επαρκείς πληροφορίες στον αναγνώστη για τη κατανομή του tag του αντικειμένου στο quorum που έχει απαντήσει κατά το πρώτο γύρο επικοινωνίας. Αργότερα, θα συσχετίσουμε την έννοια αυτή με τη λειτουργία του αλγορίθμου και πως μετατρέπει μια slow MWMR υλοποίηση σε weak semifast.

2.3.1 Όψεις Συστημάτων Απαρτίας

Οι όψεις συστημάτων απαρτίας (quorum views) [6] είναι μία τεχνική που χρησιμοποιείται από τον αναγνώστη για να εξακριβώσει αν έγινε κάποια λειτουργία γραφής ή αν γίνεται ταυτόχρονα με τη δική του λειτουργία. Συγκεκριμένα, μέσα από τις όψεις συστημάτων απαρτίας ο αναγνώστης βλέπει την κατανομή του μέγιστου tag μέσα στο quorum που του έχει απαντήσει κατά το πρώτο γύρο επικοινωνίας. Σύμφωνα με τις πληροφορίες που παίρνει αποφασίζει αν χρειάζεται να συνεχίσει σε δεύτερο γύρο επικοινωνίας ή όχι.

Ας υποθέσουμε πως για κάποια λειτουργία γραφής ή ανάγνωσης της διεργασίας p, κάθε εξυπηρετητής s από κάποιο quorum Q, το οποίο ανήκει στο σύστημα απαρτίας, απαντά στη p με s.tag. Έστω ότι maxTag = $\max_{s \in Q}$(s.tag). Λέμε πως η p παρατηρεί κάποιο από τις εξής όψεις συστημάτων απαρτίας για το Q:

1. qView(1): $\forall s \in Q : s.tag = maxTag$,
2. qView(2): $\forall Q' \in \mathbb{Q} : Q \neq Q' \wedge \exists A \subseteq Q \cap Q'$, τ.ω. $A \neq \emptyset$ και $\forall s \in A :$ $s.tag < maxTag$,
3. qView(3): $\exists s' \in Q : s'.tag < maxTag$ και $\exists\ Q' \in \mathbb{Q}$ τ.ω. $Q \neq Q' \wedge \forall s \in Q \cap Q' :$ $s.tag = maxTag$



Σχήμα 2.6: Τομές μεταξύ τριών συνόλων εξυπηρετητών

Ο πιο πάνω ορισμός εξηγείται ευκολότερα με κάποια παραδείγματα. Έστω ότι έχουμε τρία quorums Qi, Qj και Qz, τα οποία ανήκουν σε ένα σύστημα απαρτίας $\mathbb{Q}$ και τέμνονται μεταξύ τους όπως φαίνεται στο Σχήμα 2.6. Ας υποθέσουμε, επίσης, πως κάποιος αναγνώστης, κατά τη λειτουργία ανάγνωσης, λαμβάνει απαντήσεις από το quorum Qj και ανακαλύπτει το maxTag ως το μέγιστο tag που συναντά μέσα στο

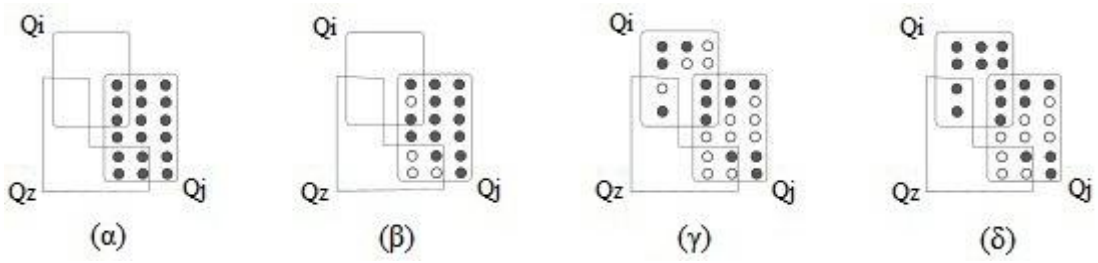
quorum. Στα παραδείγματα του Σχήματος 2.7, βλέπουμε τα διαφορετικά quorum views που μπορεί να συναντήσει. Οι σκουρόχρωμοι εξυπηρετητές κρατούν το μέγιστο tag, ενώ οι άλλοι παλαιότερα tags. Αν όλοι οι εξυπηρετητές του Qj έχουν το maxTag τότε ο αναγνώστης παρατηρεί το qView(1), όπως φαίνεται στο Σχήμα 2.7 (α). Από αυτό συμπεραίνουμε πως είτε καμία λειτουργία γραφής δεν έχει προηγηθεί είτε έχει επικοινωνήσει με το Qj και έχει ολοκληρωθεί με επιτυχία. Συμπερασματικά, ο αναγνώστης ολοκληρώνει τη λειτουργία του στο πρώτο γύρο επικοινωνίας αφού όλοι οι εξυπηρετητές του quorum έχουν το μέγιστο tag και κατά συνέπεια, σε όλα τα υπόλοιπα quorums υπάρχει το μέγιστο tag λόγω των τομών.

Αν κάποιοι εξυπηρετητές του Qj περιέχουν παλαιότερο tag, αλλά υπάρχει έστω μία τομή του με άλλο quorum, της οποίας όλοι οι εξυπηρετητές έχουν το maxTag, τότε παρατηρείται το qView(3). Εδώ υπάρχουν δύο περιπτώσεις. Η πρώτη, όπως παρουσιάζεται στο Σχήμα 2.7 (γ), φανερώνει πως ταυτόχρονα με τη λειτουργία ανάγνωσης, συμβαίνει κάποια λειτουργία γραφής η οποία δεν έχει ολοκληρωθεί. Αντίθετα, στη δεύτερη περίπτωση, και στο Σχήμα 2.7 (δ), κάποια προηγούμενη λειτουργία γραφής έχει ολοκληρωθεί λαμβάνοντας απαντήσεις από το Qi, με αποτέλεσμα η τομή μεταξύ των Qi και Qj να περιέχει το πιο πρόσφατο tag. Μια διεργασία ανάγνωσης αν παίρνει απαντήσεις μόνο από το Qj δεν μπορεί να διαχωρίσει τις δύο περιπτώσεις. Γι' αυτό χρειάζεται ο δεύτερος γύρος επικοινωνίας για να διαδοθεί το maxTag στους υπόλοιπους εξυπηρετητές του Qj.

Τέλος, το qView(2) εμφανίζεται όταν κανένα από τα άλλα δύο quorum views δεν ισχύει. Συγκεκριμένα, όταν το Qj περιέχει παλαιότερα tags και σε κάθε τομή του με άλλα quorums υπάρχουν παλαιότερα tags επίσης, τότε έχουμε το Σχήμα 2.7 (β). Σε αυτή τη περίπτωση ο αναγνώστης χρειάζεται να κάνει τους εξής υπολογισμούς:

1. Έλεγχος των tags που εμφανίζονται στο Qj και αφαίρεση των εξυπηρετητών που κρατούν το maxTag.
2. Έλεγχος αν το Qj με τους εναπομείναντες εξυπηρετητές ανήκει σε κάποιο quorum view. Εάν όχι, τότε επιστροφή στο βήμα 1.

Όπως παρατηρείται, γίνεται μια εναλλαγή των βημάτων μέχρι ο αναγνώστης καταλήξει σε μία από τις δύο περιπτώσεις. Με αυτό τον τρόπο πολλές λειτουργίες ολοκληρώνονται γρηγορότερα και, το σημαντικότερο, διατηρείται η ατομικότητα.



Σχήμα 2.7: Περιπτώσεις των quorum views [6]
(α) qView(1), (β) qView(2), (γ) qView(3) με μη-ολοκληρωμένη γραφή,
(δ) qView(3) με ολοκληρωμένη γραφή

2.3.2 Περιγραφή Αλγορίθμου

Το δεύτερο μοντέλο, CWFR, που υλοποιείται είναι ο αλγόριθμος ατομικών αντικειμένων γραφής και ανάγνωσης με πολλαπλούς εγγραφείς και αναγνώστες, ο οποίος βασίζεται στα quorum views. Χάρη στη τεχνική αυτή που εξηγήθηκε πιο πάνω, ο αλγόριθμος γίνεται weak-semifast, με τις λειτουργίες γραφής να είναι αργές και τις λειτουργίες ανάγνωσης να χρειάζονται είτε 1 είτε 2 γύρους επικοινωνίας, αναλόγως των quorum views. Οι διεργασίες του εξυπηρετητή και του εγγραφέα παραμένουν οι ίδιες, ενώ στη διεργασία του αναγνώστη εισάγεται η νέα τεχνική.

2.3.3 Αναγνώστης

Όπως και στον αλγόριθμο SIMPLE, στο πρώτο γύρο επικοινωνίας ο αναγνώστης αυξάνει τον αριθμό αιτήσεων του κατά ένα και στέλνει στους εξυπηρετητές του συστήματος ένα μήνυμα READ για να μάθει τη τιμή του αντικειμένου. Μόλις λάβει έγκυρες απαντήσεις READACK από ένα ολόκληρο quorum, αντί να συνεχίσει με το δεύτερο γύρο επικοινωνίας, ακολουθούν κάποιοι τοπικοί υπολογισμοί. Συγκεκριμένα, ο αναγνώστης χρησιμοποιεί τα quorum views για να εξετάσει την κατανομή του μέγιστου tag μέσα στο quorum που του απάντησε. Αν βρει quorum view (1), τερματίζει τη λειτουργία του στο πρώτο γύρο επικοινωνίας και φυλάει στη τοπική του μνήμη το maxTag που είχε διαβάσει. Αν βρει quorum view (3), συνεχίζει με δεύτερο γύρο επικοινωνίας για να διαδώσει το maxTag σε όλους τους εξυπηρετητές του συστήματος.

Σε αυτό το γύρο, ο αναγνώστης αυξάνει τον αριθμό αιτήσεων του κατά ένα και στέλνει στους εξυπηρετητές ένα μήνυμα WRITE με τα στοιχεία του πιο πρόσφατου αντικειμένου. Μόλις λάβει έγκυρες απαντήσεις WRITEACK από ένα quorum, ολοκληρώνεται και ο δεύτερος γύρος επικοινωνίας και ως αποτέλεσμα ολόκληρη η λειτουργία. Τέλος, αν βρει quorum view (2), τότε λειτουργώντας αναδρομικά, αφαιρεί τους εξυπηρετητές με το πιο πρόσφατο tag μέχρις ότου να συναντήσει ο αναγνώστης quorum view (1) ή (3).

Κεφάλαιο 3

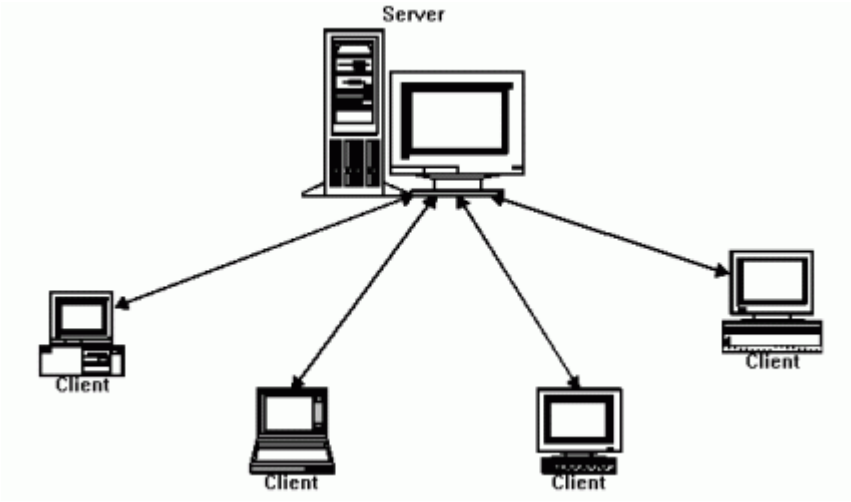
Υλοποίηση Αλγορίθμων

Οι δύο αλγόριθμοι υλοποιήθηκαν στη γλώσσα προγραμματισμού C, ακολουθώντας το μοντέλο πελάτη-εξυπηρετητή (client – server). Σε αυτό το κεφάλαιο εξηγείται αυτό το μοντέλο, καθώς επίσης οι βιβλιοθήκες και τα εργαλεία που χρησιμοποιήθηκαν για την ολοκλήρωση της υλοποίησης των αλγορίθμων.

3.1 Μοντέλο Πελάτη – Εξυπηρετητή

Το μοντέλο πελάτη – εξυπηρετητή (Σχήμα 3.1) είναι ένα σύστημα με αρχιτεκτονική δικτύου, στο οποίο υπάρχουν δύο είδη διεργασιών, η διεργασία του πελάτη (client) και

η διεργασία του εξυπηρετητή (server). Η κάθε διεργασία είναι σχεδιασμένη για να κάνει κάποια συγκεκριμένη δουλειά. Οι εξυπηρετητές παρέχουν πρόσβαση σε διάφορες υπηρεσίες και δεδομένα, όπως βάσεις δεδομένων, μονάδες εισόδου – εξόδου, επικοινωνίες κ.α., ενώ οι πελάτες αποκτούν πρόσβαση σε αυτά μέσω των εξυπηρετητών. Παραδείγματα εξυπηρετητών είναι οι web servers, οι ftp servers, οι mail servers. Κάποια είδη πελατών είναι οι web browsers και οι email clients.



Σχήμα 3.1: Μοντέλο Πελάτη – Εξυπηρετητή [12]

Η λειτουργία στο μοντέλο είναι πολύ απλή. Ένας πελάτης που επιθυμεί να επικοινωνήσει με κάποιο εξυπηρετητή, χρειάζεται αρχικά να εγκαθιδρύσει μια σύνδεση μαζί του για να είναι εφικτή η επικοινωνία τους. Ακολούθως, κάνει αίτηση σε αυτόν, και αφού ο εξυπηρετητής αποδεχτεί την αίτηση του, την επεξεργάζεται και απαντά πίσω με τις αιτούμενες πληροφορίες.

Ανάλογα με το είδος του εξυπηρετητή, οι αιτήσεις μπορεί να υπόκεινται σε επεξεργασία σειριακά ή και ταυτόχρονα. Αναλυτικότερα, υπάρχουν δύο κατηγορίες εξυπηρετητών [20], αυτή των σειριακών (iterative) και αυτή των ταυτόχρονων (concurrent). Ένας σειριακός εξυπηρετητής επεξεργάζεται μια – μια τις αιτήσεις, τη μια μετά την άλλη. Αντίθετα, ο ταυτόχρονος εξυπηρετητής για κάθε αίτηση που λαμβάνει, την αναθέτει σε κάποιο άλλο εξυπηρετητή για να τη χειριστεί. Οι άλλοι εξυπηρετητές είναι νήματα (threads) ή νέες διεργασίες στο σύστημα οι οποίες τερματίζουν με την ολοκλήρωση της αίτησης. Με αυτό τον τρόπο ο αρχικός εξυπηρετητής είναι σε θέση να

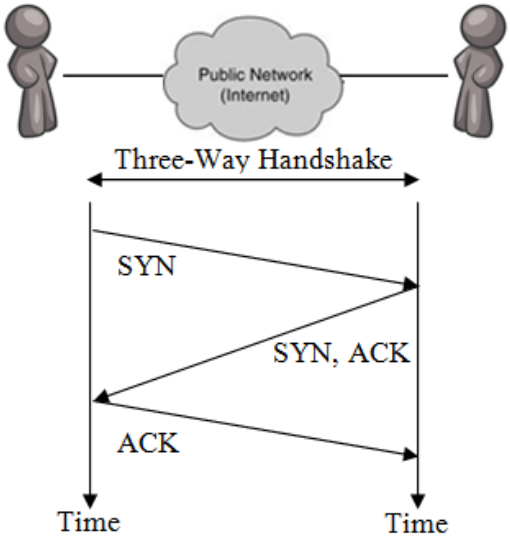
δέχεται συνεχώς αιτήσεις. Κατά συνέπεια, πολλαπλοί πελάτες εξυπηρετούνται ταυτόχρονα και γρηγορότερα.

Το μοντέλο πελάτη – εξυπηρετητή χρησιμοποιείται τόσο για τοπικά δίκτυα (LAN) όσο και για το Διαδίκτυο. Για τους σκοπούς αυτής της διπλωματικής εργασίας, οι αλγόριθμοι αποσφαλματώθηκαν τοπικά, αλλά σχεδιάστηκαν για να εφαρμοστούν σε ένα ρεαλιστικό δίκτυο. Σε ένα τέτοιο δίκτυο, το μοντέλο αυτό προσφέρει τέτοια κατανεμημένη επεξεργασία, ώστε το υπολογιστικό βάρος των εφαρμογών να μειώνεται.

3.2 Πρωτόκολλο Ρύθμισης Μεταβιβάσεων

Όπως είδαμε στο μοντέλο πελάτη – εξυπηρετητή, οι διεργασίες ανταλλάσσουν μεταξύ τους μηνύματα. Η αξιοπιστία στην επικοινωνία είναι σημαντική και κατά συνέπεια πρέπει να επιλεγεί κάποιο πρωτόκολλο που να εγγυάται την ασφαλή αποστολή των μηνυμάτων. Το πρωτόκολλο ρύθμισης μεταβιβάσεων (TCP) [18] είναι ένα ιδιαίτερα αξιόπιστο και συνδεσιστρεφές (connection – oriented) πρωτόκολλο για την επικοινωνία δύο διεργασιών.

Το TCP είναι συνδεσιστρεφές γιατί δύο διεργασίες χρειάζεται να κάνουν τριμερή χειραψία (three – way handshake) (Σχήμα 3.2) προτού στείλουν οποιαδήποτε μηνύματα μεταξύ τους. Με τη χειραψία οι δύο διεργασίες καθορίζουν τις παραμέτρους της σύνδεσης τους προτού αρχίσουν να επικοινωνούν.



Σχήμα 3.2: Χειραψία three – way handshake

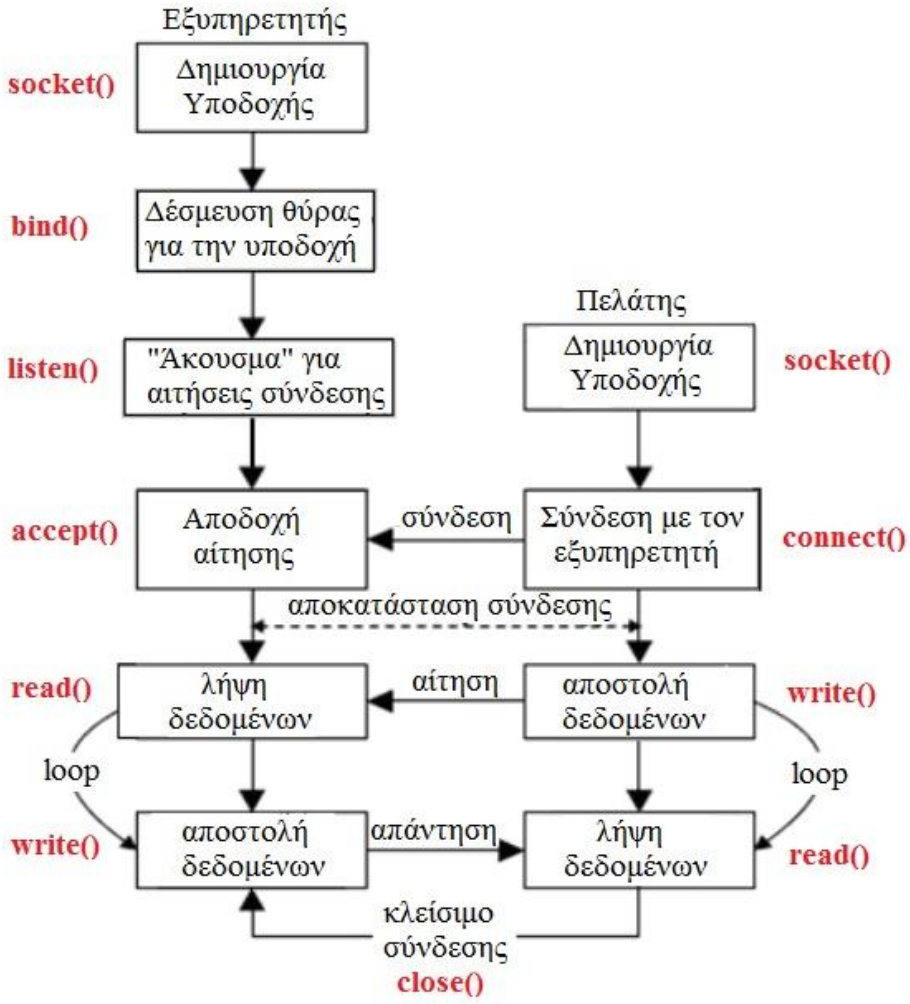
Μια σύνδεση που χρησιμοποιεί TCP συμβαίνει μεταξύ δύο σημείων, όπου το ένα είναι ο αποστολέας και το άλλο ο δέκτης. Ο αποστολέας μπορεί να στείλει αυθαίρετο αριθμό bytes με την εγγύηση ότι θα φτάσουν στο δέκτη με την ίδια σειρά που στάλθηκαν. Επιπλέον σε μια τέτοια σύνδεση η μεταφορά δεδομένων είναι αμφίδρομη, αφού είναι εφικτή η ροή δεδομένων από το ένα σημείο στο άλλο και αντίστροφα.

3.3 Προγραμματισμός με Υποδοχές Ροής

Οι αλγόριθμοι SIMPLE και CWFR απαιτείται να λειτουργούν σωστά μεταξύ διεργασιών που βρίσκονται σε διαφορετικές τοποθεσίες σε παγκόσμια κλίμακα, οι οποίες επικοινωνούν μέσω του διαδικτύου. Για το λόγο αυτό, η υλοποίηση της TCP σύνδεσης έγινε στη γλώσσα προγραμματισμού C, όπου χρησιμοποιήθηκαν κατάλληλες συναρτήσεις που προσφέρονται σε αυτή.

Να υπενθυμίσουμε πως στους αλγορίθμους έχουμε τρία είδη διεργασιών, εκ των οποίων τα δύο έχουν το ρόλο του πελάτη και το τρίτο το ρόλο του εξυπηρετητή. Στη σύνδεση TCP, ο πελάτης είναι αυτός που θα ζητήσει επικοινωνία με τον εξυπηρετητή και ο εξυπηρετητής είναι αυτός που θα την αποδεχτεί. Στο Σχήμα 3.3 απεικονίζεται η διαδικασία εγκαθίδρυσης επικοινωνίας και ανταλλαγής μηνυμάτων στο μοντέλο πελάτη – εξυπηρετητή. Προτού υπάρξει οποιαδήποτε σύνδεση, ο εξυπηρετητής χρειάζεται να είναι σε ετοιμότητα και να τρέχει. Ως εκ τούτου, δημιουργεί την υποδοχή ροής του με τη συνάρτηση socket() και δεσμεύει μια θύρα σε αυτή με την bind(), στην οποία θα δέχεται τις αιτήσεις των πελατών. Τότε, ο εξυπηρετητής περιμένει και «ακούει» τυχόν αιτήσεις με την listen(). Ακολούθως, κάποια διεργασία πελάτη δημιουργεί τη δική της υποδοχή και καθορίζει τη θύρα και τη διεύθυνση του εξυπηρετητή. Αυτά είναι αναγκαία για να ξεκινήσει τη τριμερή χειραψία, στέλνοντας αίτηση σύνδεσης στον εξυπηρετητή με την connect(). Με τη σειρά του ο εξυπηρετητής, ακούγοντας την αίτηση, την αποδέχεται εκτελώντας την accept() και δημιουργεί μια νέα υποδοχή ροής την οποία χρησιμοποιεί αποκλειστικά για τη συγκεκριμένη σύνδεση. Παρατηρούμε πως η τριμερή χειραψία γίνεται «σιωπηλά» μεταξύ της υποδοχής ροής του πελάτη και της νέας υποδοχής ροής του εξυπηρετητή. Μετά την αποκατάσταση της σύνδεσης, οι δύο διεργασίες στέλνουν και λαμβάνουν μηνύματα με τις συναρτήσεις read() και write()

μέχρις ότου ο πελάτης στείλει μήνυμα με τη close() για τερματισμό της επικοινωνίας. Με το κλείσιμο της σύνδεσης, η νέα υποδοχή ροής του εξυπηρετητή διαγράφεται.



Σχήμα 3.3: Διαδικασία εγκαθίδρυσης επικοινωνίας μεταξύ πελάτη και εξυπηρετητή

3.3.1 Είδη Υποδοχών

Υπάρχουν δύο σημαντικά είδη υποδοχών [7]: οι υποδοχές ροής (stream sockets) για συνδέσεις TCP και οι τηλεγραφικές υποδοχές (datagram sockets) για συνδέσεις UDP. Όπως επισημάνθηκε, στις υποδοχές ροής απαιτείται η τριμερής σύνδεση, προσφέροντας αξιοπιστία στην επικοινωνία και διαδοχικότητα μεταξύ των μηνυμάτων. Αντιθέτως, στις τηλεγραφικές υποδοχές δεν χρειάζεται να προηγηθεί σύνδεση μεταξύ των διεργασιών. Για το λόγο αυτό ονομάζονται και connectionless sockets. Με αυτή τη σύνδεση δεν υπάρχει αξιοπιστία και τα μηνύματα μπορεί να λαμβάνονται με

διαφορετική σειρά από αυτή που σταλθήκαν. Στους αλγορίθμους μας χρησιμοποιούνται οι υποδοχές ροής, διότι απαιτείται αξιόπιστη μεταφορά των δεδομένων.

3.3.2 Διεύθυνση IP

Μια διεύθυνση IP (Internet Protocol Address) είναι ένας αριθμός που ανατίθεται σε κάθε συσκευή που συνδέεται στο διαδίκτυο. Χρησιμοποιείται για τη μοναδική αναγνώριση της συσκευής αυτής και αντιστοιχείται με το όνομα της μηχανής μέσω μιας υπηρεσίας εύρεσης δικτύου (DNS – Domain Name Server) για διευκόλυνση των χρηστών. Υπάρχουν δύο κύριες εκδόσεις IP: το Internet Protocol Version 4 (IPv4) και το Internet Protocol Version 6 (IPv6). Το IPv4 χρησιμοποιεί διευθύνσεις των 32-bit (4 bytes) και γράφεται στη μορφή τεσσάρων οκτάδων χωρισμένων με τελεία. Λόγω του περιορισμένου πλήθους διευθύνσεων, τα τελευταία χρόνια γίνεται στροφή στο IPv6, το οποίο χρησιμοποιεί διευθύνσεις των 128-bit και σαφώς το πλήθος διευθύνσεων είναι κατά πολύ μεγαλύτερο.

Όπως είπαμε, η διεύθυνση χρησιμοποιείται για την αναγνώριση του υπολογιστή. Με το ίδιο σκεπτικό η θύρα (port), η οποία σχηματίζεται από ένα 16-bit αριθμό, χρησιμοποιείται για την αναγνώριση συγκεκριμένης διεργασίας στον υπολογιστή.

3.3.2.1 Αναπαράσταση Διευθύνσεων

Όταν δύο διεργασίες επικοινωνούν μεταξύ τους πάνω στον ίδιο υπολογιστή, τα bytes μεταφέρονται στη ίδια σειρά από τη μια διεργασία στην άλλη. Όταν όμως δύο διεργασίες βρίσκονται σε διαφορετικούς υπολογιστές, τότε πρέπει να ληφθεί υπόψη η αρχιτεκτονική της κάθε μηχανής. Διαφορετικές αρχιτεκτονικές αποθηκεύουν τα bytes με διαφορετική σειρά.

Οι Big Endian αρχιτεκτονικές αποθηκεύουν το Most Significant Byte (MSB) πρώτα, δηλαδή με τη σειρά που είναι. Αντίθετα, οι Little Endian αρχιτεκτονικές αποθηκεύουν το Least Significant Byte (LSB) πρώτα, δηλαδή η διεύθυνση αποθηκεύεται αντίστροφα. Το TCP/IP που χρησιμοποιούμε εμείς, χρησιμοποιεί την big endian διάταξη.

Η διάταξη big endian καλείται επίσης διάταξη δικτύου (Network Byte Order), διότι το δίκτυο χρησιμοποιεί αυτή τη διάταξη για την αποθήκευση των IP διευθύνσεων και των θυρών. Επιπλέον, κάθε υπολογιστής φυλάει τις διευθύνσεις σε διάταξη μηχανής (Host Byte Order) η οποία μπορεί να είναι είτε big endian είτε little endian.

Εφόσον δεν γνωρίζουμε τη διάταξη που χρησιμοποιεί ο κάθε υπολογιστής, είναι αναγκαία η μετατροπή της διάταξης μηχανής σε διάταξη δικτύου προτού σταλεί κάποιο μήνυμα μέσω του δικτύου. Παρομοίως, όταν το μήνυμα φτάσει σε κάποια διεργασία χρειάζεται η μετατροπή της διάταξης δικτύου σε διάταξη μηχανής για να την αναγνωρίζει η μηχανή στην οποία βρίσκεται η διεργασία.

Για τις πιο πάνω μετατροπές, προσφέρονται οι εξής τέσσερις συναρτήσεις:

```
#include <arpa/inet.h>

uint32_t htonl(uint32_t hostint32); // host to network long
uint16_t htons(uint16_t hostint16); // host to network short
uint32_t ntohl(uint32_t netint32);  // network to host long
uint16_t ntohs(uint16_t netint16);  // network to host short
```

Οι δύο πρώτες συναρτήσεις μετατρέπουν τον αριθμό από διάταξη μηχανής σε διάταξη δικτύου, ενώ οι δύο επόμενες μετατρέπουν τον αριθμό από διάταξη δικτύου σε διάταξη μηχανής. Ο αριθμός μπορεί να είναι είτε 32-bit (long) είτε 16-bit (short), χρησιμοποιώντας την αντίστοιχη συνάρτηση.

3.3.2.2 Δομές Διευθύνσεων

Μια πρόσφατη δομή δεδομένων, η *addrinfo*, δημιουργήθηκε για να προετοιμάζει τις δομές υποδοχής διεύθυνσης πριν από τη σύνδεση πελάτη εξυπηρετητή. Χρησιμοποιείται, επίσης, στην αναζήτηση του ονόματος της μηχανής ή κάποιας υπηρεσίας.

```
struct addrinfo {
int              ai_flags;     // AI_PASSIVE, AI_CANONNAME,…
int              ai_family;    // AF_INET, AF_INET6, AF_UNSPEC
int              ai_socktype;  // SOCK_STREAM, SOCK_DGRAM
int              ai_protocol;  // use 0 for "any"
size_t           ai_addrlen;   // size of ai_addr in bytes
struct sockaddr *ai_addr;      // struct sockaddr_in or _in6
char            *ai_canonname; // full canonical hostname
struct addrinfo *ai_next;      // linked list, next node
};
```

Το πεδίο `ai_family` προσδιορίζει τη φύση της επικοινωνίας. Η AF_INET προσδιορίζει το IPv4 internet domain, ομοίως η AF_INET6 προσδιορίζει το IPv6 internet domain, ενώ με τη AF_UNSPEC δεν χρειάζεται να ορίσουμε την έκδοση της IP. Το πεδίο `ai_socktype` περιγράφει το τύπο της υποδοχής – SOCK_STREAM, SOCK_DGRAM. Λόγω της απαιτούμενης αξιοπιστίας δουλεύουμε με το SOCK_STREAM. Για τον ίδιο λόγο, στο `ai_protocol` που προσδιορίζει το πρωτόκολλο, επιλέγουμε το IPPROTO_TCP. Αναφορικά, να πούμε πως αν δώσουμε τη τιμή 0 σε αυτό το πεδίο, τότε το πρωτόκολλο μπορεί να είναι οποιοδήποτε. Το `ai_addr` κρατά τη δυαδική διεύθυνση, ενώ το `ai_addrlen` κρατά το μέγεθος της διεύθυνσης σε bytes. Τέλος, το `ai_next` είναι ένας δείκτης σε μια ίδιου τύπου δομή.

Με τη συνάρτηση `getaddrinfo()`, αποκτούμε πρόσβαση στις πιο πάνω πληροφορίες. Δίνοντας τρεις παραμέτρους, η συνάρτηση επιστρέφει στο δείκτη `res` μια συνδεδεμένη λίστα με διευθύνσεις.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
```

Στη παράμετρο `node` δίνουμε το όνομα της μηχανής που θέλουμε να ενωθούμε ή την IP διεύθυνση της. Η παράμετρος `service` μπορεί να είναι ο αριθμός θύρας ή το όνομα κάποιας συγκεκριμένη υπηρεσίας (π.χ. http, ftp). Τέλος, η παράμετρος `hints` φιλτράρει τα αποτελέσματα που θα επιστρέψει η συνάρτηση. Αν δώσουμε NULL στο hints, τότε δεν μας ενδιαφέρει το protocol family, ο τύπος της υποδοχής ή το πρωτόκολλο που θα παρουσιαστεί στη λίστα `res`.

Η δομή δεδομένων `struct sockaddr` που συναντήσαμε πιο πάνω είναι μια γενική δομή διεύθυνσης υποδοχής, στην οποία μορφοποιούνται (casting) οι διάφορες μορφές διευθύνσεων πριν περάσουν σε συναρτήσεις υποδοχών. Φυλάει πληροφορίες αναφορικά με τη διεύθυνση για διαφορετικούς τύπους υποδοχών και τα πεδία της φαίνονται πιο κάτω:

```
#include <netinet/in.h>
struct sockaddr {
    unsigned short sa_family; // Address family
    char sa_data[14];         // 14 bytes of protocol address
};
```

Το πεδίο *sa_data* περιλαμβάνει τη διεύθυνση του προορισμού καθώς και τη θύρα υποδοχής. Για ευκολότερη χρήση των δεδομένων της *struct sockaddr*, δημιουργήθηκαν οι `struct sockaddr_in` και `struct sockaddr_in6` (όπου το «in» για το «internet») για να χρησιμοποιούνται με τα IPv4 και IPv6 αντίστοιχα.

```
// For IPv4 address
struct in_addr {
    uint32_t s_addr;            // IPv4 32-bit address
};
struct sockaddr_in {
    short int sin_family;       // Address family, AF_INET
    unsigned short int sin_port;// Port number
    struct in_addr sin_addr;    // Internet address
    unsigned char sin_zero[8];  // filler
};
```

```
// For IPv6 address
struct in6_addr {
    unsigned char   s6_addr[16]; // IPv6 address
};
struct sockaddr_in6 {
    u_int16_t sin6_family;       // Address family, AF_INET6
    u_int16_t sin6_port;         // Port number
    u_int32_t sin6_flowinfo;     // IPv6 flow information
    struct in6_addr sin6_addr;   // IPv6 address
    u_int32_t sin6_scope_id;     // Scope ID
};
```

Σημαντικό πλεονέκτημα των δομών αυτών είναι πως ένας δείκτης στο struct sockaddr_in ή στο struct sockaddr_in6 μπορεί πολύ εύκολα να μορφοποιηθεί σε δείκτη για το struct sockaddr και αντίστροφα. Να σημειώσουμε πως το πεδίο `sin_zero` πρέπει να τίθεται σε μηδενικά, χρησιμοποιώντας τη συνάρτηση memset().

```
#include <string.h>
void * memset(void *dest, int c, size_t count);
```

Η συνάρτηση αυτή δίνει τη τιμή `c` στα πρώτα `count` στοιχεία του αντικειμένου που δείχνει ο δείκτης `dest`.

Τέλος, να αναφέρουμε τη δομή δεδομένων `sockaddr_storage` η οποία σχεδιάστηκε για να χωράει διαφορετικές δομές διευθύνσεων και χρησιμοποιείται στις περιπτώσεις που δεν γνωρίζουμε αν έχουμε IPv4 ή IPv6. Αντικαθιστά κατά κάποιο τρόπο τη δομή `sockaddr`.

```
struct sockaddr_storage {
    sa_family_t  ss_family;     // Address family
    char       __ss_pad1[_SS_PAD1SIZE];
    int64_t    __ss_align;
    char       __ss_pad2[_SS_PAD2SIZE];
};
```

3.3.2.3 Μετατροπή Δυαδικών Διευθύνσεων από Διάταξη Δικτύου σε Συμβολοσειρά

Πολλές φορές, η εκτύπωση της διεύθυνσης είναι απαραίτητη. Προς διευκόλυνση του χρήστη, δημιουργήθηκαν οι πιο κάτω συναρτήσεις που μετατρέπουν μια δυαδική διεύθυνση από διάταξη δικτύου σε συμβολοσειρά και αντίστροφα.

```
#include <arpa/inet.h>

const char *inet_ntop(int domain, const void *restrict addr,
                      char *restrict str, socklen_t size);

int inet_pton(int domain, const char *restrict str,
              void *restrict addr);
```

Η `inet_ntop()` – όπου «ntop» παραπέμπει στο «network to presentation» – επιστρέφει ένα δείκτη σε συμβολοσειρά ή NULL σε περίπτωση λάθους.

Η `inet_pton()` – όπου «pton» παραπέμπει στο «presentation to network» - επιστρέφει 1 αν έγινε η μετατροπή με επιτυχία ή -1 σε περίπτωση λάθους.

Η παράμετρος `domain` προσδιορίζει τη φύση της επικοινωνίας, η `addr` δείχνει στη διεύθυνση, η `str` δείχνει στη συμβολοσειρά και το `size` στη πρώτη συνάρτηση προσδιορίζει τον μέγιστο αριθμό της συμβολοσειράς που επιθυμούμε να δούμε.

3.3.3 Δημιουργία Υποδοχής

Κάθε διεργασία με υποδοχή χρειάζεται ένα περιγραφέα υποδοχής για να έχουν πρόσβαση άλλες διεργασίες σε αυτή. Οι περιγραφείς υποδοχής λειτουργούν σαν τους περιγραφείς αρχείων. Για τη δημιουργία υποδοχής χρησιμοποιείται η συνάρτηση `socket()` η οποία επιστρέφει το περιγραφέα της υποδοχής. Σε περίπτωση λάθους επιστρέφει -1. Οι παράμετροι αυτής της συνάρτησης είναι η δομή της διεύθυνσης, ο τύπος υποδοχής και το πρωτόκολλο. Στις παρούσες υλοποιήσεις, δίνονται οι παράμετροι `AF_INET`, `SOCK_STREAM` και 0 αντίστοιχα. Με τη τιμή 0, δηλώνουμε πως θα χρησιμοποιηθεί το εξ' ορισμού πρωτόκολλο που αντιστοιχεί στις άλλες δύο παραμέτρους. Στη περίπτωση μας, θα χρησιμοποιηθεί το TCP.

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

3.3.4 Δέσμευση Θύρας για την Υποδοχή

Αφότου δημιουργηθεί η υποδοχή, χρειάζεται να δεσμευτεί μια θύρα της μηχανής για τη συγκεκριμένη διεργασία. Αυτό γίνεται συνήθως από την πλευρά του εξυπηρετητή, ο οποίος αργότερα θα «ακούει» μέσω αυτής της θύρας. Επίσης, οι πελάτες πρέπει να γνωρίζουν τη θύρα του, για να μπορούν να στείλουν αίτηση για σύνδεση. Για τη διεργασία του πελάτη, αφήνεται το σύστημα να επιλέξει μία διαθέσιμη διεύθυνση, εφόσον δεν θα χρειαστεί να τη γνωρίζουμε. Η συνάρτηση `bind()` που φαίνεται πιο κάτω είναι υπεύθυνη για αυτή τη συσχέτιση.

```
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr,
         socklen_t len);
```

Η παράμετρος `sockfd` κρατά το περιγραφέα υποδοχής που δημιουργήθηκε από τη συνάρτηση connect(). Ο δείκτης `addr` δείχνει σε μια δομή `struct sockaddr` η οποία περιέχει πληροφορίες για τη διεύθυνση. Τέλος, η `len` κρατά το μέγεθος σε bytes του `addr`. Σε περίπτωση επιτυχίας επιστρέφει 0, διαφορετικά -1.

Κάποιες φορές παρουσιάστηκε αποτυχία από μέρους της `bind()` να δεσμεύσει κάποια θύρα, δίνοντας το μήνυμα "`Address already in use`". Αυτό το μήνυμα δηλώνει πως η θύρα την οποία προσπαθούμε να δεσμεύσουμε δεν έχει αποδεσμευτεί από προηγούμενη κλήση της συνάρτησης. Για αποφυγή αυτού το προβλήματος, καλούμε τη συνάρτηση `setsockopt()`, η οποία χρησιμοποιείται για χειρισμό διαφόρων επιλογών των υποδοχών.

```
#include <sys/socket.h>
int getsockopt(int socket, int level, int option_name,
               void *restrict option_value,
               socklen_t *restrict option_len);
```

Συγκεκριμένα, δίνοντας SOL_SOCKET στο `level`, προσδιορίζουμε πως θέλουμε να χειριστούμε το επίπεδο της υποδοχής και με την εισαγωγή της επιλογής SO_REUSEADDR στη παράμετρο `option_name`, διασφαλίζεται η δυνατότητα επαναχρησιμοποίησης της θύρας. Οι παραμέτροι `option_value` και `option_len` επιστρέφουν συμβολοσειρά με το αποτέλεσμα που μπορεί να απαιτεί η δοθείσα επιλογή και το μέγεθος της συμβολοσειράς. Σε αυτή τη περίπτωση, μπορούν να μείνουν κενά, αφού δεν είναι αναγκαία. Σε περίπτωση επιτυχίας επιστρέφει 0, διαφορετικά -1.

3.3.5 «Άκουσμα» αιτήσεων σύνδεσης

Ο εξυπηρετητής, αφού έχει δημιουργήσει μια υποδοχή και την έχει συσχετίσει με μια θύρα, ανακοινώνει πως είναι έτοιμος να δεχτεί αιτήσεις σύνδεσης στη θύρα `sockfd`. Αυτό επιτυγχάνεται με τη συνάρτηση `listen()`.

```
#include <sys/types.h>
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

Η παράμετρος `sockfd` είναι ο περιγραφέας υποδοχής που δημιουργήθηκε από την socket(). Η `backlog` κρατά το μέγιστο αριθμό αιτήσεων που μπορούν να περιμένουν

σε ουρά για να τις αναλάβει ο εξυπηρετητής. Η μεγαλύτερη τιμή που μπορεί να πάρει είναι SOMAXCONN. Αν ο αριθμός αιτήσεων ξεπεράσει το μέγιστο αριθμό, τότε οι επιπρόσθετες αιτήσεις απορρίπτονται, μέχρι να εξυπηρετηθούν κάποιες από την ουρά και υπάρξει χώρος στην ουρά. Σε περίπτωση επιτυχίας, επιστρέφει 0, διαφορετικά -1.

3.3.6 Σύνδεση με Εξυπηρετητή

Όπως περιγράφηκε το μοντέλο πελάτη – εξυπηρετητή, για να υπάρξει επικοινωνία μεταξύ των δύο διεργασιών χρειάζεται να προηγηθεί η τριμερής χειραψία, την οποία ξεκινά ο πελάτης. Το βήμα αυτό είναι εφικτό μόνο εάν ο εξυπηρετητής είναι έτοιμος να δεχτεί αιτήσεις και αν έχει ήδη δημιουργήσει υποδοχή ο πελάτης. Με τη συνάρτηση connect() ο πελάτης ξεκινά τη χειραψία στέλνοντας αίτηση σύνδεσης.

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd, struct sockaddr *serv_addr,
            int addrlen);
```

Το sockfd είναι ο περιγραφέας υποδοχής του πελάτη, ο οποίος αν δεν έχει συσχετιστεί προηγουμένως με κάποια διεύθυνση, θα γίνει αυτόματα από τη συνάρτηση. Το serv_addr είναι δείκτης στη διεύθυνση υποδοχής του εξυπηρετητή και το addrlen κρατά το μέγεθος της διεύθυνσης αυτής. Σε περίπτωση επιτυχίας, επιστρέφει 0, διαφορετικά -1.

3.3.7 Αποδοχή αίτησης σύνδεσης

Καθώς ο εξυπηρετητής βρίσκεται στο στάδιο που «ακούει» για τυχόν αιτήσεις, μόλις καταφθάσει κάποια εισερχόμενη αίτηση σύνδεσης στην υποδοχή του sockfd, την αποδέχεται με τη συνάρτηση accept(), ολοκληρώνοντας έτσι τη τριμερή χειραψία.

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr,
           socklen_t *addrlen);
```

Σε περίπτωση επιτυχίας, η συνάρτηση επιστρέφει το νέο περιγραφέα υποδοχής της σύνδεσης και στις παραμέτρους `addr` και `addrlen` αποθηκεύονται πληροφορίες για τη διεύθυνση του πελάτη που συνδέθηκε και το μέγεθος της διεύθυνσης του. Σε περίπτωση αποτυχίας, επιστρέφει -1.

3.3.8 Ανταλλαγή Δεδομένων

Αφού έγινε πλέον η σύνδεση, οι δύο διεργασίες αρχίζουν να ανταλλάζουν μηνύματα. Να διευκρινίσουμε πως στο σύστημα UNIX, οι περιγραφείς υποδοχής υλοποιούνται ως περιγραφείς αρχείων, γι' αυτό διάφορες συναρτήσεις των περιγραφέων αρχείων, δουλεύουν και με τους περιγραφείς υποδοχής. Για παράδειγμα, οι απλές συναρτήσεις read() και write() χρησιμοποιούνται με τους περιγραφείς αρχείων, αλλά πολύ εύκολα χρησιμοποιούνται στην υλοποίηση των δύο αλγορίθμων για τη λήψη και αποστολή μηνυμάτων μεταξύ πελάτη εξυπηρετητή.

```
#include <sys/types.h>
#include <unistd.h>
ssize_t  write(int fd,  const  void  *buf,  size_t count);
ssize_t read(int fd, void *buf, size_t count);
```

Η συνάρτηση `write()` γράφει μέχρι `count` bytes από τη συμβολοσειρά που δείχνει το `buf`, στο αρχείο που δείχνει ο περιγραφέας αρχείου `fd`.
Η συνάρτηση `read()` διαβάζει μέχρι `count` bytes από το περιγραφέα αρχείου `fd` και τα αποθηκεύει στη συμβολοσειρά που δείχνει το `buf`.
Και οι δύο συναρτήσεις επιστρέφουν τον πραγματικό αριθμό των που έχουν γραφεί/διαβαστεί. Σε περίπτωση λάθους, επιστρέφουν -1. Στη περίπτωση επιστροφής της τιμής 0, σημαίνει πως τίποτα δεν έχει γραφτεί/διαβαστεί. Τέλος, και οι δύο συναρτήσεις είναι blocking εντολές, δυσκολία που αντιμετωπίζεται στο επόμενο υποκεφάλαιο.

3.3.9 Ταυτόχρονος Χειρισμός Υποδοχών

Όπως έχουμε προαναφέρει, η χρήση blocking εντολών είναι αναπόφευκτη. Ως εκ τούτου, χρησιμοποιούμε τη select() η οποία προσφέρει τη δυνατότητα χειρισμού πολλών υποδοχών ταυτόχρονα. Συγκεκριμένα, ανακαλύπτει ποιες υποδοχές είναι έτοιμες για διάβασμα δεδομένων, για γράψιμο δεδομένων ή ακόμη ποιες έχουν ρίξει exception.

```
#include <sys/types.h>
#include <unistd.h>
#include <sys/time.h>
int select(int numfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

Η select() παίρνει ως παραμέτρους τρία σύνολα περιγραφέων αρχείων: τα readfds, τα writefds και τα exceptfds, στα οποία βάζουμε τους περιγραφείς που θέλουμε να ελέγξουμε αν είναι έτοιμοι για διάβασμα, για γράψιμο ή αν έχουν βγάλει exception αντίστοιχα. Οποιοδήποτε σύνολο δεν μας ενδιαφέρει, το θέτουμε με `NULL`. Στην επιστροφή της συνάρτησης, στα σύνολα που μας ενδιαφέρουν, παραμένουν οι περιγραφείς που είναι έτοιμοι. Η παράμετρος `numfds` κρατά τον αριθμό του μεγαλύτερου περιγραφέα αρχείου συν ένα. Η παράμετρος `timeout` ορίζει το χρονικό όριο λήξης της συνάρτησης. Δηλαδή, η select() περιμένει περιγραφείς που να ανήκουν στο σύνολο που μας ενδιαφέρει, μέχρις ότου λήξει η χρονική περίοδος λήξης. Αν έχει βρει κάποιους περιγραφείς, τότε επιστρέφει τον αριθμό των περιγραφέων που είναι έτοιμοι. Στη περίπτωση που δεν υπάρχει έτοιμος περιγραφέας και η συνάρτηση λήξει, επιστρέφει 0, ενώ σε περίπτωση λάθους επιστρέφει -1.

Η δομή `struct timeval` περιέχει τα πιο κάτω πεδία. Αν ορίσουμε τη τιμή 0 στα πεδία, τότε η συνάρτηση δεν θα περιμένει, αλλά θα τερματίζει αμέσως με τους έτοιμους περιγραφείς. Αν θέσουμε `NULL` στα πεδία, τότε η συνάρτηση μπορεί να περιμένει για πάντα, ή τουλάχιστον μέχρι να είναι έτοιμος κάποιος περιγραφέας.

```
struct timeval {
    int tv_sec;             // Seconds
    int tv_usec;            // Microseconds
};
```

Επιπρόσθετα, για να έχουμε πρόσβαση στα σύνολα fd_set μπορούμε να χρησιμοποιούμε τις πιο κάτω μακροεντολές (macros):

```
FD_SET(int fd, fd_set *set);  // Πρόσθεσε το fd στο set.
FD_CLR(int fd, fd_set *set);  // Αφαίρεσε το fd από το set.
FD_ISSET(int fd, fd_set *set);// Επίστρεψε true εάν το fd
                              // υπάρχει στο set.
FD_ZERO(fd_set *set);         // Άδειασε το set.
```

3.3.10 Κλείσιμο Υποδοχών

Για το κλείσιμο των περιγραφέων υποδοχής, μπορούμε να καλέσουμε τη συνάρτηση `close()`. Υπάρχει ακόμη η δυνατότητα κλήσης της `shutdown()`, η οποία επιτρέπει την απενεργοποίηση της λήψης, της μετάδοσης στη σύνδεση ή και των δύο.

```
close(sockfd);

#include <sys/socket.h>
int shutdown(int sockfd, int how);
```

Η παράμετρος μπορεί πάρει κάποια από τις πιο κάτω τιμές:

- 0: απενεργοποιείται το διάβασμα από την υποδοχή.
- 1: απενεργοποιείται η αποστολή από την υποδοχή.
- 2: απενεργοποιούνται και οι δύο δυνατότητες (όμοιο με τη close()).

Στη περίπτωση επιτυχίας επιστρέφει 0, διαφορετικά -1.

3.4 Νήματα

Να ανακαλέσουμε από το υποκεφάλαιο 3.1, πως υπάρχουν δύο κατηγορίες εξυπηρετητών. Στις υλοποιήσεις μας, χρησιμοποιούνται οι ταυτόχρονοι εξυπηρετητές για να αναλαμβάνουν πολλούς πελάτες ταυτόχρονα και έτσι να εξυπηρετούνται γρηγορότερα. Για να είναι αυτό εφικτό, εισάγουμε τα νήματα (threads) [14,15], ένα εργαλείο που χρησιμοποιείται για τη κοινή χρήση της μνήμης.

Το νήμα είναι η βασική μονάδα που χρονοδρομολογείται από το λειτουργικό σύστημα. Μια διεργασία είναι στη πραγματικότητα ένα νήμα. Με τη βιβλιοθήκη pthread.h, μπορούμε να δημιουργήσουμε πολλά νήματα εντός μιας διεργασίας, τα οποία μοιράζονται τον ίδιο χώρο διευθύνσεων.

Οι πρόσβαση σε κοινούς πόρους είναι και ο κυριότερος λόγος που τα νήματα προτιμούνται έναντι της δημιουργίας άλλων διεργασιών με τη fork(). Με τη fork(), η κάθε διεργασία έχει το δικό της χώρο μνήμης και η επικοινωνία μεταξύ των διεργασιών είναι δύσκολη. Επομένως, η πολυπλοκότητα αυξάνεται, αντιθέτως με τα νήματα όπου δεν υπάρχουν τέτοια προβλήματα.

Τέλος, για τη μεταγλώττιση ενός αρχείου που χρησιμοποιεί νήματα, χρειάζεται να προστεθεί στο τέλος της εντολής η επιλογή που φαίνεται πιο κάτω:

```
gcc –o <filename> <filename>.c –lpthread
```

3.4.1 Δημιουργία Νήματος

Όταν δημιουργείται ένα νήμα, χρειάζεται να του δώσουμε κάποια χαρακτηριστικά και να του δείξουμε τι θα τρέξει με το που δημιουργείται. Επομένως, η πιο κάτω συνάρτηση δημιουργεί ένα νήμα δίνοντας του τα χαρακτηριστικά που ορίζονται στο `attr` και δείχνοντας του, στη τρίτη παράμετρο, τη συνάρτηση που θα τρέξει. Αν το `attr` τεθεί σε `NULL` τότε θα χρησιμοποιηθούν τα εξ' ορισμού χαρακτηριστικά ενός νήματος.

```
#include <pthread.h>
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
                   void *(* start_routine )(void *), void *arg);
```

Η συνάρτηση που θα τρέξει το νήμα πρέπει να έχει τύπο επιστροφής void, αφού δεν καλείται με το παραδοσιακό τρόπο και δεν μπορεί να φυλάξει τιμή επιστροφής σε μεταβλητή. Η τελευταία παράμετρος είναι ένας δείκτης στη παράμετρο της συνάρτησης. Αν υπάρχουν περισσότερες από μια παραμέτρους, τότε πρέπει να αποθηκευτούν σε μια δομή και ο δείκτης να δείχνει στη δομή αυτή. Αν δεν υπάρχουν παράμετροι, τότε η `arg` τίθεται σε `NULL`. Αν ολοκληρωθεί με επιτυχία, η συνάρτηση φυλάει στο `thread` την ταυτότητα του νήματος που δημιουργήθηκε και επιστρέφει τη τιμή 0. Διαφορετικά επιστρέφει κάποιο κωδικό λάθους.

3.4.2 Μηχανισμός Συγχρονισμού

Ένας εξυπηρετητής δημιουργεί καινούργιο νήμα για την εξυπηρέτηση ενός νέου πελάτη, με αποτέλεσμα πολλά νήματα ενός εξυπηρετητή να λειτουργούν ταυτόχρονα έχοντας πρόσβαση σε κοινή μνήμη. Για αυτό το λόγο, κάποιος μηχανισμός συγχρονισμού είναι απαραίτητος για πρόληψη των προβλημάτων ασυνέπειας των δεδομένων και για καθορισμό της σειράς εκτέλεσης ταυτόχρονων λειτουργιών στο ίδιο κομμάτι της μνήμης.

Ο μηχανισμός που χρησιμοποιούμε είναι τα mutexes (**mut**ual **ex**clusion). Κάθε global μεταβλητή στην όποια έχουν πρόσβαση περισσότερα από ένα νήματα, πρέπει να συσχετίζεται με ένα mutex, για να προστατεύει τη κρίσιμη περιοχή από ταυτόχρονες προσβάσεις. Όπως και στα νήματα, ένα mutex εφαρμόζεται εντός μιας και μόνο διεργασίας και δεν μπορεί να λειτουργήσει μεταξύ πολλών διεργασιών.

Ένα mutex έχει δύο πιθανές καταστάσεις:

- Unlocked: Δεν είναι κλειδωμένο από κανένα νήμα, άρα κανένα δεν έχει πρόσβαση στη συγκεκριμένη global μεταβλητή.
- Locked: Κάποιο νήμα το έχει κλειδώσει και χρησιμοποιεί το συγκεκριμένο κομμάτι μνήμης.

Δεν επιτρέπεται ένα mutex να κλειδωθεί από δύο νήματα ταυτόχρονα.

Με τη συνάρτηση `pthread_mutex_lock()`, αν το mutex είναι ακλείδωτο, τότε κλειδώνεται και ανήκει πλέον στο νήμα που το κλείδωσε. Η συνάρτηση επιστρέφει

αμέσως, διαφορετικά αν είναι ήδη κλειδωμένο τότε το νήμα που το κάλεσε, αναστέλλεται μέχρι να ξεκλειδωθεί το mutex.

Με τη συνάρτηση `pthread_mutex_unlock()`, αν το mutex είναι κλειδωμένο, τότε ξεκλειδώνεται και η συνάρτηση επιστρέφει αμέσως. Αν το mutex είναι ήδη ξεκλείδωτο ή είναι κλειδωμένο από κάποιο άλλο νήμα, τότε η συνάρτηση επιστρέφει κωδικό λάθους.

Ποιο κάτω παρουσιάζονται οι δύο αυτές συναρτήσεις:

```
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

3.4.3 Μηχανισμός Αναμονής/Συνέχειας Νήματος

Ο εξυπηρετητής προτού αρχίσει να δέχεται αιτήσεις, δημιουργεί ένα σύνολο νημάτων που θα βρίσκονται στη διάθεση των πελατών, καθώς επίσης ένα νήμα – υπεύθυνο των υπολοίπων. Το σύνολο των νημάτων, με το που δημιουργούνται, αρχίζουν να τρέχουν τη συνάρτηση που τους ορίστηκε ως το σημείο έναρξης. Αφού έχουν αρχίσει προτού λάβει ο εξυπηρετητής κάποιες αιτήσεις, χρειάζεται με κάποιο τρόπο να σταματήσουν και να περιμένουν. Το νήμα που χειρίζεται αυτό το σύνολο, είναι υπεύθυνο για τη επανεκκίνηση τους. Έτσι εισάγεται σε αυτό το σημείο ο μηχανισμός αναμονής του νήματος και επανέναρξης της λειτουργίας του.

Ο μηχανισμός αυτός λέγεται condition variable και είναι τύπου pthread_cond_t. Επιτρέπει στα νήματα να αναστείλουν την εκτέλεση τους και να ελευθερώσουν τον επεξεργαστή μέχρι κάποια συγκεκριμένη συνθήκη να γίνει true. Μια condition variable πρέπει να συσχετίζεται με κάποιο mutex, ούτως ώστε να αποφευχθεί η περίπτωση ταυτόχρονης χρήσης της από δύο νήματα, κάτι που θα οδηγούσε σε αδιέξοδο. Ένα παράδειγμα που δείχνει αυτή τη περίπτωση είναι το εξής: κάποιο νήμα μπορεί να ετοιμάζεται να περιμένει, ενώ ένα δεύτερο νήμα που περιμένει ήδη με την ίδια condition variable, λαμβάνει το σήμα να προχωρήσει. Το πρώτο νήμα θα παραμείνει να περιμένει χωρίς να λάβει το σήμα ποτέ, αφού το έλαβε το δεύτερο νήμα.

Πιο κάτω βλέπουμε τις δύο συναρτήσεις αυτού του μηχανισμού, που χρησιμοποιούνται στο κώδικα:

```
#include <pthread.h>
int pthread_cond_wait(pthread_cond_t *cond,
                      pthread_mutex_t *mutex);
int pthread_cond_signal(pthread_cond_t *cond);
```

Η συνάρτηση `pthread_cond_wait()` καλείται με παραμέτρους τη `cond` την οποία μπλοκάρει και το `mutex` το οποίο πρέπει να έχει ήδη κλειδωθεί από το νήμα. Αν το `mutex` δεν έχει κλειδωθεί προηγουμένως, η συμπεριφορά της συνάρτησης είναι απροσδιόριστη. Με τη κλήση της, η pthread_cond_wait() αναγκάζει το νήμα να μπλοκαριστεί στη `cond` και να περιμένει μέχρι η `cond` ξεμπλοκαριστεί από τη συνάρτηση pthread_cond_signal().

Η συνάρτηση `pthread_cond_signal()` ξεμπλοκάρει τουλάχιστον ένα από τα νήματα που είναι κλειδωμένα στην ίδια condition variable, την `cond`. Το νήμα που ξεμπλοκάρεται συνεχίζει κανονικά τη λειτουργία του.

3.5 Περιγραφή Κύριων Συστατικών των προγραμμάτων

Αν και έχουμε περιγράψει αρκετά τις διεργασίες του συστήματος, εντούτοις εδώ θα αναφερθούμε σε περισσότερες λεπτομέρειες ως προς την υλοποίηση τους. Στο Παράρτημα Α βρίσκεται ο κώδικας των αλγορίθμων. Η υλοποίηση έχει γίνει στη γλώσσα προγραμματισμού C, στο λειτουργικό σύστημα Ubuntu 10.4, με τη χρήση του μοντέλου πελάτη – εξυπηρετητή. Να υπενθυμίσουμε πως με τον όρο πελάτη εννοούμε δύο διαφορετικές εφαρμογές, τον αναγνώστη και τον εγγραφέα. Ο εξυπηρετητής επικοινωνεί με τις δύο εφαρμογές, απαντώντας αναλόγως των μηνυμάτων που παίρνει. Τα σύνολα των αναγνωστών και των εγγραφέων δεν μπορούν να επικοινωνήσουν μεταξύ τους, αλλά μόνο με τους εξυπηρετητές. Ομοίως, δεν επιτρέπεται η ανταλλαγή μηνυμάτων μεταξύ των εξυπηρετητών. Οι εξυπηρετητές ανήκουν σε κάποιο σύστημα απαρτίας, χωρίς να έχουν γνώση για αυτό. Αντίθετα, οι εγγραφείς και οι αναγνώστες γνωρίζουν από την αρχή το σύστημα απαρτίας, το οποίο είναι στατικό και δεν αλλάζει

κατά τη διάρκεια του προγράμματος. Τέλος, τα μηνύματα που ανταλλάζονται είναι της μορφής: <Τύπος μηνύματος, Tag, Τιμή αντικειμένου, Αριθμός Αίτησης>, όπου το tag αποτελείται από τη ταυτότητα του τελευταίου εγγραφέα που άλλαξε τη τιμή του αντικειμένου και τη χρονοσφραγίδα. Ο τύπος του μηνύματος μπορεί να είναι: READ, WRITE, READACK, WRITEACK, όπου τα πρώτα δύο προέρχονται μόνο από τις εφαρμογές του εγγραφέα και του αναγνώστη, ενώ τα άλλα δύο προέρχονται από τον εξυπηρετητή. Και οι δύο εφαρμογές πελάτη μπορούν να στείλουν μηνύματα READ και WRITE, αλλά μόνο ο εγγραφέας μπορεί να αλλάξει τη τιμή του αντικειμένου στέλνοντας ένα WRITE μήνυμα – αφότου έχει μάθει το πιο πρόσφατο tag στο πρώτο γύρο με το μήνυμα READ. Ο αναγνώστης διαφέρει στους δύο αλγορίθμους. Στο SIMPLE, ο αναγνώστης στέλνει μήνυμα WRITE στο δεύτερο γύρο επικοινωνίας για να ενημερώσει τους εξυπηρετητές του συστήματος απαρτίας με τη νέα τιμή που ανακάλυψε στο πρώτο γύρο. Στον αλγόριθμο CWFR, ο δεύτερο γύρος επικοινωνίας μπορεί να είναι αχρείαστος βάση του Quorum View που ανακάλυψε.

3.5.1 Εφαρμογή εξυπηρετητή (server)

Όπως αναφέραμε στο υποκεφάλαιο 2.1.5, ο εξυπηρετητής κρατά ένα αντίγραφο του αντικειμένου και ρόλος του είναι να απαντά στις αιτήσεις του πελατών. Αρχίζει τη λειτουργία του, τρέχοντας της εξής εντολή στη γραμμή εντολών του κελύφους:

```
./server <server Id> <port number> <fail frequency>
```

Το `server` είναι το όνομα του εκτελέσιμου αρχείου. Το πρώτο όρισμα είναι η ταυτότητα που δίνουμε στον εξυπηρετητή, το δεύτερο είναι ο αριθμός της θύρας στη οποία «ακούει» για νέες αιτήσεις και το τρίτο όρισμα είναι μια τιμή που καθορίζει πόση είναι η πιθανότητα να καταρρεύσει ο εξυπηρετητής κατά τη διάρκεια λειτουργίας του. Αν η τιμή είναι x τότε υπάρχει μια πιθανότητα (100-x)% να καταρρεύσει και μια πιθανότητα x να μην καταρρεύσει.

Ο εξυπηρετητής φυλάει στη τοπική του μνήμη ένα αντίγραφο του αντικειμένου, δηλαδή την τιμή του αντικειμένου (αρχικοποιημένη με -1), τη ταυτότητα του εγγραφέα και τη χρονοσφραγίδα του αντικειμένου (αρχικοποιημένα με 0). Προτού αρχίσει να λαμβάνει αιτήσεις από τους πελάτες, δημιουργεί ένα “threadpool”, δηλαδή ένα σύνολο από

threads (νήματα), κάθε ένα από τα οποία εξυπηρετεί ένα μόνο πελάτη καθ'όλη τη διάρκεια της σύνδεσης του με τον εξυπηρετητή. Όταν μια σύνδεση κλείσει, το νήμα είναι διαθέσιμο για να αναλάβει νέο πελάτη.

Σε κάθε μήνυμα που λαμβάνει ο εξυπηρετητής, κάνει δύο ελέγχους που καθορίζουν κατά πόσο θα συνεχίσει με την επεξεργασία του μηνύματος. Πρώτα ελέγχει κατά πόσο το μήνυμα είναι το πιο πρόσφατο. Αυτό υλοποιείται, χρησιμοποιώντας ένα πίνακα (Πίνακας 3.1) με δύο στήλες και με αριθμό γραμμών ίσο με το σύνολο των εγγραφέων και των αναγνωστών. Στη πρώτη στήλη φυλάει την ταυτότητα του πελάτη και στη δεύτερη φυλάει το μετρητή αιτήσεων του, ο οποίος αυξάνεται κατά ένα από τον πελάτη πριν στείλει το μήνυμα.

Ταυτότητα	Μετρητής Αιτήσεων
1	x_1
2	x_2
⋮	⋮
W+R	x_{W+R}

Πίνακας 3.1: Μετρητής Αιτήσεων για κάθε πελάτη.
Ο αριθμός γραμμών ισούται με το σύνολο των εγγραφέων(W) και των αναγνωστών(R).

Θα μπορούσε να υλοποιηθεί ένας μονοδιάστατος πίνακας έχοντας τον αριθμό του πεδίου του πίνακα ως την ταυτότητα του πελάτη. Παρόλα αυτά προτιμήθηκε αυτός ο τρόπος υλοποίησης για να είναι εφικτή η εισαγωγή τυχαίων ταυτοτήτων και όχι αναγκαστικά συνεχόμενων αριθμών. Λόγω της ασυγχρονίας του δικτύου, υπάρχει πιθανότητα κάποιο μήνυμα να μην φτάσει ποτέ στο εξυπηρετητή και κάποιο μεταγενέστερο μήνυμα να φτάσει κανονικά. Έτσι, ο πίνακας είναι αναγκαίος για να φυλάγεται ο αριθμός αίτησης του πελάτη από τη τελευταία φορά που έστειλε μήνυμα. Ο αριθμός αίτησης αυξάνεται κατά ένα από το πελάτη πριν σταλεί το μήνυμα. Επομένως, ο έλεγχος που γίνεται είναι εάν ο αριθμός αίτησης του μηνύματος είναι μεγαλύτερος από το μετρητή που κρατά στο πίνακα για το συγκεκριμένο πελάτη ο εξυπηρετητής. Εάν δεν είναι, τότε το αγνοεί.

Ο δεύτερος έλεγχος που γίνεται είναι για τη περίπτωση που ο εξυπηρετητής χρειάζεται να καταρρεύσει. Σε κάθε μήνυμα που λαμβάνει, ο εξυπηρετητής παράγει ένα τυχαίο αριθμό x μεταξύ του 0 και του 100. Αν ο αριθμός αυτός είναι μεγαλύτερος ή ίσος της παραμέτρου που λαμβάνει το πρόγραμμα κατά την έναρξη του, τότε ο εξυπηρετητής καταρρέει, δηλαδή αγνοεί το μήνυμα. Με αυτό τον τρόπο προσομοιώνεται η κατάρρευση του εξυπηρετητή, κάτι που θα βοηθήσει αργότερα στην αξιολόγηση.

3.5.2 Αρχεία Δεδομένων

Οι εφαρμογές του εγγραφέα και του αναγνώστη λαμβάνουν ως ορίσματα από την γραμμή εντολών δύο αρχεία δεδομένων. Το πρώτο περιέχει το σύστημα απαρτίας των εξυπηρετητών και το δεύτερο πληροφορίες για τους εξυπηρετητές.

Το αρχείο με το σύστημα απαρτίας περιλαμβάνει στη πρώτη γραμμή τον αριθμό των quorums. Ακολούθως, παρουσιάζονται ένα – ένα τα quorums με τους εξυπηρετητές τους και τις τομές τους με άλλα quorums. Συγκεκριμένα, κάθε quorum ξεκινά με το χαρακτήρα Q και τον αριθμό του quorum. Μετά ανοίγει παρένθεση και παρουσιάζονται οι ταυτότητες των εξυπηρετητών που ανήκουν στο συγκεκριμένο quorum χωρισμένες με κόμμα, και τέλος κλείνει η παρένθεση. Στις επόμενες γραμμές, αναγράφονται οι τομές που έχει το quorum με άλλα quorums. Κάθε τομή ξεκινά με το χαρακτήρα I (από το **I**ntersection), τον αριθμό του quorum, μια παύλα και τον αριθμό του quorum με το οποίο έχει τομή. Συνεχίζει με παρένθεση στην οποία περιέχονται οι ταυτότητες των κοινών εξυπηρετητών χωρισμένες με κόμμα. Ένα παράδειγμα αρχείου με δύο quorums θα έμοιαζε ως εξής:

```
2
Q1(1,2,3,4)
I1-2(1,4)
Q2(1,4,5,6)
I2-1(1,4)
```

Παρατηρούμε πως η τομή επαναλαμβάνεται δύο φορές στο αρχείο. Αυτό είναι αναγκαίο για τον τρόπο που έγινε η υλοποίηση των αλγορίθμων.

Το δεύτερο αρχείο περιέχει πληροφορίες για τον κάθε εξυπηρετητή του συστήματος. Στη πρώτη γραμμή αναγράφεται ο συνολικός αριθμός των εξυπηρετητών και

ακολούθως σε κάθε γραμμή δίνονται τα αναγκαία στοιχεία που πρέπει να γνωρίζουν οι πελάτες για ενωθούν μαζί τους, χωρισμένα με κενό. Πρώτα δίνεται η ταυτότητα του εξυπηρετητή, μετά η θύρα στην οποία δέχεται αιτήσεις και το όνομα της μηχανής στην οποία ανήκει (hostname). Τέλος, δίνεται μια τιμή 0 ή 1 η οποία καθορίζει αν ο εξυπηρετητής θα προσομοιώσει κατάρρευση ή όχι αντίστοιχα. Πιο κάτω δίνεται ένα παράδειγμα αρχείου με δύο εξυπηρετητές και τα στοιχεία τους.

```
2
1 10001 hostname1 1
2 10002 hostname2 1
```

3.5.3 Κοινές Λειτουργίες εφαρμογών αναγνώστη και εγγραφέα

Προτού περιγράψουμε εκτενέστερα τις εφαρμογές αναγνώστη και εγγραφέα, μπορούμε να αναφερθούμε σε κάποιες κοινές λειτουργίες που κάνουν οι δύο εφαρμογές.

Με το που ξεκινά τη λειτουργία της η εφαρμογή πελάτη, φυλάει στη μνήμη της τα δεδομένα από τα δύο αρχεία που εξηγήθηκαν πιο πάνω. Συγκεκριμένα αποθηκεύει δυναμικά σε πίνακα τα quorums και τις τομές τους, τα οποία θα καταστούν αναγκαία αργότερα που θα ελέγχει αν έλαβε απαντήσεις από ένα ολόκληρο quorum. Ακολούθως, χρειάζεται να συνδεθεί με όλους τους εξυπηρετητές του συστήματος. Ως εκ τούτου, δημιουργεί ένα πίνακα που κρατά τις υποδοχές ροής για επικοινωνία για κάθε εξυπηρετητή.

Επίσης, η συνάρτηση που στέλνει μήνυμα στους εξυπηρετητές είναι πανομοιότυπη στις δύο εφαρμογές πελάτη. Εδώ να διευκρινίσουμε πως η αποστολή των μηνυμάτων στους εξυπηρετητές γίνεται με τυχαία σειρά. Επιπλέον, η λήψη απαντήσεων από τους εξυπηρετητές γίνεται με τον ίδιο τρόπο, αφού εξαρτάται από τη καθυστέρηση που υπάρχει σε κάθε εξυπηρετητή, και μάλιστα γίνεται ταυτόχρονα με την αποστολή των μηνυμάτων. Αυτό επιτυγχάνεται με τη select() που εξηγήθηκε προηγουμένως. Στη λήψη απαντήσεων, παρατηρήθηκε πως χρειάζεται ένας πίνακας με πεδία όσα είναι όλοι οι εξυπηρετητές, τα οποία είναι αρχικοποιημένα με 0. Για κάθε απάντηση που παίρνει ο πελάτης, αλλάζει τη τιμή του πεδίου που αντιστοιχεί στο συγκεκριμένο εξυπηρετητή σε 1. Έτσι μπορεί εύκολα να ελέγχει αν απάντησε ένα ολόκληρο quorum στην αίτηση του. Σαφέστερα, ο πελάτης διασχίζει τη δομή του συστήματος απαρτίας και για κάθε

quorum ελέγχει αν οι εξυπηρετητές του έχουν απαντήσει. Μόλις βρει κάποιο που δεν απάντησε, προχωρεί στο επόμενο quorum. Αν διασχίσει όλο το quorum, τότε σημαίνει πως απάντησαν όλοι οι εξυπηρετητές και η αίτηση ολοκληρώθηκε. Με την ολοκλήρωση μιας λειτουργίας, ο πελάτης σταματά να δέχεται άλλες αιτήσεις, μηδενίζει τα πεδία του πίνακα και προχωρεί για την επόμενη λειτουργία.

Να αναφέρουμε ξανά πως στις εφαρμογές πελάτη έχει προστεθεί ο έλεγχος που διατηρεί την ατομικότητα του αλγορίθμου. Ο πελάτης δεν μπορεί να ξεκινήσει μια νέα λειτουργία εάν δεν έχει ολοκληρωθεί η προηγούμενη.

3.5.4 Εφαρμογή εγγραφέα (writer)

Ο εγγραφέας είναι η μόνη εφαρμογή που μπορεί να γράψει στα αντίγραφα του αντικειμένου. Χρειάζεται δύο γύρους επικοινωνίας, όπου στο πρώτο στέλνει μήνυμα READ για να ανακαλύψει το πιο πρόσφατο αντίγραφο και στο δεύτερο γύρο στέλνει μήνυμα WRITE με τη νέα τιμή που θέλει να γράψει. Αρχίζει τη λειτουργία του, τρέχοντας της εξής εντολή στη γραμμή εντολών του κελύφους:

```
./writer <writer Id> <servers file> <quorum file>
(<Operations number> <Operation Frequency>)
```

Το `writer` είναι το όνομα του εκτελέσιμου αρχείου και το πρώτο όρισμα είναι η ταυτότητα του εγγραφέα. Το `servers file` είναι το αρχείο με τις πληροφορίες για τους εξυπηρετητές και το `quorum file` είναι το αρχείο με το σύστημα απαρτίας. Τα επόμενα δύο ορίσματα είναι προαιρετικά και για σκοπούς αυτοματοποίησης. Αν θέλουμε ο εγγραφέας να εκτελέσει κάποιο συγκεκριμένο αριθμό λειτουργιών, τότε δίνουμε αυτό τον αριθμό στο όρισμα `Operations number` και στο επόμενο όρισμα δίνουμε τη συχνότητα με την οποία θα εκτελεί τις λειτουργίες.

Ο εγγραφέας φυλάει στη τοπική του μνήμη τη τιμή του αντικειμένου, το tag του που αποτελείται από τη ταυτότητα του και τη αντίστοιχη χρονοσφραγίδα, και το αριθμό αιτήσεων που έκανε.

3.5.5 Εφαρμογή αναγνώστη (reader)

Ο αναγνώστης δικαιούται μόνο να διαβάζει τα αντίγραφα του αντικειμένου, αλλά έχει τη δυνατότητα να ενημερώσει τους εξυπηρετητές με το πιο πρόσφατο αντίγραφο που βρήκε. Αρχικά να πούμε πως ξεκινά τη λειτουργία του, τρέχοντας της εξής εντολή στη γραμμή εντολών του κελύφους:

```
./reader <reader Id> <servers file> <quorum file>
(<Operations number> <Operation Frequency>)
```

Το `reader` είναι το όνομα του εκτελέσιμου αρχείου και το πρώτο όρισμα είναι η ταυτότητα του αναγνώστη. Όπως και στον εγγραφέα το `servers file` είναι το αρχείο με τις πληροφορίες για τους εξυπηρετητές και το `quorum file` είναι το αρχείο με το σύστημα απαρτίας. Τα επόμενα δύο ορίσματα είναι προαιρετικά με σκοπό να αποφευχθεί η αλληλεπίδραση του χρήστη με το πρόγραμμα. Αν θέλουμε ο αναγνώστης να εκτελέσει κάποιο συγκεκριμένο αριθμό λειτουργιών, τότε δίνουμε αυτό τον αριθμό στο όρισμα `Operations number` και στο επόμενο όρισμα δίνουμε τη συχνότητα με την οποία θα εκτελεί τις λειτουργίες.

Ομοίως με τον εγγραφέα, ο αναγνώστης φυλάει στη τοπική του μνήμη τη τιμή του αντικειμένου, το tag του που αποτελείται από τη ταυτότητα του τελευταίου εγγραφέα που έγραψε σε αυτό με τη αντίστοιχη χρονοσφραγίδα, και το αριθμό αιτήσεων που έκανε μέχρι στιγμής.

Να επαναλάβουμε πως στον αλγόριθμο SIMPLE, ο αναγνώστης εκτελεί πάντοτε δύο γύρους επικοινωνίας, όπου στο πρώτο στέλνει μήνυμα READ για να διαβάσει τη τιμή του αντικειμένου και στο δεύτερο γύρο στέλνει μήνυμα WRITE για να ενημερώσει όλους τους εξυπηρετητές με το πιο πρόσφατο αντίγραφο. Περισσότερο ενδιαφέρον παρουσιάζει ο αναγνώστης στον αλγόριθμο CWFR, όπου εισάγονται τα quorum views και βάση αυτών αποφασίζει αν θα προχωρήσει σε δεύτερο γύρο επικοινωνίας ή όχι. Στο Σχήμα 3.4 περιγράφεται σε μορφή ψευδοκώδικα η συνάρτηση που εντοπίζει ποιο quorum view αντιμετωπίζει ο αναγνώστης μετά την απάντηση ενός ολόκληρου quorum. Προτού αρχίσει όμως η διαδικασία, ο αναγνώστης βρίσκει το μέγιστο tag ανάμεσα στους εξυπηρετητές του quorum που απάντησε και το αποθηκεύει στο maxTag. Να

υπενθυμίσουμε πως η σύγκριση δύο tags γίνεται λεξικογραφικά, ελέγχοντας πρώτα τη χρονοσφραγίδα και ακολούθως τη ταυτότητα του εγγραφέα. Στο Σχήμα 3.5 παρουσιάζεται ο ψευδοκώδικας για τη σύγκριση δύο tags. Μετέπειτα, γίνεται ο πρώτος έλεγχος για το αν όλοι οι εξυπηρετητές του quorum έχουν το μέγιστο tag. Εάν ναι, τότε η συνάρτηση επιστρέφει με Quorum View 1. Κατά τη διάρκεια της διάσχισης του quorum, αν βρεθεί κάποιος εξυπηρετητής με μικρότερο tag, ο έλεγχος σταματά και συνεχίζει στο δεύτερο έλεγχο, όπου για κάθε τομή του quorum ελέγχει αν όλοι οι εξυπηρετητές της τομής έχουν το μέγιστο tag. Μόλις βρεθεί κάποια τομή με όλους τους εξυπηρετητές να κρατούν το μέγιστο, τερματίζει επιστρέφοντας Quorum View 3. Αν καμία από τις δύο περιπτώσεις δεν ισχύει, σημαίνει πως βρισκόμαστε στο Quorum View 2, το οποίο παραπέμπει τον αναγνώστη να κάνει επανάληψη των πιο πάνω βημάτων, αφού διαγράψει πρώτα τους εξυπηρετητές με το μέγιστο tag.

```
1  Για κάθε στοιχείο i του quorum
2  Begin
3        if (server[i].tag != maxTag)
4              break
5  End
6  // Αν έχει διαπεράσει όλη τη λίστα, τότε όλοι οι εξυπηρετητές
7  // έχουν το maxTag
8  if (i == NULL)
9        return Quorum View 1
10
11 Για κάθε τομή t του quorum με άλλο quorum
12 Begin
13       Για κάθε στοιχείο i της τομής
14       Begin
15             if (server[i].tag != maxTag)
16                   break
17       End
18       // Αν έχει διαπεράσει όλη τη λίστα, τότε όλοι οι εξυπηρετητές
19       // έχουν το maxTag
20       if (i == NULL)
21             return Quorum View 3
22 End
23
24 // Αν δεν είναι Quorum View 1 ή 3 τότε είναι Quorum View 2
25 Για κάθε στοιχείο i του quorum
26 Begin
27       if (server[i].tag == maxTag)
28             remove server[i]
29 End
30 Repeat from Line 1
```

Σχήμα 3.5: Ψευδοκώδικας για την εύρεση του Quorum View

```
1  if (tag1.ts > tag2.ts)
2        return 1                          // Το tag1 είναι μεγαλύτερο
3  if (tag1.ts == tag2.ts)
4        if (tag1.wid > tag2.wid)
5              return 1                    // Το tag1 είναι μεγαλύτερο
6        if (tag1.wid == tag2.wid)
7              return 0                    // Τα δύο tags είναι ίσα
8  return 2                                // Το tag2 είναι μεγαλύτερο
```

Σχήμα 3.6: Ψευδοκώδικας για τη σύγκριση δύο tags

3.6 Αρχεία εντολών

Με την ολοκλήρωση των δύο υλοποιήσεων, κατέστη αναγκαίο το γράψιμο κάποιων αρχείων εντολών (scripts) για τη διεξαγωγή των πειραμάτων στο δίκτυο PlanetLab. Για να εξεταστεί η απόδοση του αλγορίθμου, έχουν σχεδιαστεί διαφορετικά σενάρια που δοκιμάζουν τον αλγόριθμο ως προς διαφορετικές πτυχές. Αυτά θα μελετηθούν εκτενέστερα στο Κεφάλαιο 5, όπου γίνεται η πειραματική αξιολόγηση.

Τα αρχεία που έχουν γραφεί, περιέχουν εντολές των βιβλιοθηκών του κελύφους bash (**B**ourn **A**gain **Sh**ell) και βοηθούν στην αυτοματοποίηση κάποιων χρονοβόρων λειτουργιών, όπως η ταυτόχρονη έναρξη πολλαπλών διεργασιών, η μεταφορά των αρχείων σε όλες τις μηχανές και το «κατέβασμα» των log files από τις μηχανές στο τοπικό υπολογιστή. Επίσης, έχει γραφεί ένα αρχείο εντολών για τη συλλογή των αναγκαίων δεδομένων από τα logfiles που παράχθηκαν από τα πειράματα. Στα παραρτήματα Β και Γ δίνονται τα αρχεία αυτά.

3.7 Δυσκολίες και Παραδοχές

Οι δύο αλγόριθμοι υλοποιήθηκαν και ελέγχθηκαν για την ορθότητα τους σε τοπικό δίκτυο και μετέπειτα μεταφέρθηκαν στο δίκτυο PlanetLab. Με αυτό τον τρόπο πολλά λάθη εντοπίστηκαν όσο βρίσκονταν τοπικά, εξοικονομώντας έτσι σημαντικό χρόνο από το αν έτρεχαν απευθείας στο PlanetLab.

Παρόλα αυτά, στο τοπικό δίκτυο του Πανεπιστημίου Κύπρου δεν κατέστη δυνατός ο έλεγχος του μοντέλου πελάτη εξυπηρετητή, λόγω κάποιων κανονισμών του Πανεπιστημίου που απαγορεύουν την επικοινωνία μεταξύ των μηχανών. Έτσι, ο αλγόριθμος δεν ελέγχθηκε ως προς την ασυγχρονία και τις καθυστερήσεις παράδοσης μηνυμάτων. Συνεπώς, κάποια σφάλματα δεν είχαν φανερωθεί κατά τη διάρκεια χρήσης του τοπικού δικτύου.

Η αποσφαλμάτωση στο δίκτυο αν και ήταν καλύτερη, ήταν επίσης πολύ χρονοβόρα, αφού οι αλλαγές στα προγράμματα γίνονταν τοπικά και η αποστολή τους προς τους κόμβους απαιτούσε αρκετό χρόνο.

Κεφάλαιο 4

Πλατφόρμας Πειραμάτων

Το PlanetLab [17] είναι ένα παγκόσμιο ερευνητικό δίκτυο, ή διαφορετικά, μια γεωγραφικά κατανεμημένη υπερεπίπεδη πλατφόρμα που υποστηρίζει την ανάπτυξη νέων υπηρεσιών διαδικτύου. Μέλη αυτού του δικτύου είναι ακαδημαϊκά, βιομηχανικά και κυβερνητικά ιδρύματα τα οποία χρησιμοποιούν το PlanetLab για να αναπτύξουν νέες τεχνολογίες για κατανεμημένη αποθήκευση, χαρτογράφηση δικτύων, peer-to-peer ανταλλαγή αρχείων και επεξεργασία ερωτημάτων (query processing). Σήμερα το PlanetLab αποτελείται από 1064 κόμβους σε 562 τοποθεσίες.

4.1 Κίνητρα και Εξέλιξη του PlanetLab

Μέχρι τη δεκαετία του 1960, ο μόνος τρόπος επικοινωνίας ήταν το τηλεφωνικό δίκτυο. Έτσι στη δεκαετία αυτή ιδρύεται το διαδίκτυο το οποίο προσέφερε ευκολότερο τρόπο επικοινωνίας και τη δυνατότητα ανταλλαγής δεδομένων. Στη πάροδο των χρόνων, παρουσίασε ραγδαία πρόοδο και σήμερα έφτασε να καταμετρεί σχεδόν δύο δισεκατομμύρια χρήστες. Μια μεγάλη μερίδα χρηστών, η ερευνητική κοινότητα, ανέπτυξε πολύ γρήγορα υπηρεσίες και εφαρμογές παγκόσμιας κλίμακας, προς διευκόλυνση κάθε χρήστη του διαδικτύου. Παρόλα αυτά, το διαδίκτυο παρουσίασε κάποια προβλήματα, όπως η εύκολη διάδοση ιών, η ανεπιθύμητη πρόσβαση σε προσωπικά δεδομένα και η εμφάνιση του ηλεκτρονικού εγκλήματος. Επίσης, οι περιορισμοί που επιβάλλει η αρχιτεκτονική του, παρατηρήθηκε πως οδηγούν στην κατάρρευση εξυπηρετητών στις περιπτώσεις που μεγάλος αριθμός χρηστών προσπαθούσε ταυτόχρονα να αποκτήσει πρόσβαση σε αυτούς. Λόγω της τεράστιας

εμπορικής επιτυχίας του διαδικτύου, η αλλαγή του είναι αδύνατη, αφού περισσότερο οδηγείται από οικονομικά συμφέροντα παρά από την έρευνα. Η εισαγωγή νέων πρωτοκόλλων εκτιμάται πως θα επιφέρει μείωση στην απόδοση του δικτύου καθώς και άλλες δυσλειτουργίες στη λειτουργικότητα του. Επομένως, η δημιουργία ενός υπερεπίπεδου δικτύου πάνω από το διαδίκτυο με αρχιτεκτονική προσανατολισμένη στις υπηρεσίες, κρίνεται ως η καλύτερη λύση βελτίωση του διαδικτύου και αντιμετώπιση των προβλημάτων του.

Το PlanetLab αποτελεί το υπερεπίπεδο δίκτυο που μόλις αναφέραμε. Η αρχή έγινε το Μάρτιο του 2002 από τους Larry Peterson (Princeton) και David Culler (UC Berkeley and Intel Research), όπου μετά από μια σύσκεψη με άλλους 30 ερευνητές από τους MIT, Washington, Rice, Berkeley, Princeton, Columbia, Duke, CMU και Utah, πρότιναν το PlanetLab. Εντός των επόμενων εφτά μηνών, ολοκληρώθηκε η αρχική ανάπτυξη 100 κόμβων σε 42 τοποθεσίες (sites) και εκδόθηκε η πρώτη έκδοση 1.0 του λογισμικού του PlanetLab, με το κεφάλαιο που προσέφερε η David Tennenhouse (Intel Research). Το όραμα που τέθηκε από την αρχή είναι η εξάπλωση του δικτύου σε 1000 τοποθεσίες. Από το 2004, ακαδημαϊκά και βιομηχανικά ιδρύματα αρχίζουν να συμμετέχουν, με αποτέλεσμα την ακόμη μεγαλύτερη εξάπλωση του PlanetLab. Σήμερα το PlanetLab αποτελείται από 1064 κόμβους σε 562 τοποθεσίες, συνεχίζοντας να εξαπλώνεται καθημερινά.

4.2 Χαρακτηριστικά του PlanetLab

Ένας από τους κύριους στόχους του PlanetLab είναι να λειτουργεί ως μια πλατφόρμα ανάπτυξης και δοκιμής νέων εφαρμογών και υπηρεσιών για overlay δίκτυα. Αυτή η πλατφόρμα ορίζεται μέσα από τρεις διαστάσεις:

1. Τις φυσικές διαστάσεις του δικτύου που πρέπει να είναι μεγάλες, για να είναι εφικτή τόσο η ανάπτυξη μικρού εύρους εφαρμογών όσο και παγκοσμίου κλίμακας εφαρμογές.
2. Το υπερεπίπεδο δίκτυο πρέπει να περιλαμβάνει δύο κύρια συστατικά λογισμικού, που είναι ένα virtual machine monitor (VMM) το οποίο πρέπει να τρέχει σε κάθε κόμβο, και μια υπηρεσία διαχείρισης του δικτύου.
3. Ο τρόπος λειτουργίας του δικτύου. Το δίκτυο μπορεί να χρησιμοποιείται ως μια πλατφόρμα δοκιμών για έρευνα και ως μια πλατφόρμα ανάπτυξης εφαρμογών.

Ως πλατφόρμα δοκιμών, το PlanetLab προσφέρει ένα μεγάλο σύνολο κατανεμημένων μηχανών, ένα ρεαλιστικό δίκτυο με την παρουσία συμφόρησης, καταρρεύσεων και ποικίλων συμπεριφορών από τους κόμβους. Προσφέρει επίσης τη προοπτκή ενός ρεαλιστικού φόρτου εργασίας πελάτη. Ως πλατφόρμα ανάπτυξης, προσφέρει καινοτόμες υπηρεσίες και χρήστες που έχουν πρόσβαση σε αυτές τις υπηρεσίες.

4.3 Λειτουργία του PlanetLab

Κάθε ίδρυμα που συμμετέχει στο PlanetLab, προσφέρει δύο ή περισσότερους κόμβους για να χρησιμοποιούνται από τα άλλα μέλη του δικτύου. Όλοι αυτοί οι κόμβοι σχηματίζουν το overlay δίκτυο. Για να γίνει κάποιος μέλος, πρέπει να κάνει αίτηση και να εγκριθεί από κάποιο PI (Principal Investigator) του ιδρύματος στο οποίο ανήκει.

Όλοι οι πόροι του δικτύου, δηλαδή οι πόροι κάθε κόμβου, μοιράζονται σε μερίσματα (slices). Ένας PI μπορεί να δημιουργήσει ένα μέρισμα, το οποίο παρουσιάζεται σε αυτόν, αλλά και στους χρήστες-μέλη του συγκεκριμένου μερίσματος, ως ένα δίκτυο με εικονικές μηχανές τους κόμβους που έχει επιλέξει. Κάθε ίδρυμα μπορεί να δημιουργήσει το πολύ 10 μερίσματα και κάθε μέρισμα μπορεί να περιλαμβάνει μέχρι και 32 κόμβους. Από τη στιγμή που δημιουργείται το μέρισμα έχει διάρκεια ζωής ένα μήνα και αν δεν ανανεωθεί, λήγει αποδεσμεύοντας τους πόρους του. Να προσθέσουμε πως η απόδοση κάθε κόμβου αλλά και του δικτύου εξαρτάται από όλους τους χρήστες, αλλά κάθε χρήστης χρησιμοποιεί τους κόμβους χωρίς να τον «βλέπουν» οι υπόλοιποι.
Επίσης, η απόδοση του δικτύου επιρεάζεται κάποιες φορές από κακόβουλες υπηρεσίες και λογισμικά που περιέχουν λάθη. Τέλος, να πούμε πως οι κόμβοι μπορούν ανά πάσα στιγμή να επανεκκινήσουν, χωρίς όμως να χάνονται τα δεδομένα. Με αυτό τον τρόπο γίνονται μη-αξιόπιστοι, όπως συμβαίνει και στο διαδίκτυο.

Η πρόσβαση στο δίκτυο γίνεται μέσω SSH, το οποίο προσφέρει ασφαλή και κρυπτογραφημένη επικοινωνία με τους κόμβους. Με το που εγγράφεται κάποιος ως χρήστης, χρειάζεται να δημιουργήσει ένα ζεύγος κρυπτογραφημένων κλειδιών SSH, ένα private key το οποίο πρέπει να προστατεύεται και ένα public key το οποίο πρέπει να

κάνει γνωστό στο μέρισμα του. Η δημιουργία των δύο κλειδιών γίνεται με τη πιο κάτω εντολή:

```
ssh-keygen -t rsa -f ~/.ssh/id_rsa
```

Με την επιλογή `-t` ορίζουμε το τύπο του κλειδιού και με την επιλογή `-f` προσδιορίζουμε το αρχείο που θα φυλαχτεί το ζεύγος.

Εφόσον έχει γνωστοποιηθεί το public key του χρήστη, μπορεί πλέον να ενωθεί σε κάποια μηχανή με την εξής εντολή:

```
ssh -l slice -i ~/.ssh/id_rsa hostname
```

Το `slice` είναι το όνομα του μερίσματος και το `hostname` είναι το όνομα της μηχανής στην οποία θέλει να ενωθεί. Με την επιλογή `-l` προσθέτουμε το όνομα του χρήστη, όπου στη προκειμένη περίπτωση είναι το όνομα του μερίσματος. Με την επιλογή προσδιορίζουμε το αρχείου όπου βρίσκεται το private key. Με την εκτέλεση της εντολής, ο χρήστης θα ζητηθεί να δώσει το κωδικό για να πιστοποιηθεί η ταυτότητα του.

Για τη μεταφορά αρχείων από τον προσωπικό χώρο του χρήστη σε κάποιο κόμβο χρησιμοποιείται η εξής εντολή:

```
rsync -avz -i ~/.ssh/id_rsa filename/ slice@hostname:dir/
```

Το `filename/` προσδιορίζει τη διεύθυνση του φακέλου με τα αρχεία που θα μεταφερθούν. Με το χαρακτήρα / ορίζουμε να μεταφερθούν μόνο τα περιεχόμενα και όχι ο φάκελος. Το `slice` είναι το όνομα του μερίσματος και το `hostname` είναι το όνομα της μηχανής που θα μεταφερθούν τα αρχεία. Ακολουθεί το `dir/` που είναι η διεύθυνση στη μηχανή εκείνη που θέλει ο χρήστης να αποθηκευτούν τα αρχεία. Με τις επιλογές `-avz` τα αρχεία μεταφέρονται σε “archive” mode, δηλαδή διασφαλίζεται πως θα μεταφερθούν τα δικαιώματα των αρχείων, τα ownerships κ.α. ως έχουν. Για τη μεταφορά αρχείων από κάποιο κόμβο στο προσωπικό χώρο του χρήστη, το `filename/` μεταφέρεται στο τέλος της γραμμής εντολής.

Λόγω της διαχείρισης πολλών μηχανών ταυτόχρονα, έχει χρησιμοποιηθεί η pssh η οποία είναι η παράλληλη έκδοση των openssh εργαλείων. Αναλυτικότερα, έχουν χρησιμοποιηθεί οι εντολές: parallel-ssh και parallel-nuke, οι οποίες εξηγούνται πιο κάτω.

```
parallel-ssh –t -1 –h hosts.txt –l slice command
parallel-nuke –t -1 –h hosts.txt –l slice executables
```

Η `parallel-ssh` εκτελεί εντολές σε πολλαπλές μηχανές, των οποίων τα ονόματα βρίσκονται αποθηκευμένα στο `hosts.txt`. Ομοίως με τις προηγούμενες εντολές που είδαμε, δίνεται το όνομα του μερίσματος στο `slice`, και ακολουθεί η εντολή που θα εκτελεστεί. Η `command` μπορεί να είναι μια απλή εντολή του bash ή ακόμη και ένα πρόγραμμα. Στη περίπτωση μας, είναι ένα script που βρίσκεται σε κάθε μηχανή το οποίο εκκινεί τη λειτουργία των διεργασιών. Η επιλογή `–t –1` προσδιορίζει πως δεν θα υπάρχει χρόνος λήξης (timeout) για κάθε κόμβο.

Η `parallel-nuke` «σκοτώνει» τις διεργασίες που δηλώνονται στο `executables`, οι οποίες εκτελούνται στις μηχανές που είναι αποθηκευμένες στο `hosts.txt`. Στα πειράματα, χρησιμοποιείται κυρίως για το τερματισμό της λειτουργίας των εξυπηρετητών.

Στη παράλληλη εκτέλεση εντολών, η εισαγωγή του κωδικού είναι αναγκαία τόσες φορές όσες είναι οι μηχανές στο αρχείο `hosts.txt`. Για το λόγο αυτό, χρησιμοποιούνται οι δύο πιο κάτω εντολές για την αυτοματοποίηση της διαδικασίας και την εισαγωγή μόνο μιας φοράς του κωδικού.

```
eval $(ssh-agent)
ssh-add
```

Η πρώτη εντολή δημιουργεί ένα πράκτορα (agent), ο οποίος θα χειρίζεται το κλειδί μας. Με τη δεύτερη εντολή, προστίθεται ο κωδικός στον πράκτορα για να χρησιμοποιείται για οποιαδήποτε πιστοποίηση χρειάζεται. Στο τέλος κάθε λειτουργίας είναι ασφαλέστερο να τερματίζεται ο πράκτορας για να κλείνει κάθε πρόσβαση στο κλειδί.
Ο πράκτορας τερματίζεται με την εντολή `ssh-agent –k`.

Κεφάλαιο 5

Πειραματική Αξιολόγηση

Σε αυτό το σημείο της διπλωματικής περιγράφονται τα σενάρια τα οποία δοκιμάστηκαν στα πειράματα που χρησιμοποιήθηκαν για σύγκριση των δύο αλγορίθμων.

5.1 Προετοιμασία

Κατ' αρχήν για να ελεγχθεί η ορθότητα του αλγορίθμου δοκιμάσαμε τους δύο αλγορίθμους σε ένα μικρό σύστημα απαρτίας των 8 εξυπηρετητών. Αφού η επικοινωνία των εξυπηρετητών και πελατών διεξήχθηκε ορθά προχωρήσαμε στην διεξαγωγή των πειραμάτων. Για την συλλογή των δεδομένων εξόδου προστέθηκε και στις δύο εφαρμογές πελάτη χρονόμετρο για την μέτρηση του χρόνου ολοκλήρωσης της κάθε λειτουργίας (latency). Πριν το τερματισμό των εφαρμογών, υπολογίζεται πλεόν ο μέσος χρόνος ολοκλήρωσης των λειτουργιών. Επιπλέον, στην εφαρμογή του αναγνώστη προστέθηκαν δυο μετρητές που απαριθμούν τις αργές (2 RTT) και τις γρήγορες λειτουργίες (1 RTT). Τα δεδομένα αυτά φυλάγονται σε αρχείο εξόδου με τη χρήση της εντολής fprintf(). Στο τέλος κάθε σεναρίου, τα αρχεία με τα δεδομένα μεταφέρονται στο προσωπικό υπολογιστή για περαιτέρω επεξεργασία.

Σε όλα τα σενάρια, οι διεργασίες του εγγραφέα και του αναγνώστη εκτελούν 200 αιτήσεις εγγραφής και ανάγνωσης αντίστοιχα. Κάθε σενάριο εκτελέστηκε πέντε φορές για να μπορούμε να διαγράψουμε τις ακραίες τιμές, δηλαδή τα σενάρια με το μεγαλύτερο και μικρότερο χρόνο, και να υπολογίσουμε το μέσο όρο των υπολοίπων τριών σεναρίων. Κάθε εκτέλεση χρειάστηκε περίπου 30 λεπτά για να ολοκληρωθεί.

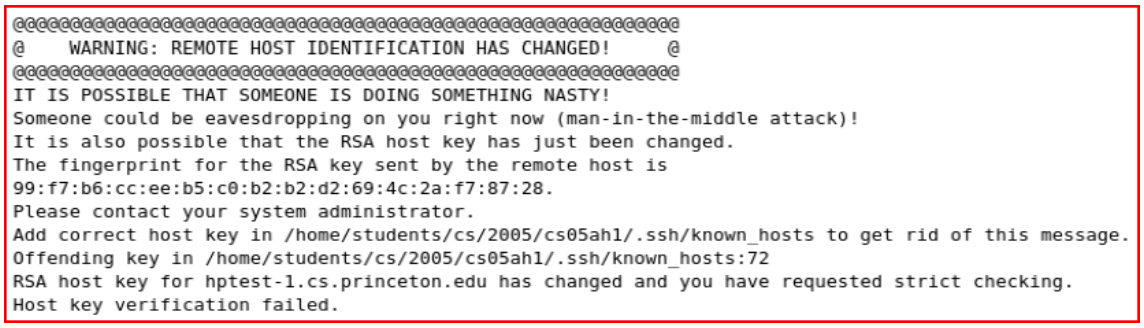
5.2 Προβλήματα και Παραδοχές

Κατά τη διάρκεια εκτέλεσης των πειραμάτων παρουσιάστηκαν αρκετά προβλήματα και δυσκολίες. Το δίκτυο, ως ρεαλιστικό και ασύγχρονο, επέτρεπε την καθυστέρηση παράδοσης των μηνυμάτων. Η καθυστέρηση διέφερε από στιγμή σε στιγμή και από κόμβο σε κόμβο. Η συμφόρηση του δικτύου συνέτεινε στη παρουσία μεγαλύτερης καθυστέρησης. Ακόμη, κάθε κόμβος, ανάλογα με τη τοποθεσία και τα χαρακτηριστικά του, είχε το δικό του ρυθμό λειτουργίας. Έτσι, κάποιοι κόμβοι ήταν πολύ πιο αργοί από άλλους, καθυστερώντας σημαντικά την εκτέλεση των πειραμάτων.

Όπως έχει προαναφερθεί, οι κόμβοι μπορούν ανά πάσα στιγμή να καταρρεύσουν και να μην είναι διαθέσιμοι ή ακόμη και να μην λειτουργούν ορθά. Παρατηρήθηκε πως κάποιοι κόμβοι χρειάζονταν μέρες για να γίνουν ξανά διαθέσιμοι, καθυστερώντας την αποπεράτωση των πειραμάτων. Επίσης, η κατάρρευση των κόμβων του Πανεπιστημίου για συγκεκριμένο χρονικό διάστημα, οδήγησε στην απαγόρευση χρήσης του PlanetLab με βάση το κανονισμό που λεει πως κάθε μέλος πρέπει να προσφέρει δύο κόμβους στο δίκτυο για να του παρέχεται πρόσβαση σε αυτό. Αυτό είχε ως αποτέλεσμα την ακόμη μεγαλύτερη καθυστέρηση στην ολοκλήρωση των πειραμάτων.

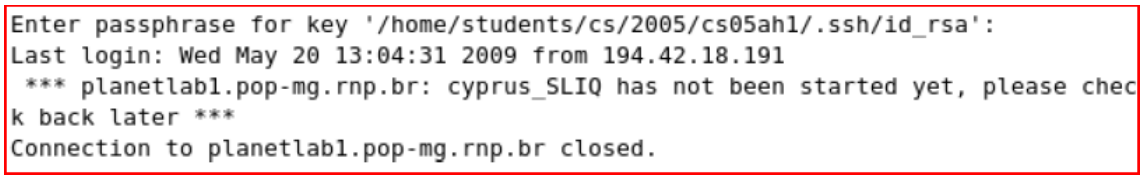
Επιπρόσθετα, κάποιοι κόμβοι που εκτελούσαν τη διεργασία του εξυπηρετητή λάμβαναν μηνύματα από το Bit Torrent [21], κάτι που έχει αναφερθεί ως πρόβλημα που αντιμετωπίζει το PlanetLab. Η επικοινωνία με τις μηχανές του Bit Torrent οδηγούσε στην πλήρη απασχόληση των εξυπηρετητών και στη πρόκληση προβλημάτων στη δική μας αξιολόγηση. Επιπλέον, πολλοί κόμβοι παρουσίαζαν παράξενη συμπεριφορά κατά την εκτέλεση των αλγορίθμων, προκαλώντας πολλές φορές σφάλμα καταμερισμού σε άλλους κόμβους που συνδέονταν σε αυτούς.

Μια άλλη μερίδα κόμβων δεν ήταν προσβάσιμη για διάφορους άλλους λόγους, εκτός από τους προαναφερόμενους. Να διευκρινήσουμε πως τη πρώτη φορά που ο χρήστης ενώνεται σε κάποια μηχανή, το αναγνωριστικό της μηχανής εκείνης φυλάγεται στο αρχείο known_hosts. Ανά διαστήματα αυτό το αναγνωριστικό αλλάζει για σκοπούς ασφαλείας, με αποτέλεσμα όταν συμβεί αυτό, να χάνεται η πρόσβαση στη μηχανή. Το μήνυμα που φαίνεται στο Σχήμα 5.1 παρουσιάζεται μετά από μια τέτοια ενέργεια.



Σχήμα 5.1: Μήνυμα προειδοποίησης στην αλλαγή αναγνωριστικού κάποιου σταθμού στο PlanetLab [20].

Για να ξεπεραστεί αυτό το πρόβλημα, χρειαζόταν να διαγραφεί το αναγνωριστικό της μηχανής από το known_hosts, ούτως ώστε στην επόμενη προσπάθεια σύνδεσης, να προστεθεί το νέο αναγνωριστιό στο αρχείο. Ακόμη, μετά την επανεκκίνηση ενός κόμβου, στη προσπάθεια σύνδεσης με αυτόν εμφανιζόταν το μήνυμα του Σχήματος 5.2. Για σύντομο χρονικό διάστημα η σύνδεση δεν ήταν εφικτή, μέχρι να ξεπεραστεί το πρόβλημα.



Σχήμα 5.2: Μήνυμα προειδοποίησης στην επανεκκίνηση ενός κόμβου [20].

Τέλος, να αναφέρουμε πως στη παρουσία μεγάλου αριθμού διεργασιών, παρατηρούνταν προβλήματα. Οι εξυπηρετητές σταματούσαν να απαντούν σε μερίδα πελατών ή ακόμη και σε όλες. Πιθανώς αυτό να συνέβαινε λόγω της ανάγκης μεγαλύτερου ποσοστού μνήμης των μηχανών, το οποίο δεν ήταν διαθέσιμο εκίνη τη στιγμή. Ακόμη, παρατηρήθηκε οι διεργασίες από το ένα είδος πελάτη να εμφανίζουν μήνυμα λάθους *"recv failed: Connection reset by peer"*. Το πρόβλημα αυτό αναφέρθηκε από άλλους χρήστες, χωρίς όμως να δοθεί τεκμηριωμένη εξήγηση που το προκαλεί ή απάντηση που να το επιλύει.

5.3 Σενάρια

Στα πειράματα χρησιμοποιήθηκε το σύστημα απαρτίας «crumbling walls» [13] με 25 εξυπηρετητές. Ο αριθμός αυτός επιλέκτηκε βάση των πειραμάτων που εκτελέστηκαν για τους προηγούμενους αλγορίθμους που αναφέρονται στο υποκεφάλαιο με τις προηγούμενες εργασίες. Είναι σημαντικό να χρησιμοποιούμε τον ίδιο αριθμό, ούτως ώστε να γίνεται η αξιολόγηση βάση των παλαιότερων πειραμάτων. Επίσης, οι αλγόριθμοι δοκιμάστηκαν με 10, 20 και 40 εγγραφείς, καθώς επίσης με 10, 20 και 40 αναγνώστες. Η επιλογή των αριθμών αυτών δεν είναι τυχαία. Αρχικά δοκιμάζουμε τους αλγορίθμους με μικρό αριθμό διεργασιών ούτως ώστε να ελεγχθεί ότι οι αλγόριθμοι εκτελούνται σωστά στο δίκτυο και σταδιακά τους αυξάνουμε για να παρατηρούμε τις αλλαγές στη συμπεριφορά του συστήματος. Να προσθέσουμε πως οι ίδιοι αριθμοί έχουν χρησιμοποιηθεί και σε προηγούμενα πειράματα και βάση αυτών κτίσαμε τα δικά μας.

Στα πειράματα δεν προσομοιώσαμε εσκεμμένα σφάλματα, αλλά πρέπει να ληφθεί υπόψη πως το δίκτυο επιτρέπει καταρρεύσεις. Έτσι, αναμένεται να υπάρχουν κάποιες διαφορές στα αποτελέσματα εξαιτίας των καταρρεύσεων. Ο χρόνος μεταξύ δύο λειτουργιών και για τις δύο διεργασίες πελάτη είναι 4.3 δευτερόλεπτα. Με αυτό το σενάριο, εξετάζεται η απόδοση των δύο αλγορίθμων κάτω από παρόμοιες συνθήκες, δηλαδή με ίδιο σύστημα απαρτίας και με ίδιο πλήθος εξυπηρετητών, αλλά με διαφορετικό αριθμό εγγραφέων και αναγνωστών.

Αναμένεται πως ο χρόνος ολοκλήρωσης μιας λειτουργίας ανάγνωσης ή γραφής θα αυξάνεται όσο αυξάνονται οι αναγνώστες και οι εγγραφείς. Αυτό δικαιολογείται αφού με περισσότερους πελάτες, αυξάνεται ο αριθμός των μηνυμάτων που στέλονται εντός του δικτύου με αποτέλεσμα να δημουργείται μεγαλύτερη συμφόρηση και τα μηνύματα να καθυστερούν να φτάσουν στο προορισμό τους.

Επίσης, αναμένεται πως στον αλγόριθμο CWFR, οι λειτουργίες ανάγνωσης θα ολοκληρώνονται γρηγορότερα από τις λειτουργίες γραφής, λόγω του ότι οι λειτουργίες γραφής χρειάζονται πάντα δύο γύρους επικοινωνίας, σε αντίθεση με τις λειτουργίες ανάγνωσης που αναλόγως των quorum views μπορούν να ολοκληρωθούν σε ένα γύρο. Ομοίως, μπορούμε να πούμε πως οι λειτουργίες ανάγνωσης για τον αλγόριθμο CWFR

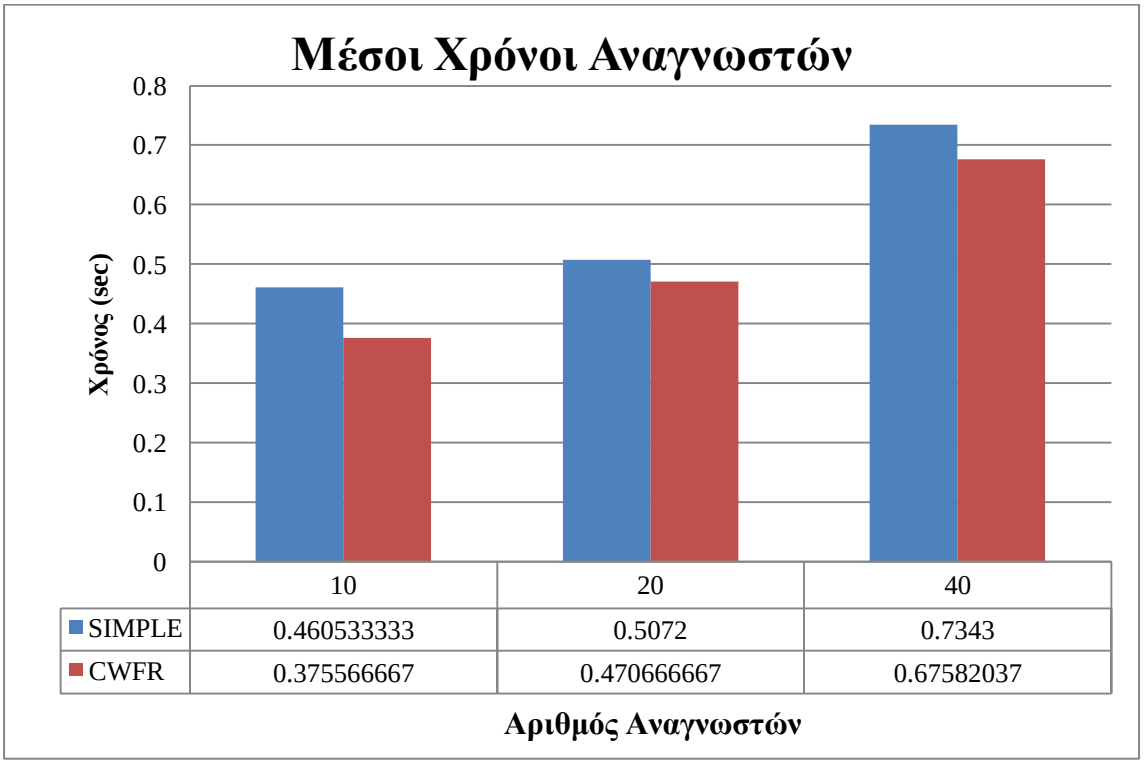
θα ολοκληρώνονται γρηγορότερα από τις λειτουργίες ανάγνωσης για τον αλγόριθμο SIMPLE, στον οποίο χρειάζονται πάντα δύο γύρους επικοινωνίας.

Στα γραφήματα 5.1 εώς 5.6 διαπιστώνονται οι πιο πάνω αναφορές. Συγκεκριμένα, στα γραφήματα 5.1 και 5.2 παρουσιάζεται ο μέσος χρόνος ολοκλήρωσης μιας λειτουργίας ανάγνωσης ως προς το πλήθος των αναγνωστών, για 10 και 20 εγγραφείς αντίστοιχα. Στα γραφήματα 5.3 και 5.4 παρουσιάζεται ο μέσος χρόνος ολοκλήρωσης μιας λειτουργίας ανάγνωσης και γραφής με 40 εγγραφείς και 10 αναγνώστες. Η εκτέλεση των πειραμάτων με 20 και 40 αναγνώστες απέτυχε, εμφανίζοντας σε όλες τις διεργασίες εγγραφέων το πρόβλημα *"recv failed: Connection reset by peer"* όπως αναφέρθηκε στο προηγούμενο υποκεφάλαιο. Στα γραφήματα 5.5 και 5.6 παρουσιάζεται ο μέσος χρόνος ολοκλήρωσης μιας λειτουργίας γραφής ως προς το πλήθος των εγγραφέων, για 10 και 20 αναγνώστες αντίστοιχα.

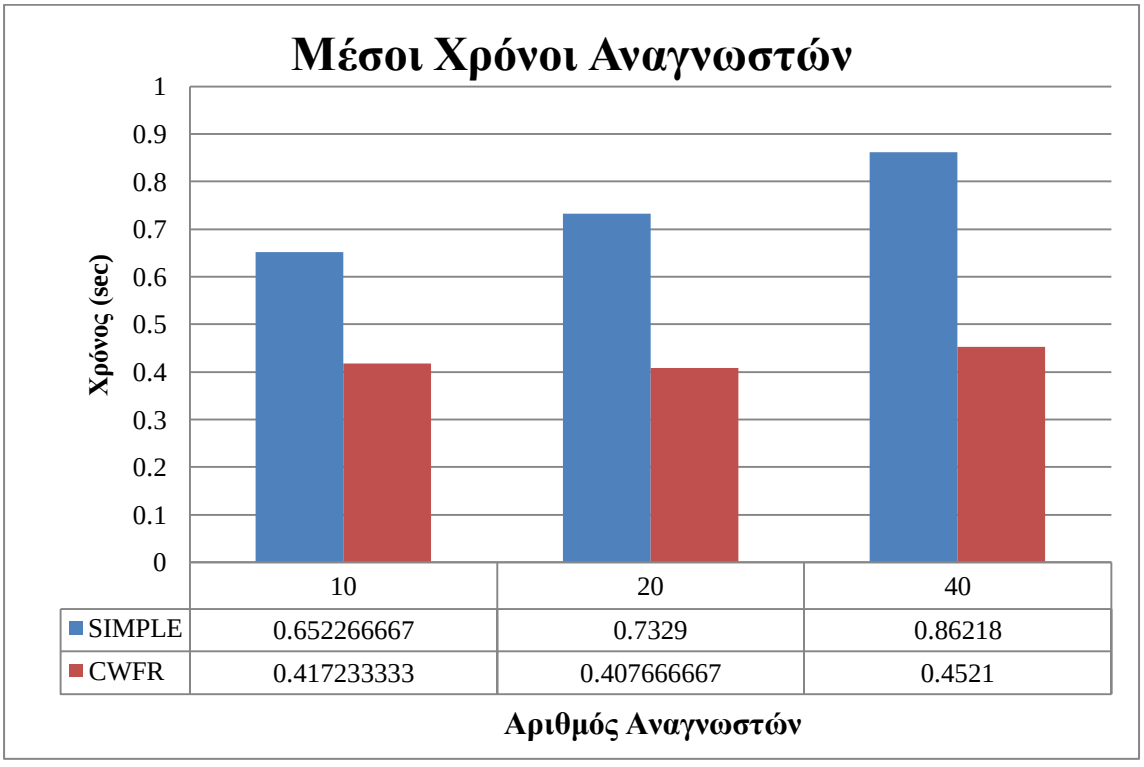
Πράγματι, όπως φαίνεται πιο κάτω, ο χρόνος ολοκλήρωσης μιας λειτουργίας γραφής ή ανάγνωσης αυξάνεται όσο αυξάνεται ο αριθμός αναγνωστών και εγγραφέων. Αυτό που δεν γνωρίζαμε όμως είναι ο ρυθμός με τον οποίο αυξάνεται, και τώρα βλέπουμε πως ο ρυθμός μεγαλώνει στην αύξηση των αναγνωστών. Επίσης, παρατηρείται πως σε περισσότερες γραφές, ο ρυθμός αύξησης του χρόνου για τους αναγνώστες είναι πολύ μικρότερος απ'ότι σε λιγότερες γραφές. Αυτό ίσως να οφείλεται στη συμφόρηση του δικτύου τη συγκεκριμένη χρονική στιμγή. Ακόμη, βλέπουμε πως ο αλγόριθμος CWFR ολοκληρώνει τις λειτουργίες ανάγνωσης γρηγορότερα από τον SIMPLE, κάτι που ήταν αναμενόμενο χάρη στις όψεις των συστημάτων απαρτιάς.

Όπως γνωρίζουμε, οι εγγραφείς εκτελούν δύο γύρους επικοινωνίας και στους δύο αλγορίθμους. Παρόλα αυτά, βλέπουμε πως ο ρυθμός αύξησης του χρόνου των εγγραφέων κατά την αύξηση τους είναι πολύ μεγάλος στους 10 αναγνώστες, με τον SIMPLE να ξεπερνά κατά πολύ τον CWFR. Αντίθετα, στους 20 αναγνώστες, ο ρυθμός είναι πολύ μεγαλύτερος στον CWFR και ομοίως η αύξηση του χρόνου είναι μεγαλύτερη από ότι στον SIMPLE. Πιθανή αιτία ήταν η κατάρρευση αρκετών εξυπηρετητών με αποτέλεσμα να υπάρχουν λιγότερα σύνολα quorums στο σύστημα για να απαντούν στους πελάτες. Για αυτό, ελέγχθηκαν τα logfiles όπου παρατηρήθηκε πως λόγω της κατάρρευσης 2 εξυπηρετητών, οι εγγραφείς έπαιρναν απαντήσεις από άλλα

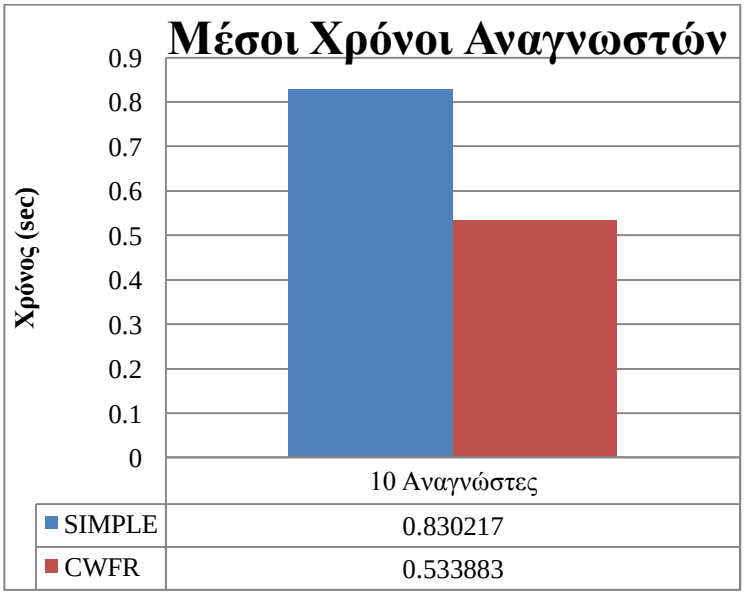
quorums που καθυστερούσαν περισσότερο να απαντήσουν. Έτσι, χρειάζονταν περισσότερο χρόνο για να ολοκληρώσουν τις λειτουργίες τους.



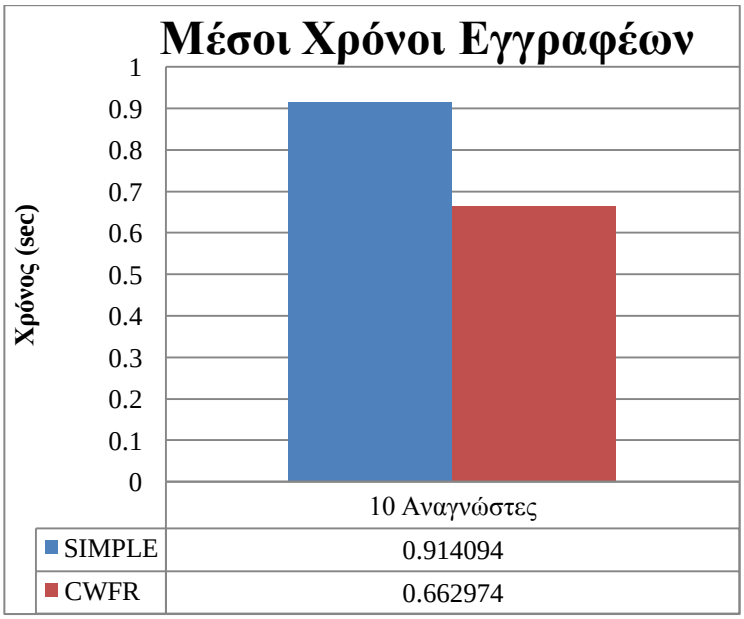
Γράφημα 5.1: Μέσος χρόνος των λειτουργιών ανάγνωσης όσο αυξάνεται ο αριθμός αναγνωστών στο σύστημα απαρτίας Crumbling Walls, με 10 εγγραφείς.



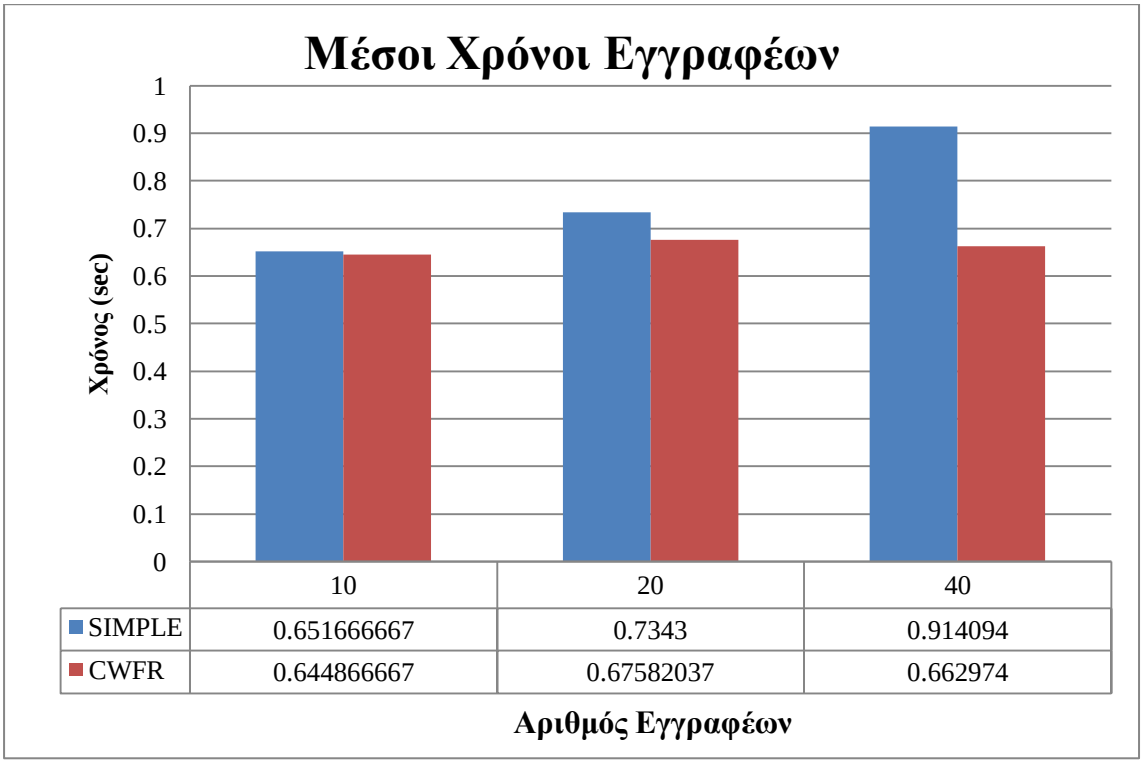
Γράφημα 5.2: Μέσος χρόνος των λειτουργιών ανάγνωσης όσο αυξάνεται ο αριθμός αναγνωστών στο σύστημα απαρτίας Crumbling Walls, με 20 εγγραφείς.



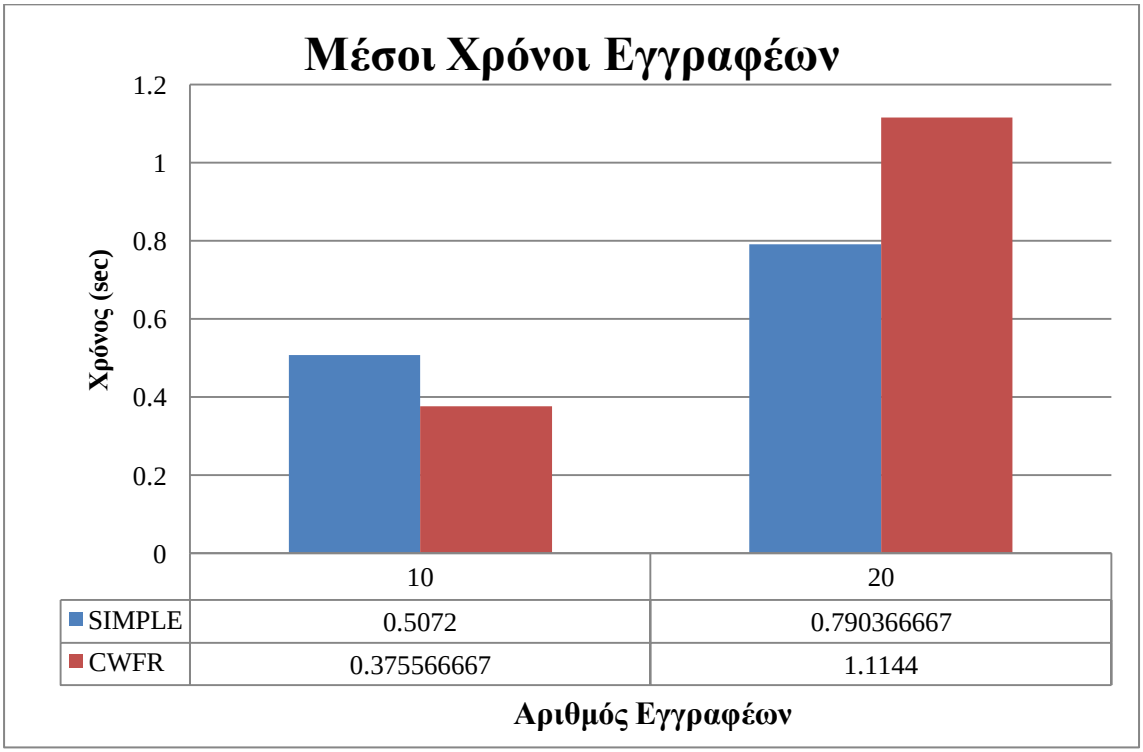
Γράφημα 5.3: Μέσος χρόνος των λειτουργιών ανάγνωσης στο σύστημα απαρτίας Crumbling Walls, με 40 εγγραφείς και 10 αναγνώστες.



Γράφημα 5.4: Μέσος χρόνος των λειτουργιών γραφής στο σύστημα απαρτίας Crumbling Walls, με 40 εγγραφείς και 10 αναγνώστες.

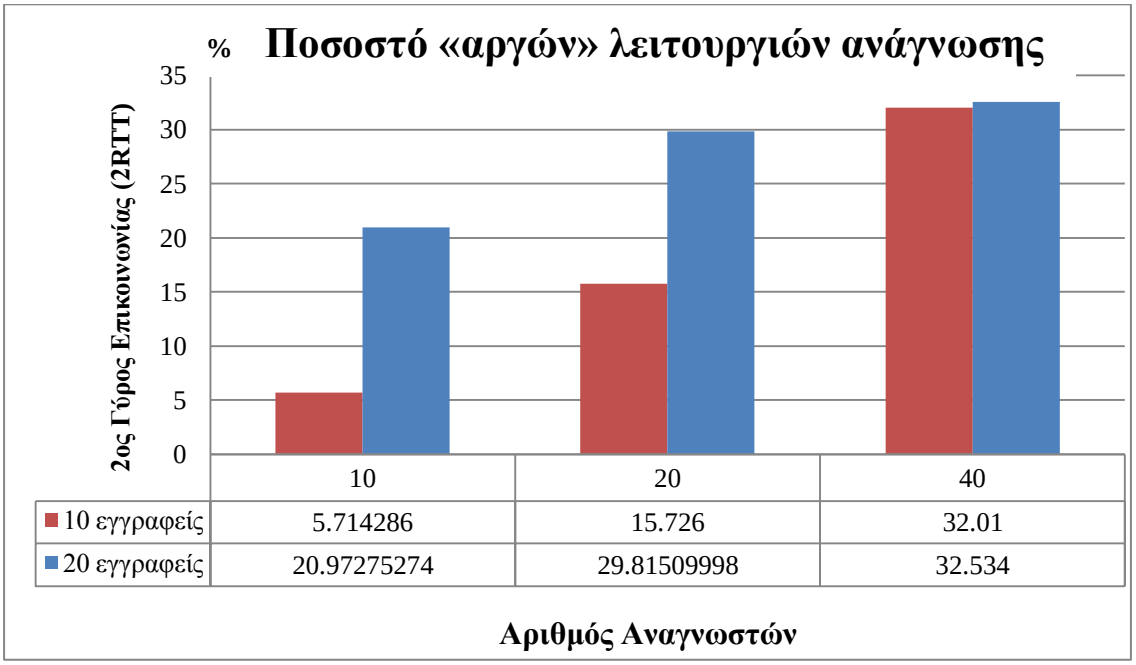


Γράφημα 5.5: Μέσος χρόνος των λειτουργιών γραφής όσο αυξάνεται ο αριθμός εγγραφέων στο σύστημα απαρτίας Crumbling Walls, με 10 αναγνώστες.



Γράφημα 5.6: Μέσος χρόνος των λειτουργιών γραφής όσο αυξάνεται ο αριθμός εγγραφέων στο σύστημα απαρτίας Crumbling Walls, με 20 αναγνώστες.

Στο γράφημα 5.7 παρουσιάζεται το ποσοστό των λειτουργιών ανάγνωσης στο CWFR που χρειάστηκαν 2 γύρους επικοινωνίας για να ολοκληρωθούν. Συγκεκριμένα, βλέπουμε τη διαφορά στα ποσοστά στις περιπτώσεις των 10, 20 και 40 εγγραφέων, όσο αυξάνονται οι αναγνώστες. Παρατηρείται πως στους 20 εγγραφείς, το ποσοστό ξεκινά με μεγαλύτερη τιμή, γεγονός που είναι αναμενόμενο αφού με περισσότερες γραφές υπάρχει μεγαλύτερο ενδεχόμενο να συμβούν κάποιες λειτουργίες γραφής ταυτόχρονα με τις λειτουργίες ανάγνωσης. Και τα δύο ποσοστά αυξάνονται με την αύξηση των αναγνωστών. Αυτό δικαιολογείται πλήρως, αφού όσο περισσότερους αναγνώστες έχουμε, τόσο μεγαλύτερη είναι η πιθανότητα να συμπέσει κάποια λειτουργία ανάγνωσης με κάποια λειτουργία γραφής και να χρειαστεί δεύτερο γύρο επικοινωνίας. Αξιοσημείωτο είναι το γεγονός πως το ποσοστό μεγαλώνει σημαντικά σε όλες τις περιπτώσεις, εκτός στη περίπτωση των 20 εγγραφέων, όταν αυξάνονται οι αναγνώστες από 20 σε 40. Δηλαδή, βλέπουμε στο τέλος το ποσοστό να μην ξεπερνά το 33%, παρόλο που αν συνέχιζε με τον ίδιο ρυθμό αύξησης, θα «έπρεπε» να ξεπεραστεί. Αυτό όμως είναι αναμενόμενο, αφού έχουμε ένα συγκεκριμένο πλήθος εξυπηρετητών, και ως εκ τούτου μπορεί να υπάρχει ένας συγκεκριμένος αριθμός διαφορετικών τιμών του αντικειμένου, μέσα σε ένα quorum. Έτσι οι επαναλήψεις που μπορεί να κάνει ο αναγνώστης για να καταλήξει σε ένα συγκεκριμένο quorum view δεν είναι δυνατόν να ξεπερνούν ένα όριο.



Γράφημα 5.7: Ποσοστό αργών λειτουργιών ανάγνωσης ανάλογο του αριθμού αναγνωστών στο σύστημα απαρτίας Crumbling Walls, με 10 και 20 εγγραφείς.

Τέλος, το ποσοστό αργών λειτουργιών ανάγνωσης για 40 εγγραφείς και 10 αναγνώστες ανέρχεται στο μεγάλο ποσοστό του 47%. Το συγκεκριμένο σενάριο εκτελέστηκε 3 φορές, με ποσοστά 50.5488, 44.8888 και 46. Δεν αναμενόταν αυτό το αποτέλεσμα και χρειάζεται περεταίρω μελέτη, με επανάληψη αυτού του σεναρίου, και με δοκιμή περισσότερων αναγνωστών. Με τα πειράματα αποδείκτηκε πως ο αλγόριθμος CWFR είναι όντως αποδοτικότερος. Αυτό το ποσοστό αποτελεί αντίφαση και υπάρχουν πολλοί πιθανοί λόγοι που ίσως να το προκαλούν όπως η μεγάλη συμφόρηση του δικτύου, η ταυτόχρονη εκτέλεση των λειτουργιών που οδηγεί στο δεύτερο γύρο επικοινωνίας.

5.4 Γενικές Παρατηρήσεις

Κατά τη διάρκεια της πειραματικής αξιολόγησης, παρατηρήθηκαν τα χαρακτηριστικά του δικτύου PlanetLab που είναι η ασυγχρονία, η κατάρρευση κόμβων, η μη ανταπόκριση τους και η συμφόρηση στο δίκτυο που ήταν τις περισσότερες φορές διαφορετική. Το τελευταίο χαρακτηριστικό είχε επηρεάσει τις μετρήσεις δίνοντας μη αναμενόμενα αποτελέσματα.

Με μεγάλους αριθμούς αναγνωστών και εγγραφέων και ειδικά στο σενάριο με τους 80 αναγνώστες ή/και εγγραφείς τα πειράματα «αποτύγχαναν», με πολλές διεργασίες να καταρρέουν. Παρατηρήθηκε ειδικά οι εξυπηρετητές να καταρρέουν κατά τη διάρκεια σύνδεσης με όλους τους πελάτες, με αποτέλεσμα πολλές φορές να καταρρέει ολόκληρο το σύστημα. Επίσης, στους 40 εγγραφείς παρουσιάστηκε άλλο μήνυμα λάθους όπως εξηγήθηκε πιο πάνω. Αυτό συνέβαινε πιθανώς λόγω της ανάγκης μεγαλύτερου ποσοστού μνήμης στις μηχανές ή και εξαιτίας της μεγάλης συμφόρησης του δικτύου. Έτσι, πρέπει να γίνεται προσεκτική επιλογή του αριθμού των εξυπηρετητών και των πελατών για να αποφεύγονται αυτού του είδους τα προβλήματα.

Επίσης, παρατηρείται σε πάρα πολλές εκτελέσεις, η πρώτη λειτουργία κάθε πελάτη να παίρνει πάρα πολύ χρόνο. Αυτό οφείλεται στη ταυτόχρονη εκκίνηση των διεργασιών και την πρόκληση μεγάλης συμφόρησης στο δίκτυο. Ως εκ τούτου, η πρώτη τιμή δεν συμπεριλαμβανόταν στα αποτελέσματα για να έχουμε πιο ρεαλιστικές τιμές.

Τέλος, το ποσοστό των λειτουργιών που απαιτούν δεύτερο γύρο επικοινωνίας δεν ξεπερνά το 33%, στη περίπτωση με τους περισσότερους πελάτες. Αυτό το ποσοστό

είναι ικανοποιητικό, αφού αυτό το σενάριο παρουσίαζε τη «χειρότερη περίπτωση», όπου ο χρόνος μεταξύ των δύο λειτουργιών είναι ο ίδιος και για τους δύο πελάτες και είναι πιθανότερο κάποιες λειτουργίες να εκτελεστούν ταυτόχρονα. Παρόλα αυτά, η ταυτοχρονία είναι λιγότερο πιθανή λόγω της ασυγχρονίας του δικτύου, ευνοόντας έτσι τον αλγόριθμο. Εδώ παρατηρείται πως ακόμη και όταν αυξάνονται οι εγγραφείς, μετά από κάποιο όριο το ποσοστό δεν αυξάνεται, λόγω του αριθμού των εξυπηρετητών.

Κεφάλαιο 6

Συμπεράσματα

6.1 Γενικά Συμπεράσματα

Σε αυτή τη διπλωματική εργασία υλοποιήθηκαν και αξιολογήθηκαν δύο αλγόριθμοι γραφής/ανάγνωσης ατομικών αντικειμένων σε ένα σύστημα με πολλαπλούς εγγραφείς και πολλαπλούς αναγνώστες. Η αξιολόγηση έγινε στο ρεαλιστικό και ασύγχρονο δίκτυο PlanetLab, όπου η κατάρρευση και η μη ανταπόκριση των κόμβων είναι πιθανή. Τα αποτελέσματα των δύο αλγορίθμων συγκρίθηκαν με σκοπό τη βαθύτερη αξιολόγηση του αλγορίθμου CWFR.

Γενικά, οι αλγόριθμοι συμπεριφέρονται όπως αναμενόταν, εκτός των περιπτώσεων που ο αριθμός των πελατών ήταν πολύ μεγάλος. Βάση των αποτελεσμάτων από τα πειράματα, ο αλγόριθμος CWFR σε ρεαλιστικό δίκτυο είναι αποδοτικότερος από τον απλό αλγόριθμο. Οι αργές λειτουργίες ανάγνωσης δεν ξεπερνούν το 33%, με αποτέλεσμα ο αλγόριθμος να συμπεριφέρεται τις περισσότερες φορές ως γρήγορος παρά ως ασθενές-ημιγρήγορος.

6.2 Μελλοντική Εργασία

Η αξιολόγηση των αλγορίθμων έγινε εξετάζοντας την απόδοση τους στο σύστημα απαρτίας Crumbling Walls, μεταβάλλοντας τον αριθμό των εγγραφέων και των αναγνωστών. Σαν μελλοντική εργασία, θα μπορούσαν να αξιολογηθούν σε διαφορετικά συστήματα απαρτίας, όπως είναι το matrix και το majority. Επίσης, θα μπορούσε να μεταβληθεί ο χρόνος μεταξύ δύο εκτελέσεων, για να αξιολογηθεί η συχνότητα αργών

λειτουργιών ανάγνωσης στη περίπτωση πιο συχνών λειτουργιών εγγραφής. Ακόμη, η προσομοίωση καταρρεύσεων των εξυπηρετητών θα βοηθούσε στην εξέταση της ευρωστίας του συστήματος και της ικανότητας εκτέλεσης των λειτουργιών αποδοτικά.

Η σημαντικότητα αυτού του αλγορίθμου έγγυται στην ανάγκη απόκτησης ενός εύρωστου, αξιόπιστου και αποδοτικού συστήματος, στο οποίο τα δεδομένα μας θα υπάρχουν σε πολλά αντίγραφα σε διαφορετικές τοποθεσίες, με σκοπό την εύκολη πρόσβαση τους από οποιοδήποτε σημείο. Παράδειγμα ενός τέτοιου συστήματος είναι το Dropbox. Στο Dropbox, όταν γίνεται ταυτόχρονη πρόσβαση στο ίδιο χώρο μνήμης, αποθηκεύονταν δύο διαφορετικά αντίγραφα όπου το κάθε ένα κρατούσε τις αλλαγές που έγιναν από τον κάθε τόπο ξεχωριστά. Σε αντίθεση, ο αλγόριθμος CWFR δεν δημιουργεί δύο διαφορετικά αντίγραφα, αλλά βάζει τις ταυτόχρονες λειτουργίες σε μια διάταξη για να γίνουν γραμμικά και να μην δημιουργηθούν διαφορετικά αντίγραφα. Σαφώς, αυτός ο τρόπος είναι αποδοτικότερος και πιο πρακτικός.

Βιβλιογραφία

[1] Attiya, H, Bar-Noy, A & Dolev, D (1995) 'Sharing Memory Robustly in Message-Passing Systems', *Journal of the ACM*,42(1),pp. 124-142.

[2] Dutta, P, Guerraoui, R, Levy, R, Chakraborty, A (2004) 'How fast can a distributed atomic read be?', *Proceedings of the 23rd ACM symposium of Principles of Distributed Computing (PODC)*, pp.236-245.

[3] Englert, B, Georgiou, C, Musial, P, Nicolaou, N, Shvartsman, A (2009) 'On the Efficiency of Atomic Multi-Reader, Multi-Writer Distributed Memory', *Proceedings of the 13th International Conference on Principles of Distributed Systems (OPODIS 2009)*, Nimes, France.

[4] Georgiou, C, Kentros, S, Nicolaou, N, Shvartsman, A (2009) 'Analyzing the Number of Slow Reads for Semifast Atomic Read/Write Register Implementations', *Proceedings of the 21st International Conference on Parallel and Distributed Computing and Systems (PDCS 2009)*, Cambridge, MA, pp. 229-236.

[5] Georgiou, C, Nicolaou, N & Shvartsman, A (2009) 'Fault-Tolerant SemiFast Implementations of Atomic Read/Write Registers', *Journal of Parallel and Distributed Computing (JPDC)*,69(1), pp. 62-79.

[6] Georgiou, C, Nicolaou, N, & Shvartsman, A (2008) 'On the Robustness of (Semi)Fast Quorum-Based Implementations of Atomic Shared Memory', *Proceedings of the 22nd International Symposium on Distributed Computing (DISC 2008)*, Arcachon, France, pp. 289-304.

[7] Hall, B (2005) *Beej's Guide to Network Programming Using Internet Sockets*, Version 2.3.23.

[8] Kshemkalyani, A & Singhal, M (2008) *Distributed Computing Principles, Algorithms, and Systems*, 1st ed, Cambridge University Press.

[9] Lynch, N (1996) *Distributed Algorithms*, Morgan Kaufmann Publishers.

[10] Lynch, N & Shvartsman, A (1997) 'Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts', *Proceedings of Symposium on Fault-Tolerant Computing*, pp.272-281.

[11] Malkhi, D & Reiter, M (1998) 'Byzantine Quorum Systems', *Distributed Computing*,11(4), pp. 203-213.

[12] Navaneeth, K (2001) *The Jxta solution to P2P*, ανάκτηση στις 1 Μάη 2011, <http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-jxta.html>.

[13] Peleg, D & Wool, A (2009) 'Crumbling walls: A class of high availability quorum system', *Proceedings of 14th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 120-129.

[14] *POSIX thread (pthread) libraries*, ανάκτηση στις 10 Μάη 2011, <http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html>.

[15] Robbins, D (2000). *POSIX threads explained*, Ανάκτηση στις 10 Μάη 2011, <http://www.ibm.com/developerworks/linux/library/l-posix1/index.html>.

[16] Savva, A (2010) *Evaluation of algorithms implementing multiple writer multiple reader atomic registers on Planet-Lab*, Master Thesis, Department of Computer Science, University of Cyprus.

[17] University of Princeton (2007), PlanetLab Home Page, <http://www.planet-lab.org/>.

[18] University of Southern California (1981) RFC 793, Transmission Control Protocol, Darpa Internet Program Protocol Specification, ανάκτηση στις 1 Μάη 2011, <http://www.rfc-editor.org/rfc/rfc793.txt>.

[19] Ζεϊναλιπούρ, Δ (2010) *Σημειώσεις Μαθήματος ΕΠΛ371 – Προγραμματισμός Συστημάτων*, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, <http://www.cs.ucy.ac.cy/courses/EPL371/>.

[20] Χατζηδημητρίου, Α (2009). *Υλοποίηση και πειραματική αξιολόγηση του αλγορίθμου ατομικών αντικειμένων γραφής/ανάγνωσης SLIQ*, Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

[21] Kayne, R (2011) *What is BitTorrent?*, ανάκτηση στις 30 Μάη 2011, <http://www.wisegeek.com/what-is-bittorrent.htm>.

Παράρτημα Α

Σε αυτό το Παράρτημα δίνονται τα προγράμματα του εξυπηρετητή και του εγγραφέα για τους δύο αλγορίθμους. Εφόσον, οι δύο αλγόριθμοι διαφέρουν μόνο στον αναγνώστη, δίνεται ο κώδικας του αναγνώστη με τον επιπρόσθετο κώδικα για τον αλγόριθμο CWFR να δίνεται με έντονα γράμματα (bold letters).

```
/*************************************************************************
 * Name & Surname: Maria Stylianou       ID: 1012147                     *
 * Department: Computer Science of University of Cyprus                  *
 * Last Update: 30-MAR-2011                                              *
 * Title of program: fserver.c                                           *
 *                                                                       *
 * Functionality:                                                        *
 * It has a pathetic role in the system;                                 *
 * It receives messages from clients and answers back. When it receives: *
 * (a) a WRITE message, it checks if it is valid. If positive, it saves
 *                      the whole message and sends a WRITEACK message   *
 * (b) a READ message, it checks if it is valid. If positive, it sends
                        back a READACK message with its local message    */
/*************************************************************************
 *                              SERVER                                   *
 *************************************************************************/
/* LIBRARIES ************************************************************/
#include <stdio.h>              /* For I/O                                */
#include <stdlib.h>             /* For exit() malloc() free()             */
#include <string.h>             /* For strings - strcpy() strcmp() strcat()*/
#include <pthread.h>            /* For threads                            */
#include <unistd.h>             /* For close()                            */
#include <sys/socket.h>  /* For sockets - socket() bind() listen()        *
                                    accept() connect() shutdown()        *
                                    close() setsockopt()                 */
#include <sys/types.h>   /* For sockets -  same as above                  */
#include <netinet/in.h>  /* For Internet sockets -                        *
                            address formating in a general structure     */
#include <arpa/inet.h>     /* For inet_ntop()                            */
#include <netdb.h>            /* For getaddrinfo()                       */
#include <signal.h>        /* For struct sigaction                       */
/* CONSANTS *************************************************************/
#define MAX_CLIENTS 160  /* Maximum number of accepted clients           */
#define FIELDS 2         /* Fields of the table with the requests        */
#define MAX_QUEUE 160    /* Maximum number of requests for connections   *
                            waiting in the queue                         */
#define BUFLEN 256              /* Buffer length                         */
#define ACK "ACK"               /* To be concatenated with the type of the *
                                   received message                      */
#define READ "READ"             /* READ message                          */
#define WRITE "WRITE"    /* WRITE message                                */
#define EXIT "EXIT"             /* EXIT message for closing the connection */
#define SIZE 10          /* Size of type of message                      */
#define FILENAME_LEN 20  /* Length for filename                         */
/* STRUCTURES ***********************************************************/
/* Structure for node                                                    */
typedef struct node_t node;

struct node_t {
    int sock;           /* Socket descriptor                             */
    node *next;         /* Pointer to next node                          */
};

/* Structure for queue                                                   */
typedef struct {
    node *head;         /* Head of the queue                             */
    node *tail;         /* Tail of the queue                             */
    int size;           /* Size of the queue                             */
} queue;

/* Structure for tag                                                     */
typedef struct {
    int ts;             /* Timestamp                                     */
```

```
    int wid;            /* Writer's id                                          */
} tag_type;

/* Structure for message                                                        */
typedef struct {
    char type[SIZE];    /* Type of message; READACK WRITEACK                    */
    int pid;            /* Process ID                                           */
    tag_type tag;       /* Tag: timestamp and writer's id                       */
    int value;          /* Object's New value                                   */
    int reqNo;          /* Number of client's request                           */
} message;
/* FUNCTIONS ************************************************************/
void sigchld_handler(int signo);
void *get_in_addr(struct sockaddr *sa);
/*Threads Functions                                                             */
void enqueue(int sock);
int dequeue();
void createTpool(FILE *fout);
void *handleClient(void *args);
void *check();
void initQueue();
/* Clients Table Functions                                                      */
void initTable();
void printfClientStatus();
void saveClientId(int pid);
int returnClientPos(message *msg);
void rmvClient(int position);
/* Message Functions                                                    */
void initMsg(message *msg);
void printfMsg(message *msg);
void fprintfMsg(FILE *fout, message *msg);
void strToMsg(char *token, message *msg);
void msgToStr(char *buf, message msg);
int checkValid(FILE *fout, int position, message *msg);
int notFail(FILE *fout, int position);
int cmpTag(tag_type tag1, tag_type tag2);
void msgCpy(message *dest, message src, int cmp);
/* GLOBAL VARIABLES *****************************************************/
queue q;
pthread_t pid[MAX_CLIENTS]; /* Table with thread IDs                    */
pthread_t pidc;             /* Thread ID for the "master thread"        */
int threadnum;              /* Number of threads                                */
pthread_mutex_t mutex;      /* Synchronization mechanism for threads            */
pthread_cond_t cond;        /* Condition variable for threads           */
message smsg;               /* Local message of server                          */
message cmsg;               /* Message from client                              */
int clients[MAX_CLIENTS][FIELDS]; /* Table with clients' IDs and
                                     clients' most recent request Number        */
int failFreq;               /* In auto-procedure - Frequency of failures  */
/************************************************************************/
/*                              MAIN                                            */
/************************************************************************/
int main(int argc, char *argv[]) {
    /* Definitions                                                              */
    int id;                     /* Server's id                                  */
    int port;                   /* Server's port                                */
    int sock;                   /* Socket file descriptor               */
    int newsock;                /* Socket fd for a new connection       */
    struct sockaddr_storage client; /* Client's information                     */
    socklen_t clientlen;        /* Address of client's address          */
    int yes;                    /* Flag for setsockopt()                */
    FILE *fout;                 /* Pointer to output file               */
    char outFile[FILENAME_LEN]; /* Name of output log file                      */
    int status;                 /* Result from getaddrinfo()                    */
    struct addrinfo hints;      /* For getaddrinfo()                            */
    struct addrinfo *servinfo;  /* Point to info for ipv4                       */
    struct addrinfo *p;         /* Counter                                      */
    struct sigaction sa;
    char hostname[INET6_ADDRSTRLEN]; /* Hostname                            */
    /************************************************************************/
    /* Check if input arguments are given                                       */
    if (argc != 4) {
        printf("\nUsage of file: %s <serverID> <port number> <failFreq>\
```

```
        \n(0<=failFreq<=100)\n",
           argv[0]);
    exit(1);
}
/* Convert arguments to integers                                        */
id = atoi(argv[1]);
port = atoi(argv[2]);
failFreq = atoi(argv[3]);
printf("FAIL = %d \n", failFreq);
sprintf(outFile, "log-fs%d.dat", id);

/* Open file to write                                                   */
if (!(fout = fopen(outFile, "w"))) { /* Servers file                    */
   perror("fopen()");
   exit(1);
}
printf("\nStart fserver...\n");
fprintf(fout,"\nStart fserver...\n");

srand(time(NULL));      /* Begin rand() seed related to current time  */

memset(&hints, 0, sizeof hints); /* Initialise struct hints to zero   */
hints.ai_family = AF_UNSPEC;     /* Either IPv4 or IPv6               */
hints.ai_socktype = SOCK_STREAM; /* TCP stream sockets                */
hints.ai_flags = AI_PASSIVE;     /* Fill in IP automatically          */

/* Get server's information                                             */
if ((status = getaddrinfo(NULL, argv[2], &hints, &servinfo)) != 0) {
    fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(status));
    exit(1);
}
/* gai_strerror prints the error if there is a non-zero result          */
/* servinfo now points to a linked list of 1 or more struct addrinfos */

/* Create socket                                                        */
if ((sock = socket(servinfo->ai_family, servinfo->ai_socktype,
                   servinfo->ai_protocol)) < 0) {
    perror("socket()");
    exit(1);
}

yes = 1;
// lose the pesky "Address already in use" error message
if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof (int)) < 0) {
    perror("setsockopt");
    close(sock);
    exit(1);
}

/* Bind socket to address                                               */
if (bind(sock, servinfo->ai_addr, servinfo->ai_addrlen) < 0) {
    perror("bind()");
    close(sock);
    exit(1);
}

freeaddrinfo(servinfo); // free the linked-list

/* Listen for connections                                               */
if (listen(sock, MAX_QUEUE) < 0) {
    perror("listen()");
    close(sock);
    exit(1);
}


/* Initialize servers queue, clients table, server & client message   */
initQueue();
initTable();
initMsg(&smsg);
initMsg(&cmsg);

/* Create Threadpool                                                        */
```

```
    createTpool(fout);


    while (1) {
        printf("<Wait for connection on TCP port %d>\n", port);
        fprintf(fout, "<Wait for connection on TCP port %d>\n", port);
        printf("Local message: ");
        fprintf(fout, "Local message: ");
        printfMsg(&smsg);
        fprintfMsg(fout, &smsg);

        /* Accept connection                                                */
        clientlen = sizeof client;
        if ((newsock = accept(sock, (struct sockaddr *) & client,
                              &clientlen)) < 0) {
            printf("server: id=%d => ",id);
            perror("accept()");
            close(sock);
            exit(1);
        }
        /* Find client's address                                            */
        inet_ntop(client.ss_family,
                  get_in_addr((struct sockaddr *) & client),
                  hostname, sizeof hostname);

        printf("************************************************\n");
        fprintf(fout, "************************************************\n");

        printf("<Accepted connection from %s - ", hostname);
        fprintf(fout, "<Accepted connection from %s - ", hostname);
        if (threadnum) {        /* If there is an available thread          */
            printf("a thread has undertaken the connection>\n");
            fprintf(fout, "a thread has undertaken the connection>\n");
            pthread_mutex_lock(&mutex);
            enqueue(newsock);   /* -> Add connection fd in the queue        */
            pthread_mutex_unlock(&mutex);
        }
        else {                  /* Else                                     */
            printf("clients queue is full>\n \
      <Closing Connection with socket %d...>\n\n", newsock);
            fprintf(fout, "clients queue is full>\n \
      <Closing Connection with socket %d...>\n\n", newsock);
            shutdown(newsock, SHUT_RDWR); /* -> Close connection          */
            close(newsock);
        }
    } /* End of while(1)                                                    */
    fclose(fout);
    return;
} /* End of main()                                                          */
/*************************************************************************/
/*                              FUNCTIONS                                  */
/*************************************************************************/
/* Get sockaddr, IPv4 or IPv6:                                          */
void *get_in_addr(struct sockaddr *sa) {
    if (sa->sa_family == AF_INET) {
        return &(((struct sockaddr_in*) sa)->sin_addr);
    }

    return &(((struct sockaddr_in6*) sa)->sin6_addr);
}
/*************************************************************************/
/* Initialize queue                                                       */
void initQueue() {
    q.head = NULL;
    q.tail = NULL;
    q.size = 0;
} /* End of initQueue()                                                   */
/*************************************************************************/

/* Add a node to the queue (at the tail)                                  */
void enqueue(int sock) {
    node* newNode = (node *) malloc(sizeof (node)); /* Create node        */
    newNode->sock = sock;
```

```
    newNode->next = NULL;

    if (!q.head)            /* If queue is empty                                */
        q.head = q.tail = newNode;  /* -> Add newNode as a head and tail  */
    else { /* Else                                                                */
        q.tail->next = newNode;     /* -> Add newNode as a tail           */
        q.tail = newNode;
    }
    ++(q.size);                     /* Increase queue size +1                   */
} /* End of enqueue()                                                           */
/*****************************************************************************/

/* Remove a node from the queue (from the head)
   Return connection file descriptor                                          */
int dequeue() {
    int socket;
    node* tmp;

    if (!q.head) {                  /* If queue is empty                      */
        printf("Queue is empty!\n");    /* -> nothing to remove           */
        return -1;
    }                               /* Else                                   */
    socket = q.head->sock;          /* -> Save connection fd          */
    tmp = q.head;                   /*    and remove last fd from queue */
    q.head = q.head->next;
    free(tmp);

    --(q.size);                     /* Dicrease size -1                       */
    if (q.size == 0)                /* If queue is empty                      */
        q.tail = NULL;              /* -> Direct tail to show to NULL   */

    return socket;                  /* Return socket of connection    */
} /* End of dequeue()                                                          */
/*****************************************************************************/

/* Create thread pool                                                         */
void createTpool(FILE *fout) {
    int i;

    /* Handle Requests                                                    */
    for (i = 0; i < MAX_CLIENTS; i++) {
        pthread_create(&pid[i], NULL, handleClient, (void *) fout);
    }
    threadnum = i;                  /* Number of threads              */
    /* Check if there's request                                           */
    pthread_create(&pidc, NULL, check, NULL);
} /* End of createTpool                                                       */
/*****************************************************************************/

/* Check every second if there is a request                                   */
void *check() {
    while (1) {
        if (threadnum && q.head != NULL) /* If there is an available thread
                                            & queue is not empty          */
            pthread_cond_signal(&cond); /* -> Send a signal to thread     */
        sleep(1);
    }
} /* End of check()                                                                */
/*****************************************************************************/

/* Function for handling a new request                                        */
void *handleClient(void *args) {
    int socket;             /* Connection fd extracted from queue             */
    char *token;            /* Usage: To keep part of the line from the file*/
    int i;                  /* Counter                                        */
    char buf[BUFLEN];       /* Buffer                                         */
    int position;           /* Client position from clients table         */
    int cmp;                /* Flag for comparing 2 tags                  */
    FILE *fout;             /* File Descriptor for output file            */
    int times;              /* Counts requests                            */

    fout = (FILE *) args;
    while (1) {
```

```
        pthread_mutex_lock(&mutex);         /* Lock mutex                */
        pthread_cond_wait(&cond, &mutex);   /* Block the calling thread
                                               until cond is signalled.         */
        pthread_mutex_unlock(&mutex);       /* Unlock mutex                      */

        pthread_mutex_lock(&mutex);
        threadnum--;                        /* Reduce number of threads   */
        pthread_mutex_unlock(&mutex);

        pthread_mutex_lock(&mutex);
        socket = dequeue();                 /* Extract the fd from queue  */
        pthread_mutex_unlock(&mutex);

        times = 1;
        while (1) {
            bzero(buf, strlen(buf));                /* Initialize buffer  */
            if (read(socket, buf, BUFLEN) < 0) {    /* Receive Message    */
                perror("read()");
                close(socket);
                exit(1);
            }
            if (buf[0] != 'W' && buf[0] != 'R'){
                if (!strcmp(buf, EXIT)) { /* If message is Exit            */
                    printf("\n'%s' received from clientId: %d\n", buf, cmsg.pid);
                    fprintf(fout, "\n'%s' received from clientId: %s\n", buf, cmsg.type);
                }
                else{ /* To avoid connecting with unwanted clients         */
                    printf("---Unknown type of message \n");
                    fprintf(fout, "---Unknown type of message \n");
                }
                printf("<Closing Connection with socket %d...>\n\n", socket);
                fprintf(fout, "<Closing Connection with socket %d...>\n\n", socket);
                shutdown(socket, SHUT_RDWR);
                close(socket);
                break;
            }
            if (buf[0] == 'W' || buf[0] == 'R') {
                pthread_mutex_lock(&mutex);
                token = strtok(buf, ",");   /* Extract type of message    */
                strToMsg(token, &cmsg); /* Save string message to cmsg    */
                pthread_mutex_unlock(&mutex);

                if (times == 1) {           /* If it is the 1st msg              */
                    times++;
                    pthread_mutex_lock(&mutex);
                    saveClientId(cmsg.pid); /* Save Client's id          */
                    pthread_mutex_unlock(&mutex);
                }
                printf("---Receive a %s message from clientId: %d\t", cmsg.type, cmsg.pid);
                fprintf(fout, "---Receive a %s message from clientId: %d\t", cmsg.type,
cmsg.pid);
                printfMsg(&cmsg);
                fprintfMsg(fout, &cmsg);
                position = returnClientPos(&cmsg); /* Find client's position*/

                /* If cmsg is valid && server not failed                */
                if (checkValid(fout, position, &cmsg) && notFail(fout, position))
                  {

                    pthread_mutex_lock(&mutex);
                    clients[position][1] = cmsg.reqNo;/*->Save client'sreqNo*/
                    cmp = cmpTag(cmsg.tag, smsg.tag); /*->Compare server&client
                                                                          tags */
                    msgCpy(&smsg, cmsg, cmp);      /* -> Save cmsg to smsg        */
                    pthread_mutex_unlock(&mutex);
                    bzero(buf, strlen(buf));        /* Initialize buffer  */
                    msgToStr(buf, smsg);
                    printf("---Send an ACK message\t");
                    fprintf(fout, "---Send an ACK message\t");
                    printfMsg(&smsg);
                    fprintfMsg(fout, &smsg);
                    if (write(socket, buf, BUFLEN) < 0) { /* Send ACK     */
                        perror("write");
```

```
                    exit(EXIT_FAILURE);
                }
                printf("<Waiting for next request...>\n\n");
                fprintf(fout, "<Waiting for next request...>\n\n");
            } /* End of if checkValid() && notFail()                */
        }
    } /* End of inner while(1)                                       */
    /* Client has closed connection                                  */
    pthread_mutex_lock(&mutex);
    threadnum++;                /* Increase number of threads        */
    rmvClient(position);        /* Remove client from table          */
    pthread_mutex_unlock(&mutex);
  } /* End of while(1)                                               */
} /* End of handleClient()                                         */
/*********************************************************************/

/* Initialize table                                                  */
void initTable() {
    int i, j;
    for (i = 0; i < MAX_CLIENTS; i++)
        for (j = 0; j < FIELDS; j++)
            clients[i][j] = 0;
} /* End of initTable()                                            */
/*********************************************************************/

/* Save client's id in clients table                                 */
void saveClientId(int pid) {
    int i;
    for (i = 0; i < MAX_CLIENTS; i++)
        if (clients[i][0] == 0) {
            clients[i][0] = pid;
            break;
        }
} /* End of saveClientId()                                         */
/*********************************************************************/

/* Print Clients' status (id and request number)                     */
void printfClientStatus() {
    int i;
    for (i = 0; i < MAX_CLIENTS; i++)
        printf("clients[%d][0]=%d, clients[%d][1]=%d\n",
               i, clients[i][0], i, clients[i][1]);
} /* End of printfClientStatus()                                   */
/*********************************************************************/

/* Based on a message
   Return Client's Position on the clients table. Return -1 otherwise */
int returnClientPos(message *msg) {
    int i;
    for (i = 0; i < MAX_CLIENTS; i++)
        if ((msg->pid == clients[i][0]))
            return i;
    return -1;
} /* End of returnClientPos()                                      */
/*********************************************************************/

/* Remove client from table                                          */
void rmvClient(int position) {
    clients[position][0] = 0;
    clients[position][1] = 0;
} /* End of rmvClient()                                            */
/*********************************************************************/

/* Initialize message                                                */
void initMsg(message *msg) {
    strcpy(msg->type, "\0");
    (msg->tag).ts = 0;
    (msg->tag).wid = 0;
    msg->value = -1;
    msg->reqNo = 0;
} /* End of initMsg()                                              */
/*********************************************************************/
```

```
/* Print message content                                                    */
void printfMsg(message *msg) {
    printf("(type,pid,<ts,wid>,value,req)\t(%s,%d,<%d,%d>,%d,%d)\n",
           msg->type, msg->pid, (msg->tag).ts, (msg->tag).wid, msg->value,
           msg->reqNo);
} /* End of printfMsg()                                                     */
/*************************************************************************/
/* Print message contents                                                                */
void fprintfMsg(FILE *fout, message *msg) {
    fprintf(fout, "(type,<ts,wid>,value,req)\t(%s,<%d,%d>,%d,%d)\n",
            msg->type, (msg->tag).ts, (msg->tag).wid, msg->value, msg->reqNo);
} /* End of fprintfMsg()                                                    */
/*************************************************************************/
/* Transform string to the structure message                                */
void strToMsg(char *token, message *msg) {
    strcpy(msg->type, token);
    token = strtok(NULL, ",");
    msg->pid = atoi(token);
    token = strtok(NULL, ",");
    (msg->tag).ts = atoi(token);
    token = strtok(NULL, ",");
    (msg->tag).wid = atoi(token);
    token = strtok(NULL, ",");
    msg->value = atoi(token);
    token = strtok(NULL, ",");
    msg->reqNo = atoi(token);
} /* End of strToMsg()                                                      */
/*************************************************************************/
/* Transform structure message to string                                    */
void msgToStr(char *buf, message msg) {
    sprintf(buf, "%s,%d,%d,%d,%d,%d\0", msg.type, msg.pid, (msg.tag).ts,
            (msg.tag).wid, msg.value, msg.reqNo);
} /* End of msgToStr()                                                      */
/*************************************************************************/
/* Check if the message received is valid
   (request number is the most recent)
   Return 1 if it is valid. Return 0 otherwise                              */
int checkValid(FILE *fout, int position, message *msg) {
    int i;
    if (msg->reqNo >= clients[position][1]) /* If most recent request     */
        return 1;
    printf("--- Ignore message - Request number is not the most recent\n");
    fprintf(fout, "--- Ignore message - Request number is not the most recent\n");
    return 0;
} /* End of checkValid()                                                    */
/*************************************************************************/
/* Check if server will fail                                                *
   If random > failFreq then fail                                                */
int notFail(FILE *fout, int position) {
    int num = 0; /* Random number                                                */

    num = rand() % 100; /* Random Number between [0...100]                  */
    printf("Fail rundom number = %d\n", num);
    fprintf(fout, "Fail rundom number = %d\n", num);
    if (num < failFreq)
        return 1;
    if (num >= failFreq) {
        printf("--- Ignore message - Server failed\n");
        fprintf(fout, "--- Ignore message - Server failed\n");
        return 0;
    }
} /* End of notFail()                                                       */
/*************************************************************************/
/* Based on two tags
   Rerun 1 if tag1 > tag2, 0 if tag1 == tag2, -1 if tag1 < tag2          */
int cmpTag(tag_type tag1, tag_type tag2) {
    if (tag1.ts > tag2.ts)                  /* If ts1 > ts2                 */
        return 1;                           /* -> tag1 > tag2               */
    if (tag1.ts == tag2.ts)                 /* If ts1 == ts2                */
        if (tag1.wid > tag2.wid)            /*    If wid1 > wid2            */
            return 1;                       /*    -> tag1 > tag2            */
    if (tag1.ts == tag2.ts)                 /*    If ts1== ts2              */
        return 0;                           /*    -> tag1 = tag2            */
```

```
    return -1;                              /* -> tag1 < tag2            */
} /* End of cmpTag()                                                     */
/*************************************************************************/
/* Copy src message into dest message depending on cmp                   */
void msgCpy(message *dest, message src, int cmp) {
    strcpy(dest->type, src.type);       /* Save type in dest             */
    strcat(dest->type, ACK);
    dest->reqNo = src.reqNo;            /* Save reqNo in dest            */
    dest->pid = src.pid;                /* Save pid in dest              */

    if (cmp == 1) {                     /* If src.tag > dest.tag         */
        (dest->tag).ts = (src.tag).ts;  /* -> Save ts                    */
        (dest->tag).wid = (src.tag).wid;/* -> Save wid                   */
        dest->value = src.value;        /* -> Save value                 */
    } /* End of if                                                       */
} /* End of msgCpy                                                       */
/*************************************************************************/
/*                           END OF FILE                                 */
/*************************************************************************/

/*************************************************************************
 * Name & Surname: Maria Stylianou       ID: 1012147                     *
 * Department: Computer Science of University of Cyprus                  *
 * Last Update: 30-MAR-2011                                              *
 * Title of program: fwriter.c                                           *
 *                                                                       *
 * Functionality:                                                        *
 * It has an active role in the system;                                  *
 * It writes a message, to the object which the servers hold, in 2RTT.   *
 * 1RTT: It sends a READ message to find the most recent message         *
 * 2RTT: It sends a WRITE message with the new value                     */
/*************************************************************************
 *                           WRITER                                      *
 *************************************************************************/
/* LIBRARIES *************************************************************/
#include <stdio.h>              /* For I/O                               */
#include <stdlib.h>             /* For exit() malloc() free()            */
#include <string.h>             /* For strings - strcpy() strcmp() strcat()*/
#include <unistd.h>             /* For close()                           */
#include <sys/socket.h>  /* For sockets - socket() bind() listen()       *
                                    accept() connect() shutdown()        *
                                    close() setsockopt()                 */
#include <sys/types.h>   /* For sockets -  same as above                 */
#include <netinet/in.h>  /* For Internet sockets -                       *
                            address formating in a general structure     */
#include <netdb.h>              /* For getaddrinfo()                     */
#include <time.h>               /* For time()                            */
#include <sys/time.h>
#include <sys/select.h>  /* For select()                                 */
/* CONSTANTS *************************************************************/
#define BUFLEN 256              /* Buffer Length                         */
#define SIZE 10          /* Size of type of message                      */
#define RANGE 100               /* Generate a random number [0-RANGE]    */
#define ACK "ACK"               /* To be concatenated with the type of the *
                                   received message                      */
#define READ "READ"             /* READ message                          */
#define WRITE "WRITE"    /* WRITE message                                */
#define EXIT "EXIT"             /* EXIT message for closing the connection */
#define FILENAME_LEN 20  /* Length for filename                          */
#define MINOP 1          /* Minimum option on the menu                   */
#define MAXOP 2          /* Maximum option on the menu                   */
#define AUTO 6           /* Number of arguments for auto-procedure       */
#define MAN 4                   /* Number of argumetns for manual procedure */
#define MILLION 1000000  /* One million - for math                       */
/* STRUCTURES ************************************************************/

/* Structure for tag                                                     */
typedef struct {
    int ts;              /* Timestamp                                    */
    int wid;             /* Writer's id                                  */
} tag_type;

/* Structure for message                                                 */
```

```
typedef struct {
    char type[SIZE];     /* Type of message: WRITE, READ                  */
    int pid;             /* Process id                                    */
    tag_type tag;        /* Tag: timestamp and writer's id                */
    int value;           /* Object's value                                */
    int reqNo;           /* Number of request                             */
} message;

/* Structure for server                                                   */
typedef struct {
    int id;              /* Server's Id                                   */
    int sock;            /* Socket file descriptor                        */
    char hostname[RANGE];   /* Server's hostname                          */
    struct addrinfo *serv;  /* Server                                     */
    int rand;            /* Flag showing if it was chosen randomly        */
    int ack;             /* Flag showing if it sent ACK                   */
    message msg;         /* Mesage                                        */
} server;

/* Structure for quorum                                                   */
typedef struct {
    int id;              /* Quorum's name                                 */
    int servNum;         /* Number of servers                             */
    int *servers;        /* Table with Servers' names                     */
} quorum;
/* FUNCTIONS ************************************************************/
/* Print Functions                                                        */
void printMenu(int *option);
void execError(FILE *fin, FILE *qin, FILE *out);
/* Initialization Functions                                               */
void initSrv(server *srvs, int srvNo);
void initIds(quorum *qrm, int len);
void clrRand(int size, server *srvs);
void clrAck(int size, server *srvs);
void initMsg(message *msg);
/* Message Functions                                                      */
void printfMsg(message *msg);
void fprintfMsg(FILE *fout, message *msg);
void fillMsg(message *msg, message *smsg, char *type, int pid);
void sndMsg(int sock, message *msg, char *buf, char *type);
void strToMsg(char *token, message *msg);
void msgToStr(char *buf, message msg);
int checkValid(FILE *fout, message *smsg, message *msg);
void sendProcd(FILE *out, char* type, int id, quorum *qrm, int qrmNo,
               int srvNo, message *wmsg, message *smsg, server *srvs);
/* Quorum File Functions                                                  */
int checkQrmFile(char *file, char * buf);
int returnServNum(char *file, char * buf, int cnt);
void fillInQrm(char *file, char *buf, quorum *qrm, int cnt);
/* Quorum Functions                                                       */
void sendToRdmSrv(FILE *out, int srvNo, char *buf, server *srvs,
                  message *wmsg, int *i);
int rcvAckFromQrm(FILE *out, char *buf, server *srvs, int srvNo, message *smsg,
                  message *wmsg, int qrmNo, quorum *qrm);
int checkQrmCmp(int qrmNo, quorum *qrm, server *srvs);
/* Tag Functions                                                          */
void findMaxTag(int srvNo, server *srvs, message *wmsg,
                message *smsg, quorum *qrm, FILE *out);
int cmpTag(tag_type tag1, tag_type tag2);
void tagCpy(tag_type* dest, tag_type src);
/************************************************************************/
/*                              MAIN                                    */
/************************************************************************/
main(int argc, char *argv[]) {
    /* Definitions and (some) Initializations                             */
    FILE* fin;           /* File descriptor of servers file               */
    FILE* qin;           /* File descriptor of quorum system file         */
    int id=0;            /* Writer's id                                   */
    int srvNo = 0;       /* Total number of servers                       */
    int qrmNo = 0;       /* Total number of quorums                       */
    int port = 0;        /* Server's port                                 */
    int i = 0;           /* Counters                                      */
    char buf[BUFLEN];    /* Buffer                                        */
```

```
int option = 0;     /* User's option                                        */
int ready = 0;      /* Flag showing if writer is ready to make
                       a new request                                        */
message wmsg, smsg; /* Writer's and server's message                  */
quorum *qrm;        /* Quorum System                                                */
server *srvs;       /* Servers' information                                 */
int opNo = 0;       /* In auto-procedure - Number of operations             */
int opCnt = 0;      /* Counter of operations                          */
float opFreq = 0;   /* In auto-procedure - Frequency of operations    */
char *opFreqStr;    /* Frequency of operations converted to string    */
int cond = 0;       /* Condition - automatic or manual procedure            */
struct timeval tim;
double start, end;  /* Time: start and end                                  */
double opLat = 0;   /* Operation Lattency                                   */
char outFile[FILENAME_LEN]; /* Name of output log file                      */
FILE *out;
int dummy;      /* Keep dummy numbers                          */
int status;         /* Result from getaddrinfo()                            */
struct addrinfo hints;      /* For getaddrinfo()                            */
struct addrinfo *ipv4info;  /* Point to info for ipv4                       */
struct addrinfo *p; /* Counter                                                      */
char prt[5];        /* Port converted to string                             */
char ipstr[INET6_ADDRSTRLEN]; /* IP converted to string                     */
void *addr;         /* Holds the address                                    */
char *ipver;        /* Message showing the Internet Protocol          */
/*********************************************************************/
/* Check if server's host name and port number are given                    */
if (argc != MAN && argc != AUTO) {
    printf("\nUsage of file (if auto-procedure give parameters of parenthesis):\
\n%s <writerID> <serversFile> <quorumFile> (<operationsNo> <operationsFreq>)\n",
           argv[0]);
    exit(EXIT_FAILURE);
}
id=atoi(argv[1]);
sprintf(outFile, "log-fw%d.dat", id);
/* Open/Create file to write                                                    */
if (!(out = fopen(outFile, "w"))) { /* Servers file                         */
    perror("fopen()");
    exit(1);
}

printf("\nStart fwriter...\n");
fprintf(out, "Start fwriter...\n");
sleep(5);
printf("Analysing data...\n");
fprintf(out, "Analysing data...\n");
/* Check if automatic procedure and save new parameters                     */
if (argc == AUTO) {
    opNo = atoi(argv[4]);
    opFreq = strtof(argv[5], &opFreqStr);
    printf("Total operations: %d\n", opNo);
    fprintf(out, "Total operations: %d\n", opNo);
    printf("Frequency of operations: %.2f\n", opFreq);
    fprintf(out, "Frequency of operations: %.2f\n", opFreq);
}
/* Open servers file & quorum file                                          */
if (!(fin = fopen(argv[2], "r"))) { /* Servers file                         */
    printf("fwriter: id=%d => ",id);
    perror("fopen()");
    exit(EXIT_FAILURE);
}
if (!(qin = fopen(argv[3], "r"))) { /* Quorum file                          */
    printf("fwriter: id=%d => ",id);
    perror("fopen()");
    exit(EXIT_FAILURE);
}
/* Find total number of servers & quorums                                   */
fscanf(fin, "%d", &srvNo);
fscanf(qin, "%d", &qrmNo);

/* Create dynamically 2 tables for: quorum struc and servers struc    */
qrm = (quorum *) malloc(qrmNo * sizeof (quorum));
srvs = (server *) malloc(srvNo * sizeof (server));
```

```
/* Initialize quorum ids, writer and servers message,tables with hosts */
initIds(qrm, qrmNo);
initMsg(&wmsg);
initMsg(&smsg);
initSrv(srvs, srvNo);
for (i=0; i<srvNo; i++)
    strcpy(srvs[i].hostname, "localhost");

/* Clear flag tables                                                    */
clrRand(srvNo, srvs);
clrAck(srvNo, srvs);

/* Print Information                                                    */
printf("Total servers: %d\n", srvNo);
fprintf(out, "Total servers: %d\n", srvNo);
printf("Total quorums: %d\n", qrmNo);
fprintf(out, "Total quorums: %d\n", qrmNo);

/* Check quorum file's format                                           */
if (!checkQrmFile(argv[3], buf))
    execError(fin, qin, out);

/* Fill in Quorum System - For each quorum:                             */
for (i = 0; i < qrmNo; i++) {
    /* Save number of servers                                           */
    qrm[i].servNum = returnServNum(argv[3], buf, i);
    /* Create dynamically a table of servers id belonging in quorum   */
    qrm[i].servers = (int *) malloc(qrm[i].servNum * sizeof (int));
    /* Fill in quorum's id and servers ids                              */
    fillInQrm(argv[3], buf, &qrm[i], i);
}

for (i = 0; i < srvNo; i++) {       /* For each server                  */
    /* Find server's id, port and hostname                              */
    fscanf(fin, "%d %d %s %d", &(srvs[i].id), &port, srvs[i].hostname, &dummy);

    /* Create socket - Fill in the table sock[]                         */
    memset(&hints, 0, sizeof hints); /* Initialise  hints to zero     */
    hints.ai_family = AF_UNSPEC;     /* Either IPv4 or IPv6           */
    hints.ai_socktype = SOCK_STREAM; /* TCP stream sockets            */
    sprintf(prt, "%d", port);

    /* Get server's information                                         */
    if ((status = getaddrinfo(srvs[i].hostname, prt, &hints, &srvs[i].serv)) != 0) {
        perror("getaddrinfo");
        exit(1);
    }
    /* srvs[i].serv now points to a linked list of 1 or more struct addrinfos*/
    /* For each struct in the linked list                               */
    for (p = srvs[i].serv; p != NULL; p = p->ai_next) {
        if (p->ai_family == AF_INET) { // IPv4
            struct sockaddr_in *ipv4 = (struct sockaddr_in *) p->ai_addr;
            addr = &(ipv4->sin_addr);
            ipver = "IPv4";
            ipv4info = p;
        }
        else { // IPv6
            struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *) p->ai_addr;
            addr = &(ipv6->sin6_addr);
            ipver = "IPv6";
        }

        /* Convert the IP to a string and print it                      */
        inet_ntop(p->ai_family, addr, ipstr, sizeof ipstr);
        printf("  %s: %s\n", ipver, ipstr);

        /* Request connection                                           */
        if ((srvs[i].sock = socket(p->ai_family, p->ai_socktype,
                                   p->ai_protocol)) < 0) {
            perror("socket()");
        }
        if (connect(srvs[i].sock, p->ai_addr, p->ai_addrlen) < 0) {
```

```
            perror("connect()");
            close(srvs[i].sock);
            continue;
        }
        srvs[i].serv = p;
        printf("Requested connection established to host %s - port %d - socket %d\n",
               srvs[i].hostname, port, srvs[i].sock);
        fprintf(out, "Requested connection established to host %s - port %d - socket %d\n",
                srvs[i].hostname, port, srvs[i].sock);
    }
} /* End of for each server                                                */

fclose(fin);        /* Close fd to servers file                           */
fclose(qin);        /* Close fd to quorums file                           */
ready = 1;          /* Writer is ready to make a request                  */

if (argc == AUTO)   /* If automatic procedure                        */
    cond = opNo + 1;/* -> While number of operations+1 for exit           */
else                /* If manual procedure                                */
    cond = 1;       /* -> Endless while                                   */

while (cond) {
    if (argc == AUTO) { /* If automatic procedure                         */
        cond--;         /* -> Dicrease counter                            */
        option = 1;
        if (!cond)      /* extra round => time to exit                    */
            option = 2;
        usleep(opFreq * MILLION);
    }
    else {
        printMenu(&option); /* Print Menu                                 */
    }
    switch (option) {
    case 1:         /* Option 1: Write to object                          */
        /* Ensure that a request can happen only if
         * previous one has finished                                 */
        if (ready == 0) {
            printf("Last write request hasn't finished\n");
            fprintf(out,"Last write request hasn't finished\n");
            cond++;
            continue;
        }
        ready = 0; /* Set flag to 0 - This request hasn't finished    */
        gettimeofday(&tim, NULL); /* Current Time - 1st                   */
        start=tim.tv_sec+(tim.tv_usec/1000000.0);
        opCnt++;
        printf("Operation No.: %d\n", opCnt);
        fprintf(out, "Operation No.: %d\n", opCnt);
        /* 1st RTT: Send a READ message to all servers                    */
        sendProcd(out, READ, atoi(argv[1]), qrm, qrmNo, srvNo,
                  &wmsg, &smsg, srvs);
        /* 2nd RTT: Send a WRITE message to all servers                   */
        sendProcd(out, WRITE, atoi(argv[1]), qrm, qrmNo, srvNo,
                  &wmsg, &smsg, srvs);

        ready = 1; /*This request has now finished                        */
        gettimeofday(&tim, NULL); /* Current Time - 2nd                   */
        end=tim.tv_sec+(tim.tv_usec/1000000.0);
        opLat=end-start;
        printf("Operation Lattency = %.4f seconds\n", opLat);
        printf("************************************************************\n");
        printf("************************************************************\n");
        fprintf(out, "Operation Lattency = %.4f seconds\n", opLat);
        fprintf(out, "************************************************************\n");
        fprintf(out, "************************************************************\n");
        break;
    case 2:         /* Option 2: Exit                                     */
        for (i = 0; i < srvNo; i++) {
            sndMsg(srvs[i].sock, &wmsg, buf, EXIT);
        }
        printf("Exiting...\n\n");
        fprintf(out,"Exiting...\n\n");
        exit(EXIT_FAILURE);
```

```
            break;
        } /* End of switch(option)                                              */
    } /* End of while(cond)                                                     */
    fclose(out);
    return;
} /* End of main()                                                             */
/***************************************************************************/

/* Print Menu                                                                 */
void printMenu(int *option) {
    do {
        printf("\n");
        printf("  ********\n");
        printf("  * Menu *\n");
        printf("  ********\n");
        printf("1. Write to object\n");
        printf("2. Exit\n");
        printf("Option: ");
        scanf("%d", option);
        printf("\n");
    }
    while (*option < 1 || *option > 2);
} /* End of printMenu()                                                        */
/***************************************************************************/

/* Print error message, close files and exit                                  */
void execError(FILE *fin, FILE *qin, FILE *out) {
    printf("Wrong format in the quorum system file.\n");
    printf("Please correct the file!\n\nExiting...\n\n");
    fprintf(out, "Wrong format in the quorum system file.\n");
    fprintf(out, "Please correct the file!\n\nExiting...\n\n");
    fclose(fin);
    fclose(qin);
    exit(EXIT_FAILURE);
} /* End of execError()                                                        */
/***************************************************************************/

/* Initialize structure server                                                */
void initSrv(server *srvs, int srvNo) {
    int i;
    for (i = 0; i < srvNo; i++) {
        srvs[i].id = 0;
        srvs[i].rand = 0;
        srvs[i].ack = 0;
        initMsg(&srvs[i].msg);
    }
} /* End of initSrv()                                                          */
/***************************************************************************/

/* Initialize table                                                           */
void initIds(quorum *qrm, int len) {
    int i;
    for (i = 0; i < len; i++)
        qrm[i].id = 0;
} /* End of initIds()                                                          */
/***************************************************************************/

/* Set all servers rand to 0 (means: they haven't been chosen)                */
void clrRand(int size, server *srvs) {
    int i;
    for (i = 0; i < size; i++)
        srvs[i].rand = 0;
} /* End of clrRand()                                                          */
/***************************************************************************/

/* Set all servers Ack to 0 (means: they haven't replied)                     */
void clrAck(int size, server *srvs) {
    int i;
    for (i = 0; i < size; i++)
        srvs[i].ack = 0;
} /* End of clrAck()                                                           */
/***************************************************************************/
```

```c
/* Initialize message                                                    */
void initMsg(message *msg) {
    strcpy(msg->type, "\0");
    msg->pid = 0;
    (msg->tag).ts = 0;
    (msg->tag).wid = 0;
    msg->value = -1;
    msg->reqNo = 0;
} /* End of initMsg()                                                   */
/************************************************************************/

/* Print message contents                                                */
void printfMsg(message *msg) {
    printf("(type,pid,<ts,wid>,value,req)\t(%s,%d,<%d,%d>,%d,%d)\n",
           msg->type, msg->pid, (msg->tag).ts, (msg->tag).wid,
           msg->value, msg->reqNo);
} /* End of printfMsg()                                                 */
/************************************************************************/

/* Print message contents into a file                                    */
void fprintfMsg(FILE *fout, message *msg) {
    fprintf(fout, "(type,<ts,wid>,value,req)\t(%s,<%d,%d>,%d,%d)\n",
            msg->type, (msg->tag).ts, (msg->tag).wid, msg->value, msg->reqNo);
} /* End of fprintfMsg()                                                */
/************************************************************************/

/* Fill in the fields of the message msg                                 */
void fillMsg(message *msg, message *smsg, char *type, int pid) {
    strcpy(msg->type, type);
    msg->pid = pid;
    (msg->tag).wid = pid;
    msg->reqNo += 1;

    if (!strcmp(msg->type, WRITE)) { /* If message is WRITE               */
        (msg->tag).ts = (smsg->tag).ts + 1; /* Increase ts               */
        srand(time(NULL));
        msg->value = rand() % RANGE;        /* Find random value         */
    }
} /* End of fillMsg()                                                   */
/************************************************************************/

/* Send Message to servers                                               */
void sndMsg(int sock, message *msg, char *buf, char *type) {
    int i;
    bzero(buf, strlen(buf));                /* Initialize buffer         */
    if (!strcmp(type, EXIT)) {              /* If msg is EXIT            */
        strcpy(buf, EXIT);                  /* -> save only typed        */
    }
    else                                    /* Else                      */
        msgToStr(buf, *msg);                /* ->save the whole msg      */
    if (write(sock, buf, BUFLEN) < 0) {     /* Send message              */
        perror("write");
        exit(EXIT_FAILURE);
    }
} /* End of sndMsg()                                                    */
/************************************************************************/

/* Transform string to the structure message                             */
void strToMsg(char *token, message *msg) {
    token = strtok(NULL, ",");
    msg->pid = atoi(token);
    token = strtok(NULL, ",");
    (msg->tag).ts = atoi(token);
    token = strtok(NULL, ",");
    (msg->tag).wid = atoi(token);
    token = strtok(NULL, ",");
    msg->value = atoi(token);
    token = strtok(NULL, ",");
    msg->reqNo = atoi(token);
} /* End of strToMsg()                                                  */
/************************************************************************/
/* Transform structure message to string                                 */
void msgToStr(char *buf, message msg) {
```

```
    sprintf(buf, "%s,%d,%d,%d,%d,%d\0", msg.type, msg.pid, (msg.tag).ts,
            (msg.tag).wid, msg.value, msg.reqNo);
} /* End of msgToStr()                                                    */
/**************************************************************************/
/* Check if the message received is valid
   (request number is the most recent)
   Return 1 if it is valid. Return 0 otherwise                           */
int checkValid(FILE *fout, message *smsg, message *msg) {
    if (smsg->reqNo == msg->reqNo) /* If most recent request             */
        return 1;
    printf("--- Ignore message - Request number is not the most recent\n");
    fprintf(fout, "--- Ignore message - Request number is not the most recent\n");
    return 0;
} /* End of checkValid()                                                  */
/**************************************************************************/
/* Proceed with sending the message                                      */
void sendProcd(FILE *out, char* type, int id, quorum *qrm, int qrmNo,
               int srvNo, message *wmsg, message *smsg, server *srvs) {
    int qrmCmp;
    int i;
    char buf[BUFLEN];   /* Buffer                                                  */
    float waitTime = 0; /* Random time to wait before sending a message   */
    time_t start, end;  /* Time: start and end                           */
    double dif = 0;     /* Difference of time                            */

    fillMsg(wmsg, smsg, type, id);
    qrmCmp = 0;
    i = 0;
    do { /* Do - while qrm is not complete                               */
        if (i != srvNo)
            /* Send msg to random server                                 */
            sendToRdmSrv(out, srvNo, buf, srvs, wmsg, &i);
        /* Wait random number between [0.3-0.7)                          */

        qrmCmp = rcvAckFromQrm(out, buf, srvs, srvNo, smsg, wmsg, qrmNo, qrm);
        if (qrmCmp != -1)       /* If a quorum is complete               */
            break;              /* Exit                                  */
    }
    while (qrmCmp == -1);

    findMaxTag(srvNo, srvs, wmsg, smsg, &(qrm[qrmCmp]), out);
    clrRand(srvNo, srvs);
    clrAck(srvNo, srvs);

} /* End of sendProcd()                                                   */
/**************************************************************************/

/* Check if Quorum File's format is correct and save data                *
 * Return 1 if correct, otherwise 0                                      */
int checkQrmFile(char *file, char * buf) {
    FILE *qin;      /* File descriptor                                   */
    int quor = 0;   /* Flag=1 if 'Q' appears in the quorum file          */
    int inter = 0;  /* Flag=1 if 'I' appears in the quorum file          */
    char *token;    /* Usage: To keep part of the line from the file     */
    int i = 0;      /* Counter                                           */

    qin = fopen(file, "r");             /* Open quorum file              */
    for (i = -1, fscanf(qin, "%s", buf); !feof(qin); fscanf(qin, "%s", buf)) {
        if (i == -1) {                  /* Ignore first line             */
            i++;
            continue;
        }
        /* If No Quorum Line & No Intersection Line                      */
        if ((!inter) && (!quor)) {
            if (buf[0] == 'Q' || buf[0] == 'q') {   /* If Quorum Line    */
                quor = 1;                           /* -> Set quor to 1  */
                continue;
            }
            else                                    /* Else              */
                return 0;                           /* ->ERROR in format*/
        }
        /* If Quorum Line & No Intersection Line                         */
        if (quor && !inter) {
```

```
            if (buf[0] == 'I' || buf[0] == 'i') {   /* If Intersec Line   */
                inter = 1;                          /* -> Set inter to 1  */
                continue;
            }
            else                                    /* Else               */
                return 0;                           /* ->ERROR in format  */
        }
        /* If Quorum Line & Intersection Line                             */
        if (quor && inter) {
            if (buf[0] == 'I' || buf[0] == 'i')     /* If Intersection Line*/
                continue;                           /* -> continue        */
            if (buf[0] == 'Q' || buf[0] == 'q')     /* If Quorum Line     */
                inter = 0;                          /* -> Set inter to 0  */
            else                                    /* Else               */
                return 0;                           /* ->ERROR in format  */
        }
    } /* End of for each line of file                                     */
    fclose(qin);
    return 1;
} /* End of checkQrmFile()                                                */
/***************************************************************************/

/* Based on a quorum file and a quorum id                                 *
   It returns how many servers belong to a spesific quorum              */
int returnServNum(char *file, char * buf, int cnt) {
    FILE *qin;      /* File descriptor                                    */
    char *token;    /* Usage: To keep part of the line from the file      */
    int quor = 0;   /* Flag = 1 if 'Q' appears in the quorum file         */
    int total = 0;  /* Total number of servers                            */
    int i = 0;      /* Counter                                            */

    qin = fopen(file, "r");             /* Open quorum file               */
    for (i = -1, fscanf(qin, "%s", buf); !feof(qin); fscanf(qin, "%s", buf)) {
        if (i == -1) {                  /* Ignore first line              */
            i++;
            continue;
        }
        token = strtok(buf, "(");       /* Parse line                     */
        if (token[0] == 'Q' || token[0] == 'q') /* If Quorum Line         */
            quor++;
        else
            continue;
        if (quor == cnt + 1)            /* If spesific Quorum Line        */
            while ((token = strtok(NULL, ",")) != NULL) { /*Parse rest of line*/
                total++;
                i++;
            } /* End of If spesific Quorum Line                           */
    } /* End of for each line from file                                   */
    fclose(qin);                        /* Close file                     */
    return total; /* Return number of servers                             */
} /* End of returnServNum()                                               */
/***************************************************************************/

/* Fill in the structure qrm with the data taken from the file            */
void fillInQrm(char *file, char *buf, quorum *qrm, int cnt) {
    FILE *qin;      /* File descriptor                                    */
    char *token;    /* Usage: To keep part of the line from the file      */
    int i, j = 0;   /* Counters                                           */
    int quor = 0;   /* Flag = 1 if 'Q' appears in the quorum file         */
    char temp;      /* Dummy variable                                     */
    int id;         /* Server's id                                        */

    qin = fopen(file, "r");             /* Open quorum file               */

    for (i = 0, fscanf(qin, "%s", buf); !feof(qin); fscanf(qin, "%s", buf)) {
        if (!i) {                       /* Ignore first line              */
            i++;
            continue;
        }
        token = strtok(buf, "(");       /* Parse line                     */
        if (token[0] == 'Q' || token[0] == 'q') /* If Quorum Line         */
            quor++;
```

```
        if (quor == cnt + 1) {          /* If spesific Quorum Line               */
            sscanf(token, "%c%d", &temp, &id);
            qrm->id = id;               /* Save quorum's id                      */
            j = 0;
            while ((token = strtok(NULL, ",")) != NULL) { /*Parse rest of line*/
                sscanf(token, "%d", &id);
                qrm->servers[j] = id;   /* Save quorum's server's id             */
                j++;
            }
            fclose(qin);                /* Close file                            */
            return;
        } /* End of if Quorum Line                                                */
    } /* End of for each line from file                                         */
} /* End of fillInQrm()                                                         */
/*************************************************************************/

/* Sends the message to a random server                                       */
void sendToRdmSrv(FILE *out, int srvNo, char *buf, server *srvs,
                  message *wmsg, int *i) {
    int num;                /* Random number representing the server         */
    do {    /* do - while srvRand[num]==1                                       */
        num = rand() % srvNo; /* Choose a random number between [0-srvNo) */
    }
    while (srvs[num].rand);
    srvs[num].rand = 1;
    (*i)++;

    /* Send msg to the random server found above                        *
       and wait random time  [0.3-0.7]                                  */
    sndMsg(srvs[num].sock, wmsg, buf, READ);
} /* End of sendToRdmSrv()                                                    */

/*************************************************************************/

/* Check for each server if there is an ACK and if there is save it           */
/* At the end of the loop check if there is a complete quorum                 */
int rcvAckFromQrm(FILE *out, char *buf, server *srvs, int srvNo, message *smsg,
                  message *wmsg, int qrmNo, quorum *qrm) {
    int i, j;                   /* Counters                                   */
    fd_set readfds;             /* Group of servers who have answered     */
    int res;                    /* Result from select()                       */
    struct timeval timeout;     /* Time for select() to wait                  */
    char *token;                /* String to parse the message                */

    timeout.tv_sec = 0;                 /* Time to wait                   */
    timeout.tv_usec = 0;
    FD_ZERO(&readfds);                  /* Clear descriptors set          */
    for (i = 0; i < srvNo; i++)
        FD_SET(srvs[i].sock, &readfds); /* Add fds of servers to set          */
    /* Select ready file descriptors (which have a message/answer)        */
    res = select(srvs[i - 1].sock + 1, &readfds, NULL, NULL, &timeout);
    if (res == -1) {
        perror("select()");
        for (j = 0; j < i; j++)
            close(srvs[j].sock);
        exit(EXIT_FAILURE);
    }

    for (i = 0; i < srvNo; i++) {               /* For each server            */
        if (FD_ISSET(srvs[i].sock, &readfds)) { /* If there's an answer   */
            /* -> Start ACK process                                           */
            /* Get a READACK message from a quorum                            */
            bzero(buf, strlen(buf));            /* Initialize buffer          */
            if (read(srvs[i].sock, buf, BUFLEN) < 0) { /* Read:blocking command*/
                perror("read()");
                for (j = 0; j <= i; j++)
                    close(srvs[j].sock);
                exit(EXIT_FAILURE);
            }
            token = strtok(buf, ",");           /* Parse message          */
            strcpy((srvs[i].msg).type, token);  /* Save type                  */
            strToMsg(token, &(srvs[i].msg));    /* Save server's message  */
            if (checkValid(out, &(srvs[i].msg), wmsg)){ /* If Most recent ACK*/
```

```
                srvs[i].ack=1;
            }
        } /* End of if FD_ISSET                                                */
    } /* End of for each quorum                                                */
    return checkQrmCmp(qrmNo, qrm, srvs);
} /* End of rcvAckFromQrm()                                                    */
/***************************************************************************/

/* If there are ACKs from a complete quorum then return number of quorum  */
int checkQrmCmp(int qrmNo, quorum *qrm, server *srvs) {
    int i = 0, j = 0;
    int flag = -1;

    for (i = 0; i < qrmNo; i++) {           /* For each quorum          */
        for (j = 0; j < qrm[i].servNum; j++) /* For each server of the quorum*/
            if(srvs[qrm[i].servers[j]-1].ack==1)/* If ack == 1          */
                flag = i;                   /* -> keep quorum id        */
            else {                          /* Else                     */
                flag = -1;                  /* -> return FALSE          */
                break;
            }
        if (flag != -1)
            return flag;
    }
    return -1;
} /* End of checkQrmCmp()                                                  */
/***************************************************************************/

/* Finds the maximum/most recent tag among the servers' tags of a quorum*/
void findMaxTag(int srvNo, server *srvs, message *wmsg, message *smsg,
                quorum *qrm, FILE *out) {
    int i;              /* Counter                                          */
    tag_type maxTag;    /* Max Tag                                          */
    int cmp;            /* Flag for comparing 2 tags                        */
    int pos = 0;        /* Flag - position of server with max Tag           */

    /* Initialise maxTag                                                    */
    maxTag.ts = 0;
    maxTag.wid = 0;

    for (i = 0; i < qrm->servNum; i++) {        /* For each server          */
        if ((srvs[qrm->servers[i] - 1].msg).reqNo == wmsg->reqNo)
            cmp = cmpTag((srvs[qrm->servers[i] - 1].msg).tag, maxTag);
            if (cmp == 1) {
                tagCpy(&maxTag, (srvs[qrm->servers[i] - 1].msg).tag);
                pos = qrm->servers[i] - 1;
            } /* End of inner if cmp==1                                     */
        } /* End of if reqNo equal                                          */
        strcpy(smsg->type, (srvs[pos].msg).type); //
        smsg->pid = (srvs[pos].msg).pid; //
        tagCpy(&smsg->tag, maxTag);
        smsg->value = (srvs[pos].msg).value;
        smsg->reqNo = (srvs[pos].msg).reqNo; //
    } // End of for each quorum                                             */
} /* End of findMaxTag()                                                   */
/***************************************************************************/

/* Based on two tags
   Rerun 1 if tag1 > tag2, 0 if tag1 == tag2, -1 if tag1 < tag2          */
int cmpTag(tag_type tag1, tag_type tag2) {
    if (tag1.ts > tag2.ts)                  /* If ts1 > ts2             */
        return 1;                           /* -> tag1 > tag2           */
    if (tag1.ts == tag2.ts)                 /* If ts1 == ts2            */
        if (tag1.wid > tag2.wid)            /*    If wid1 > wid2        */
            return 1;                       /*    -> tag1 > tag2        */
    if (tag1.ts == tag2.ts)                 /*    If ts1== ts2          */
        return 0;                           /*    -> tag1 = tag2        */
    return -1;                              /* -> tag1 < tag2           */
} /* End of cmpTag()                                                       */
/***************************************************************************/

/* Copy src tag into dest tag                                           */
void tagCpy(tag_type* dest, tag_type src) {
```

```
    dest->ts = src.ts;                          /* -> Save ts                   */
    dest->wid = src.wid;                        /* -> Save wid                  */
} /* End of msgCpy                                                              */
/*****************************************************************************/
/*                              END OF FILE                                    */
/*****************************************************************************/
```

```
/*************************************************************************
 * Name & Surname: Maria Stylianou       ID: 1012147                     *
 * Department: Computer Science of University of Cyprus                  *
 * Last Update: 30-MAR-2011                                              *
 * Title of program: freader.c                                           *
 *                                                                       *
 * Functionality:                                                        *
 * It has an active role in the system;                                  *
 * It reads a message, which the servers hold, in 1RTT or 2RTT,          *
 * depending on the Quorum Views:                                        *
 * qv1: All servers of the quorum have the maximum timestamp. - 1RTT     *
 * qv3: In at least one intersection of the quorum,                      *
        all servers have the maximum timestamp. - 2RTT (READ, WRITE)     *
 * qv2: If none of the above apply -                                     *
        it goes recursive till it endsup in onother qv                   */
/*************************************************************************
 *                            READER                                     *
 *************************************************************************/
/* LIBRARIES *************************************************************/
#include <stdio.h>              /* For I/O                                */
#include <stdlib.h>             /* For exit() malloc() free()             */
#include <string.h>             /* For strings - strcpy() strcmp() strcat()*/
#include <unistd.h>             /* For close()                            */
#include <sys/socket.h>  /* For sockets - socket() bind() listen()        *
                                  accept() connect() shutdown()          *
                                  close() setsockopt()                   */
#include <sys/types.h>   /* For sockets -  same as above                  */
#include <netinet/in.h>  /* For Internet sockets -                        *
                            address formating in a general structure     */
#include <netdb.h>              /* For getaddrinfo()                      */
#include <time.h>               /* For time()                             */
#include <sys/time.h>
#include <sys/select.h>  /* For select()                                  */
/* CONSTANTS *************************************************************/
#define BUFLEN 256              /* Buffer Length                          */
#define SIZE 10          /* Size of type of message                       */
#define RANGE 100               /* Generate a random number [0-RANGE]     */
#define ACK "ACK"               /* To be concatenated with the type of the *
                                   received message                       */
#define READ "READ"             /* READ message                           */
#define WRITE "WRITE"    /* WRITE message                                 */
#define EXIT "EXIT"             /* EXIT message for closing the connection*/
#define FILENAME_LEN 20  /* Length for filename                           */
#define MINOP 1          /* Minimum option on the menu                    */
#define MAXOP 2          /* Maximum option on the menu                    */
#define AUTO 6           /* Number of arguments for auto-procedure        */
#define MAN 4                   /* Number of argumetns for manual procedure*/
#define MILLION 1000000  /* One million - for math                        */
/* STRUCTURES ************************************************************/
/* Structure for tag                                                     */
typedef struct{
  int ts;                       /* Timestamp                              */
  int wid;                      /* Writer's id                            */
}tag_type;

/* Structure for message                                                 */
typedef struct{
  char type[SIZE];              /* Type of message: WRITE, READ           */
  int pid;                      /* Process id                             */
  tag_type tag;          /* Tag: timestamp and writer's id               */
  int value;             /* Object's value                               */
  int reqNo;                    /* Number of request                      */
}message;

/* Structure for server                                                  */
typedef struct{
  int id;                       /* Server's Id                            */
  int sock;                     /* Socket file descriptor                 */
  char hostname[RANGE];  /* Server's hostname                            */
  char ipaddr[RANGE];    /* Server's ip address                          */
  struct addrinfo *serv; /* Server                                       */
  int rand;                     /* Flag showing if it was chosen randomly */
  int ack;                      /* Flag showing if it sent ACK            */
```

```
  message msg;          /* Mesage                                          */
}server;

/* Structure for quorum                                                    */
typedef struct{
  int id;                     /* Quorum's id                               */
  int interId;          /* Intersected quorum's id                         */
  int servNum;          /* Number of servers                               */
  int *servers;         /* Table with Servers' names                 */
}quorum;
/* FUNCTIONS ************************************************************/
/* Print Functions                                                             */
void printMenu(int *option);
void execError(FILE *fin, FILE *qin, FILE *out);
/* Initialization Functions                                                    */
void initSrv(server *srvs, int srvNo);
void initIds(quorum *qrm, int len);
void initMsg(message *msg);
void clrRand(int size, server *srvs);
void clrAck(int size, server *srvs);
/* Message Functions                                                           */
void printfMsg(message *msg);
void fprintfMsg(FILE *out, message *msg);
int sendProcd(FILE *out, char* type, int id, quorum *qrm, int qrmNo,
              int srvNo, message *wmsg, message *smsg, server *srvs);
void fillMsg(message *msg, message *smsg, char *type, int pid);
void sndMsg(int sock, message *msg, char *buf,char *type);
void strToMsg(char *token, message *msg);
void msgToStr(char *buf, message msg);
int checkValid(FILE *fout, message *smsg, message *msg);
/* Quorum File Functions                                                   */
void readQrmFile();
int checkQrmFile(char *file, char * buf);
int returnIntsec(char *file, char * buf);
int returnServNum(char *file, char * buf, int cnt, char ltr1, char ltr2);
void fillInQrm(char *file, char *buf, quorum *qrm, int cnt, char ltr1,
              char ltr2);
/* Quorum Functions                                                            */
void sendToRdmSrv(FILE *out,int srvNo, char *buf, server *srvs,
                  message *wmsg, int *i);
int rcvAckFromQrm(FILE *out,char *buf, server *srvs, int srvNo, message *smsg,
                  message *wmsg, int qrmNo, quorum *qrm);
int checkQrmCmp(int qrmNo, quorum *qrm, server *srvs);
int findQuorumView(quorum qrm,  int qrmCmp, quorum *intsec, server *srvs,
                   int intsecNo, int srvNo, int reqNo, int option,
                   FILE *out, tag_type *maxTag, int *iter);
int findTotalSrvs(quorum qrm, int qrmCmp, server *srvs, tag_type maxTag,
                  int srvNo);
/* Tag Functions                                                           */
tag_type findMaxTag(int srvNo, server *srvs, message *wmsg, message *smsg,
                quorum *qrm);
int cmpTag(tag_type tag1, tag_type tag2);
void tagCpy(tag_type* dest, tag_type src);
/************************************************************************/
/*                              MAIN                                    */
/************************************************************************/
main (int argc, char *argv[]){
  /* Definitions and (some) Initializations                                */
  FILE* fin;            /* File descriptor of servers file                 */
  FILE* qin;            /* File descriptor of quorum system file           */
  FILE* out;            /* File descriptor of output file                  */
  int id;           /* Reader's id                                   */
  int srvNo=0;   /* Total number of servers                                */
  int qrmNo=0;   /* Total number of quorums                                */
  int intsecNo=0;       /* Total number of intersections                   */
  int port=0;           /* Server's port                                   */
  int i=0,j=0;   /* Counters                                               */
  char buf[BUFLEN];     /* Buffer                                          */
  int option=0;  /* User's option                                  */
  int ready=0;   /* Flag showing if writer is ready to make       *
                          a new request                                     */
  message wmsg,smsg;    /* Writer's and server's message                   */
  quorum *qrm;          /* Quorum System                                   */
```

```
quorum *intsec;        /* Intersections                                                */
server *srvs; /* Servers' information                                    */
tag_type maxTag;       /* Holds the maxTag found from a quorum                   */
int qrmCmp=0; /* Flag showing if there is a complete quorum             */
int qv;                /* Flag for quorum view                                   */
int opNo=0;            /* In auto-procedure - Number of operations               */
int opCnt=0;       * Counter of operations                                       */
float opFreq=0;        /* In auto-procedure - Frequency of operations            */
char *opFreqStr;       /* Frequency of operations converted to string            */
int cond=0;            /* Condition - automatic or manual procedure              */
int fast=0;            /* Counts the fast operations                             */
int slow=0;            /* Counts the slow operations                             */
float fastPrc=0;       /* Percentage of fast operations                          */
float slowPrc=0;       /* Percentage of slow operations                          */
int opTotal=0;/* Total number of operations                             */
struct timeval tim;
double start,end;      /* Time: start and end                                    */
double opLat=0.0;      /* Operation Lattency                                     */
int iter=0;            /* Number of iterations to find the final QV              */
char outFile[FILENAME_LEN]; /* Name of output log file                   */
int dummy;             /* Keep dummy numbers                                     */
int status;            /* Result from getaddrinfo()                              */
struct addrinfo hints;/* For getaddrinfo()                                       */
struct addrinfo *ipv4info;/* Point to info for ipv4                              */
struct addrinfo *p;    /* Counter                                                */
char prt[5];           /* Port converted to string                               */
char ipstr[INET6_ADDRSTRLEN];  /* IP converted to string                         */
void *addr;            /* Holds the address                                            */
char *ipver;           /* Message showing the Internet Protocol          */
/*************************************************************************/
/* Check if server's host name and port number are given                  */
if (argc!=MAN && argc!=AUTO){
   printf("\nUsage of file (if auto-procedure give parameters of parenthesis):\
  \n%s <readerID> <serversFile> <quorumFile> (<operationsNo> <operationsFreq>)\n",argv[0]);
  exit(EXIT_FAILURE);
}
id=atoi(argv[1]);
sprintf(outFile, "./log-fr%d.dat", id);

/* Open/Create file to write                                              */
if (!(out = fopen(outFile,"w"))){      /* Servers file                    */
  printf("freader: id=%d => ",id);
  perror("fopen()");
  exit(1);
}

printf("\nStart freader...\n");
fprintf(out,"Start freader...\n");
sleep(5);
printf("Analysing data...\n");
fprintf(out,"Analysing data...\n");
/* Check if automatic procedure and save new parameters                   */
if (argc==AUTO){
  opNo = atoi(argv[4]);
  opFreq = strtof(argv[5], &opFreqStr);
  printf("Total operations: %d\n", opNo);
  fprintf(out,"Total operations: %d\n", opNo);
  printf("Frequency of operations: %.2f\n", opFreq);
  fprintf(out,"Frequency of operations: %.2f\n", opFreq);
}
/* Open servers file & quorum file                                                */
if (!(fin = fopen(argv[2],"r"))){      /* Servers file                    */
  printf("freader: id=%d => ",id);
  perror("fopen()");
  exit(EXIT_FAILURE);
}
if (!(qin = fopen(argv[3],"r"))){      /* Quorums file                    */
  printf("freader: id=%d => ",id);
  perror("fopen()");
  exit(EXIT_FAILURE);
}

/* Find total number of servers & quorums                                 */
```

```
fscanf(fin,"%d",&srvNo);
fscanf(qin,"%d",&qrmNo);

/* Create dynamically 2 tables for: servers' message and quorum struc   */
qrm=(quorum *)malloc(qrmNo * sizeof(quorum));
srvs=(server *)malloc(srvNo * sizeof(server));

/* Initialize quorum ids, writer and servers message,tables with hosts  */
initIds(qrm, qrmNo);
initMsg(&wmsg);
initMsg(&smsg);
initSrv(srvs,srvNo);
for (i=0; i<srvNo; i++){
  strcpy(srvs[i].hostname, "localhost");
  strcpy(srvs[i].ipaddr, "127.0.0.1");
}
/* Clear flag tables                                                    */
clrRand(srvNo, srvs);
clrAck(srvNo, srvs);

/* Print Information                                                    */
printf("Total servers: %d\n",srvNo);
fprintf(out, "Total servers: %d\n",srvNo);
printf("Total quorums: %d\n",qrmNo);
fprintf(out, "Total quorums: %d\n",qrmNo);

/* Check quorum file's format                                           */
if (!checkQrmFile(argv[3], buf))
  execError(fin, qin, out);

intsecNo=returnIntsec(argv[3], buf);   /* Find total No of inters       */
printf("Total intersections: %d\n",intsecNo);
fprintf(out,"Total intersections: %d\n",intsecNo);
/* Create dynamically a table for intersections                    */
intsec=(quorum *)malloc(intsecNo * sizeof(quorum));
for (i=0; i<intsecNo; i++){            /* Fore each intersection        */
  /* Find total number of servers                                       */
  intsec[i].servNum=returnServNum(argv[3],buf,i,'I','i');
  /* Create dynamically a table of servers ids belonging to inters      */
  intsec[i].servers=(int *)malloc(intsec[i].servNum*sizeof(int));
  /* Fill in intersection's id and servers ids                          */
  fillInQrm(argv[3],buf,&intsec[i],i,'I','i');
}
/* Fill in Quorum System - For each quorum:                             */
for (i=0; i<qrmNo; i++){
  /* Save number of servers                                             */
  qrm[i].servNum=returnServNum(argv[3], buf, i, 'Q', 'q');
  /* Create dynamically a table of servers id belonging in quorum       */
  qrm[i].servers=(int *)malloc(qrm[i].servNum * sizeof(int));
  /* Fill in quorum's id and servers ids                                */
  fillInQrm(argv[3], buf, &qrm[i], i,'Q','q');
}

/* For each server                                                      */
for (i=0; i<srvNo; i++){
  /* Find server's id, port, hostname                                   */
  fscanf(fin,"%d %d %s %d",&(srvs[i].id), &port, srvs[i].hostname,&dummy)

  memset(&hints, 0, sizeof hints); /* Initialise struct hints to zero   */
  hints.ai_family = AF_UNSPEC;     /* Either IPv4 or IPv6               */
  hints.ai_socktype = SOCK_STREAM; /* TCP stream sockets                */
  sprintf(prt,"%d",port);

  /* Get server's information                                           */
  if ((status = getaddrinfo(srvs[i].hostname, prt, &hints, &srvs[i].serv))
      != 0) {
    perror("getaddrinfo()");
    fprintf(stderr, "getaddrinfo(): %s\n", gai_strerror(status));
    exit(1);
  }
  /* srvs[i].serv now points to a linked list of 1 or more struct addrinfos*/
  /* For each struct in the linked list                                                */
  for(p = srvs[i].serv; p != NULL; p = p->ai_next) {
```

```
      if (p->ai_family == AF_INET) { // IPv4
        struct sockaddr_in *ipv4 = (struct sockaddr_in *)p->ai_addr;
        addr = &(ipv4->sin_addr);
        ipver = "IPv4";
        ipv4info=p;
      }
      else { // IPv6
        struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *)p->ai_addr;
        addr = &(ipv6->sin6_addr);
        ipver = "IPv6";
      }

      /* Convert the IP to a string and print it                              */
      inet_ntop(p->ai_family, addr, ipstr, sizeof ipstr);
      printf("  %s: %s\n", ipver, ipstr);
      fprintf(out,"  %s: %s\n", ipver, ipstr);
      strcpy(srvs[i].ipaddr,ipstr);

      /* Request connection                                                   */
      if ((srvs[i].sock = socket(p->ai_family, p->ai_socktype,
                                 p->ai_protocol)) < 0){
        perror("socket()");
      }

      if (connect(srvs[i].sock, p->ai_addr,p->ai_addrlen) < 0){
        perror("connect()");
        close(srvs[i].sock);
        continue;
      }
      srvs[i].serv=p;
      printf("Requested connection established to host %s - port %d - socket %d\n",
             srvs[i].hostname, port, srvs[i].sock);
      fprintf(out,"Requested connection established to host %s - port %d - socket %d\n",
                  srvs[i].hostname, port, srvs[i].sock);
    }
  } /* End of for each server                                               */
  fclose(fin);           /* Close fd to servers file                        */
  fclose(qin);           /* Close fd to quorums file                        */
  ready=1;

  if (argc==AUTO)        /* If automatic procedure                          */
    cond=opNo+1; /* -> While number of operations+1 for exit              */
  else                   /* If manual procedure                             */
    cond=1;              /* -> Endless while                                */

  while (cond){
    if (argc==AUTO){     /* If automatic procedure                          */
      cond--;            /* -> Dicrease counter                             */
      option=1;
      if (!cond) /* extra round => time to exit                            */
        option=2;
      usleep(opFreq*MILLION);
    }
    else{
      printMenu(&option);/* Print Menu                                       */
    }
    switch(option){
      case 1:            /* Option 1: Read the object                       */
      /* Check to ensure that a request can happen if previous one
      has finished */
      if (ready==0){
        printf("Last read request hasn't finished\n");
        fprintf(out,"Last read request hasn't finished\n");
        cond++;                  /* Increade operations for not loosing                    the
round                            */
        continue;
      }
      ready=0;                   /* This request hasn't finished          */
      gettimeofday(&tim, NULL); /* Current Time - 1st                      */
      start=tim.tv_sec+(tim.tv_usec/1000000.0);
      opCnt++;
      printf("Operation No.: %d\n",opCnt);
      fprintf(out, "Operation No.: %d\n",opCnt);
```

```
      /* 1st RTT: Send a READ message to all servers                          */
      qrmCmp = sendProcd(out, READ, atoi(argv[1]), qrm, qrmNo, srvNo,
               &wmsg, &smsg, srvs);
      do{
        maxTag=findMaxTag(srvNo, srvs, &wmsg, &smsg, &(qrm[qrmCmp]));
        qv=findQuorumView(qrm[qrmCmp], qrmCmp, intsec, srvs, intsecNo, srvNo,
                          wmsg.reqNo, option, out, &maxTag, &iter);
        printf("*************************************************************\n");
        fprintf(out, "*************************************************************\n");
        switch(qv){
          case -1: printf("Detected: NaN, Something went wrong.\n");
          fprintf(out,"Detected: NaN, Something went wrong.\n");
        break;
        case 1:          /* Quorum View 1                                    */
          printf("Detected: QV1, Iterations: %d\n", iter);
          fast++;
          break;
        case 2:          /* Quorum View 2                                    */
          for (j=0; j<qrm[qrmCmp].servNum; j++){
            for (i=0; i<srvNo; i++){             /* For each server          */
              if (qrm[qrmCmp].servers[j]==srvs[i].id && (srvs[i].msg).reqNo!=0)
                if (((srvs[i].msg).tag).ts == maxTag.ts){
                  initMsg(&(srvs[i].msg));
                }// End of if ts equal
            }// End of for all servers
          }// End of for each server of the winner quorum
          break;
        case 3:          /* Quorum View 3                                   */
         fprintf(out, "Detected: QV3, Iterations: %d\n", iter);

          /* 2nd RTT: Send a WRITE message to all servers                    */
          qrmCmp = sendProcd(out, WRITE, atoi(argv[1]), qrm, qrmNo, srvNo,
                    &wmsg, &smsg, srvs);
          slow++;
          break;
       } /* End of switch(qv)                                                */
      }while(qv==2);
      ready=1;                    /* This request has now finished          */
      gettimeofday(&tim, NULL);  /* Current Time - 2nd                      */
      end=tim.tv_sec+(tim.tv_usec/1000000.0);
      opLat=end-start;
      printf("Operation Lattency = %.4f seconds\n", opLat);
      printf("*************************************************************\n");
      printf("*************************************************************\n");
      fprintf(out, "Operation Lattency = %.4f seconds\n", opLat);
      fprintf(out, "*************************************************************\n");
      fprintf(out, "*************************************************************\n");
      break;
       case 2:                    /* Option 3: Exit                          */
       for (i=0; i<srvNo; i++){
         sndMsg(srvs[i].sock,&wmsg,buf,EXIT);
       }
       opTotal=fast+slow;
       fastPrc=(float) fast/opTotal;
       slowPrc=(float) slow/opTotal;
       printf("Total number of operations = %d\n", opTotal);
       printf("Fast operations(percent) = %.2f / 100\n", fastPrc);
       printf("Slow operations(percent) = %.2f / 100\n", slowPrc);
       printf("Exiting...\n\n");
       fprintf(out, "Total number of operations = %d\n", opTotal);
       fprintf(out, "Fast operations(percent) = %.2f / 100\n", fastPrc);
       fprintf(out, "Slow operations(percent) = %.2f / 100\n", slowPrc);
       fprintf(out, "Exiting...\n\n");
       exit(EXIT_FAILURE);
       break;
     } /* End of switch(option)                                              */
   } /* End of while(cond)                                                   */
   fclose(out);
   return;
} /* End of main()                                                          */
/*************************************************************************/

/* Print Menu                                                              */
```

```
void printMenu(int *option){
  do{
    printf("\n");
    printf("  ********\n");
    printf("  * Menu *\n");
    printf("  ********\n");
    printf("1. Read an object\n");
    printf("2. Exit\n");
    printf("Option: ");
    scanf("%d", option);
    printf("\n");

  }while(*option<MINOP || *option>MAXOP);
} /* End of printMenu()                                                    */
/************************************************************************/
/* Print error message, close files and exit                             */
void execError(FILE *fin, FILE *qin, FILE *out){
  printf("Wrong format in the quorum system file.\n");
  printf("Please correct the file!\n\nExiting...\n\n");
  fprintf(out, "Wrong format in the quorum system file.\n");
  fprintf(out, "Please correct the file!\n\nExiting...\n\n");
  fclose(fin);
  fclose(qin);
  exit(EXIT_FAILURE);
} /* End of execError()                                                    */
/************************************************************************/
/* Initialize structure server                                           */
void initSrv(server *srvs, int srvNo){
  int i;
  for (i=0; i<srvNo; i++){
    srvs[i].id=0;
    srvs[i].rand=0;
    srvs[i].ack=0;
    initMsg(&srvs[i].msg);
  }
} /* End of initSrv()                                                     */
/************************************************************************/
/* Initialize table                                                      */
void initIds(quorum *qrm, int len){
  int i;
  for (i=0; i<len; i++)
    qrm[i].id=0;
} /* End of initIds()                                                     */
/************************************************************************/
/* Set all servers rand to 0 (means: they haven't been chosen)           */
void clrRand(int size, server *srvs){
  int i;
  for (i=0; i<size; i++)
    srvs[i].rand=0;
} /* End of clrRand()                                                     */
/************************************************************************/
/* Set all servers Ack to 0 (means: they haven't replied)                */
void clrAck(int size, server *srvs){
  int i;
  for (i=0; i<size; i++)
    srvs[i].ack=0;
} /* End of clrAck()                                                      */
/************************************************************************/
/* Initialize message                                                    */
void initMsg(message *msg){
  strcpy(msg->type,"\0");
  msg->pid=0;
  (msg->tag).ts=0;
  (msg->tag).wid=0;
  msg->value=-1;
  msg->reqNo=0;
} /* End of initMsg()                                                     */
/************************************************************************/
/* Print message contents                                                */
void printfMsg(message *msg){
  printf("(type,pid,<ts,wid>,value,req)\t(%s,%d,<%d,%d>,%d,%d)\n",
         msg->type,msg->pid,(msg->tag).ts,(msg->tag).wid,msg->value,msg->reqNo);
} /* End of printfMsg()                                                   */
```

```
/*****************************************************************/
/* Print message contents in a file                              */
void fprintfMsg(FILE *fout, message *msg){
 fprintf(fout, "(type,<ts,wid>,value,req)\t(%s,<%d,%d>,%d,%d)\n",
         msg->type,(msg->tag).ts,(msg->tag).wid,msg->value,msg->reqNo);
} /* End of fprintfMsg()                                         */
/*****************************************************************/
/* Fill in the fields of the message                             */
void fillMsg(message *msg, message *smsg, char *type, int pid){
  strcpy(msg->type,type);
  msg->reqNo+=1;
  msg->pid=pid;
  (msg->tag).ts=(smsg->tag).ts;
  (msg->tag).wid=(smsg->tag).wid;
  msg->value=smsg->value;
} /* End of fillMsg()                                            */
/*****************************************************************/
/* Send Message to servers                                                */
void sndMsg(int sock, message *msg, char *buf, char *type){
  int i;
  bzero(buf, strlen(buf));                    /* Initialize buffer        */
  if (!strcmp(type, EXIT)){                   /* If msg is EXIT                */
    strcpy(buf,EXIT);                  /* -> save only typed          */
  }
  else                                        /* Else                     */
    msgToStr(buf, *msg);               /* ->save the whole msg        */
  if(write(sock, buf, BUFLEN) < 0){           /* Send message             */
      perror ("write");
      exit(EXIT_FAILURE);
  }
} /* End of sndMsg()                                                  */
/*****************************************************************/
/* Transform string to the structure message                     */
void strToMsg(char *token, message *msg){
  token = strtok(NULL, ",");
  msg->pid = atoi(token);
  token = strtok(NULL, ",");
  (msg->tag).ts = atoi(token);
  token = strtok(NULL, ",");
  (msg->tag).wid = atoi(token);
  token = strtok(NULL, ",");
  msg->value = atoi(token);
  token = strtok(NULL, ",");
  msg->reqNo = atoi(token);
} /* End of strToMsg()                                           */
/*****************************************************************/
/* Transform structure message to string                         */
void msgToStr(char *buf, message msg){
  sprintf(buf,"%s,%d,%d,%d,%d,%d\0",msg.type,msg.pid,(msg.tag).ts,
          (msg.tag).wid,msg.value,msg.reqNo);
} /* End of msgToStr()                                           */
/*****************************************************************/
/* Check if the message received is valid
   (request number is the most recent)
   Return 1 if it is valid. Return 0 otherwise                   */
int checkValid(FILE *fout, message *smsg, message *msg) {
    if (smsg->reqNo == msg->reqNo) /* If most recent request     */
        return 1;
    printf("--- Ignore message - Request number is not the most recent\n");
    fprintf(fout, "--- Ignore message - Request number is not the most recent\n");
    return 0;
} /* End of checkValid()                                         */
/*****************************************************************/
/* Proceed with sending the message                              */
int sendProcd(FILE *out, char* type, int id, quorum *qrm, int qrmNo,
              int srvNo, message *wmsg, message *smsg, server *srvs){
  int qrmCmp;
  int i;
  char buf[BUFLEN];      /* Buffer                                      */

  fillMsg(wmsg,smsg,type,id);

  qrmCmp=0; i=0;
```

```
  do{ /* Do - while qrm is not complete                                    */
    if (i!=srvNo)
      /* Send msg to random server
        */
      sendToRdmSrv(out, srvNo,buf,srvs,wmsg,&i);
      qrmCmp=rcvAckFromQrm(out, buf, srvs, srvNo, smsg, wmsg, qrmNo, qrm);
      if (qrmCmp!=-1)           /* If a quorum is complete                 */
        break;                  /* Exit                                    */
    if (qrmCmp!=-1){            /* If a qrm is complete                    */

    }
  }while(qrmCmp==-1);

  clrRand(srvNo, srvs);
  clrAck(srvNo, srvs);
  return qrmCmp;
} /* End of sendProcd()                                                   */
/*************************************************************************/
/* Check if Quorum File's format is correct and save data                *
 * Return 1 if correct, otherwise 0                                      */
int checkQrmFile(char *file, char * buf){
  FILE *qin;     /* File descriptor                                              */
  int quor=0;    /* Flag=1 if 'Q' appears in the quorum file             */
  int inter=0;   /* Flag=1 if 'I' appears in the quorum file             */
  char *token;   /* Usage: To keep part of the line from the file        */
  int i=0;       /* Counter                                                      */

  qin = fopen(file,"r");         /* Open quorum file                     */
  for(i=-1,fscanf(qin,"%s", buf); !feof(qin); fscanf(qin,"%s",buf)){
    if (i==-1){                 /* Ignore first line                    */
      i++;
      continue;
    }
    if ((!inter) && (!quor)){ /*If No Intersection Line & No Quorum Line  */
      if (buf[0]=='Q' || buf[0]=='q'){   /* If Quorum Line               */
        quor=1;                          /* -> Set quor to 1             */
        continue;
      }
      else                               /* Else                         */
        return 0;                        /* -> ERROR in format           */
    }
    if (quor && !inter){ /* If Quorum Line & No Intersection Line        */
      if (buf[0]=='I' || buf[0]=='i'){   /* If Intersection Line         */
        inter=1;                /* -> Set inter to 1                    */
        continue;
      }
      else                               /* Else                         */
        return 0;                        /* -> ERROR in format           */
    }
    if (quor && inter){         /* If Quorum Line & Intersection Line    */
      if (buf[0]=='I' || buf[0]=='i')    /* If Intersection Line         */
        continue;                        /* -> continue                  */
      if (buf[0]=='Q' || buf[0]=='q')    /* If Quorum Line               */
        inter=0;                /* -> Set inter to 0                    */
      else                               /* Else                         */
        return 0;                        /* -> ERROR in format           */
    }
  } /* End of for each line of file                                       */
  fclose(qin);
  return 1;
}  /* End of checkQrmFile()                                                    */
/*************************************************************************/
/* It returns the total number of intersections                          */
int returnIntsec(char *file, char * buf){
  FILE *qin;                    /* File descriptor                              */
  int inter=0;          /* Total number of intersections                */

  qin = fopen(file,"r");
  for(fscanf(qin,"%s", buf); !feof(qin); fscanf(qin,"%s",buf)){
    if (buf[0]=='I' || buf[0]=='i')      // Quorum Line
      inter++;
  }
  return inter;
```

```
} /* End of returnIntsec()                                                    */
/*************************************************************************/
/* Based on a quorum file and a quorum id                                   *
   It returns how many servers belong to a spesific quorum                  */
int returnServNum(char *file, char * buf, int cnt, char ltr1, char ltr2){
  FILE *qin;     /* File descriptor                                         */
  char *token;   /* Usage: To keep part of the line from the file         */
  int quor=0;    /* Flag = 1 if 'Q' appears in the quorum file              */
  int total=0;   /* Total number of servers                                 */
  int i=0;       /* Counter                                                 */

  qin = fopen(file,"r");         /* Open quorum file                        */
  for(i=-1,fscanf(qin,"%s", buf); !feof(qin); fscanf(qin,"%s",buf)){
    if (i==-1){                          /* Ignore first line               */
      i++;
      continue;
    }
    token = strtok(buf, "(");               /* Parse line                 */
    if (token[0]==ltr1 || token[0]==ltr2)   /* Quorum Line                */
      quor++;
    else
      continue;
    if (quor==cnt+1)                     /* If spesific Quorum Line        */
      while((token = strtok(NULL, ","))!=NULL){ /*Parse the rest of line  */
        total++;
        i++;
      } /* End of If spesific Quorum Line                                  */
  } /* End of for each line from file                                      */
  fclose(qin);                           /* Close file                     */
  return total;                          /* Return number of servers       */
} /* End of returnServNum()                                                */
/*************************************************************************/
/* Fill in the structure qrm with the data taken from the file             */
void fillInQrm(char *file, char *buf, quorum *qrm, int cnt,
               char ltr1, char ltr2){
  FILE *qin;             /* File descriptor                                */
  char *token;   /* Usage: To keep part of the line from the file         */
  int i, j=0;            /* Counters                                       */
  int quor=0;            /* Flag = 1 if 'Q' appears in the quorum file     */
  char temp[10]; /* Dummy variable                                         */
  char d1,d2;            /* Dummy variables                                */


  qin = fopen(file,"r");         /* Open quorum file                        */
  for(fscanf(qin,"%s", buf); !feof(qin); fscanf(qin,"%s",buf)){
    if (!i){                             /* Ignore first line               */
      i++;
      continue;
    }
    sscanf(buf,"%c",&d1);
    if ((d1=='I') || (d1=='i')){ /* If Intersection Line                    */
      /* -> Save quorum's id and intersection's id                          */
      sscanf(buf,"%c%d%c%d%s",&d1,&(qrm->id),&d2,&(qrm->interId),buf);
    }
    else if ((d1=='Q') || (d1=='q')){    /* If Quorum Line                  */
      /* -> Save quorum's id                                                */
      sscanf(buf,"%c%d%c%s",&d1,&(qrm->id),&d2,buf);
    }
    if ((d1==ltr1) || (d1==ltr2))        /* If spesific Line                */
      quor++;

    if (quor==cnt+1){                    /* If spesific Line                */
      j=0;
      token = strtok(buf, "(");          /* Parse Line                      */
      sscanf(token,"%d",&(qrm->servers[j])); /* Save server's id          */
      token = strtok(buf, ",");

      while((token)!=NULL){              /* While token's not empty         */
        sscanf(token,"%d",&(qrm->servers[j])); /* Continue saving ids       */
        j++;
        token = strtok(NULL, ",");
      }
      fclose(qin);
```

```
      return;
    } /* End of if spesific line                                          */
  } /* End of for (EOF)                                                   */
} /* End of fillInQrm()                                                  */
/*********************************************************************/
/* Sends the message to a random server                                 */
void sendToRdmSrv(FILE *out, int srvNo, char *buf, server *srvs,
                  message *wmsg, int *i){
  int num;              /* Random number representing the server        */
  do{                   /* do - while srvRand[num]==1                   */
    num=rand()%srvNo;   /* Choose a random number between [0-srvNo)     */
  }while(srvs[num].rand);
  srvs[num].rand=1;     /* Set flag to 1                                */
  (*i)++;

  /* Send msg to the random server found above                          *
     and wait random time  [0.3-0.7]                                    */
  sndMsg(srvs[num].sock,wmsg,buf,READ);
} /* End of sendToRdmSrv()                                                  */
/*********************************************************************/
int rcvAckFromQrm(FILE *out,char *buf, server *srvs, int srvNo, message *smsg,
                  message *wmsg, int qrmNo, quorum *qrm){
   int i,j;                     /* Counters                             */
  fd_set readfds;               /* Group of servers who have answered   */
  int res;                      /* Result from select()                 */
  struct timeval timeout;       /* Time for select() to wait            */
  char *token;          /* String to parse the message                  */

  timeout.tv_sec = 0;                   /* Time to wait                 */
  timeout.tv_usec = 0;
  FD_ZERO(&readfds);                    /* Clear descriptors set        */
  for(i=0; i<srvNo; i++)
    FD_SET(srvs[i].sock,&readfds);      /* Add fds of servers to set    */
  res=select(srvs[i-1].sock+1,&readfds,NULL,NULL,&timeout);/*Select ready fd*/
  if(res==-1){
    perror("select()");
    for (j=0; j<i; j++)
      close(srvs[j].sock);
    exit(EXIT_FAILURE);
  }

  for (i=0; i<srvNo; i++){              /* For each server              */
    if (FD_ISSET(srvs[i].sock, &readfds)){/* If there's an answer      */
      /* -> Start ACK process                                           */
      /* Get a READACK message from a quorum                            */
      bzero(buf, strlen(buf));          /* Initialize buffer            */
      if(read(srvs[i].sock, buf, BUFLEN) < 0){ /* Read: blocking command */
        perror("read()");
        for (j=0; j<=i; j++)
          close(srvs[j].sock);
        exit(EXIT_FAILURE);
      }
      token = strtok(buf, ",");         /* Parse message                */
      strcpy((srvs[i].msg).type,token); /* Save type                    */
      strToMsg(token,&(srvs[i].msg));   /* Save server's message        */
      if (checkValid(out, &(srvs[i].msg), wmsg)){ /* If Most recent ACK */
          srvs[i].ack=1;                /* -> Set srvAck to 1           */
      }
    } /* End of if FD_ISSET                                             */
  }  /* End of for each server                                          */
  return checkQrmCmp(qrmNo,qrm, srvs);
} /* End of rcvAckFromQrm()                                             */
/*********************************************************************/

/* If there are ACKs from a complete quorum then return number of quorum */
int checkQrmCmp(int qrmNo, quorum *qrm, server *srvs){
  int i=0, j=0;
  int flag=-1;

  for (i=0; i<qrmNo; i++){              /* For each quorum              */
    for (j=0; j<qrm[i].servNum; j++){   /* For each server of the quorum */
      if(srvs[qrm[i].servers[j]-1].ack==1)/* If ack == 1                */
        flag=i;                         /* -> keep quorum id            */
```

```
      else{                              /* Else                                */
        flag=-1;                 /* -> return FALSE                             */
        break;
      }
    }
    if (flag!=-1)
      return flag;
  }
  return -1;
} /* End of checkQrmCmp()                                                       */
/*****************************************************************************/

/* Finds the maximum/most recent tag among the servers' tags of a quorum  */
tag_type findMaxTag(int srvNo, server *srvs, message *wmsg, message *smsg,
                quorum *qrm){
  int i,j;               /* Counter                                           */
  tag_type maxTag;       /* Max Tag                                           */
  int cmp;               /* Flag for comparing 2 tags                         */
  int pos=0;             /* Flag - position of server with max Tag            */

  /* Initialise maxTag                                                        */
  maxTag.ts=0;
  maxTag.wid=0;

  for (i=0; i<qrm->servNum; i++){               /* For each server            */
    for (j=0; j<srvNo; j++){
      if (qrm->servers[i] == srvs[j].id && (srvs[j].msg).reqNo!=0){
      cmp=cmpTag((srvs[j].msg).tag, maxTag);
        if (cmp==1){
          tagCpy(&maxTag, (srvs[j].msg).tag);
          pos=qrm->servers[i]-1;
        } /* End of inner if cmp==1                                           */
        strcpy(smsg->type, (srvs[pos].msg).type); //
        smsg->pid=(srvs[pos].msg).pid;            //
        tagCpy(&smsg->tag, maxTag);
        smsg->value=(srvs[pos].msg).value;
        smsg->reqNo=(srvs[pos].msg).reqNo;   //
      } /* End of if reqNo equal                                            */
    } /* End of for each quorum                                             */
  }
  return maxTag;
} /* End of findMaxTag()                                                    */
/*****************************************************************************/


/* Based on the complete quorum                                             */
/* It returns the quorum view                                               */
int findQuorumView(quorum qrm, int qrmCmp, quorum *intsec, server *srvs,
                   int intsecNo, int srvNo, int reqNo, int option,
                   FILE *out, tag_type *maxTag,int *iter){
  int i,j,k;             /* Pointers                                          */
  int cnt=0;             /* Counter                                           */
  int qv=-1;             /* Quorum View                                       */
  int size=0;            /* size of complete quorum                           */

  /* Calculate size of complete quorum                                        */
  for (i=0; i<qrm.servNum; i++){/* For each server from complete quorum   */
    for (j=i; j<srvNo; j++){
      if (srvs[j].id == qrm.servers[i] && (srvs[j].msg).reqNo!=0){
        size++;
        break;
      }
    }
  }
  (*iter)++;
  /* Check for Quorum View 1                                                  */
  cnt=findTotalSrvs(qrm, qrmCmp, srvs, *maxTag, srvNo);
  if (cnt == size)       /* If all servers of quorum have maxTag              */
    return 1;            /* -> Return quorum view 1                           */

  /* Check for Quorum View 3                                                  */
  for (k=0; k<intsecNo; k++){            /* For each intersection        */
    if (intsec[k].id == qrm.id){
```

```
      cnt=0;
      size=intsec[k].servNum;
      cnt=findTotalSrvs(intsec[k], qrmCmp, srvs, *maxTag, srvNo);
      if (cnt == size){
        return 3;
      }
    } /* End of if intsec found                                              */
   } /* End of for each intersection                                         */
    return 2;    /* Quorum View 2                                           */
} /* End of findQuorumView()                                                 */
/*************************************************************************/

int findTotalSrvs(quorum qrm, int qrmCmp, server *srvs, tag_type maxTag,
                  int srvNo){
  int i,j,k;             /* Counter                                        */
  int cnt=0;             /* Counter for servers having maxTag              */

  for (i=0; i<qrm.servNum; i++){/* For each server of complete quorum     */
    for (j=i; j<srvNo; j++){
      if (srvs[j].id==qrm.servers[i]
          && (srvs[j].msg).reqNo!=0){ /*If reqNo != 0                     */

          if (((srvs[j].msg).tag.ts == maxTag.ts) &&
              ((srvs[j].msg).tag.wid == maxTag.wid)){
            cnt++;
            break;
          } /* End of inner if ts and wid equal                           */
          else{
            cnt=-1;
            break;
          } /* End of else                                                */
      } /* End of if reqNo != 0                                          */
    } /* End of for each server from total servers                       */
  } /* End of for each server from qrm                                   */
    return cnt;
} /* End of findTotalSrvs()                                                */
/*************************************************************************/

/* Based on two tags
   Rerun 1 if tag1 > tag2, 0 if tag1 == tag2, -1 if tag1 < tag2          */
int cmpTag(tag_type tag1, tag_type tag2){
  if (tag1.ts > tag2.ts)               /* If ts1 > ts2                  */
    return 1;                                 /* -> tag1 > tag2          */
  if (tag1.ts == tag2.ts)              /* If ts1 == ts2                 */
    if (tag1.wid > tag2.wid)           /*    If wid1 > wid2             */
      return 1;                        /*    -> tag1 > tag2             */
    if (tag1.ts == tag2.ts)                   /*    If ts1== ts2        */
      return 0;                        /*    -> tag1 = tag2             */
  return -1;                                  /* -> tag1 < tag2          */
} /* End of cmpTag()                                                       */
/*************************************************************************/

/* Copy src tag into dest tag                                            */
void tagCpy(tag_type* dest, tag_type src){
  dest->ts = src.ts;                   /* -> Save ts                     */
  dest->wid = src.wid;          /* -> Save wid                          */
} /* End of msgCpy                                                       */
/*************************************************************************/
```

Παράρτημα Β

Σε αυτό το Παράρτημα δίνονται τα αρχεία εντολών (bash scripts) που χρησιμοποιήθηκαν για το χειρισμό των κόμβων στο PlanetLab. Συγκεκριμένα δίνονται τα εξείς:

1. connect.sh: γίνεται η σύνδεση με όλους τους κόμβους και ανήγει ένα terminal για τη κάθε σύνδεση.
2. uploadFiles.sh: για τη μεταφορά των αρχείων από το τοπικό υπολογιστή στους κόμβους.
3. downloadFiles.sh: για τη μεταφορά των logfiles από τους κόμβους στο τοπικό υπολογιστή.
4. runScr.sh: για την μεταφορά των scripts που περιέχουν την εντολή που θα εκτελέσει ο κάθε κόμβος.

```
#!/bin/bash
#connect.sh
#Connect with SSH from local computer with the slice given to all machines included in hostFile
#Show each connection in different terminal (tab)

if [ $# -ne 2 ]; then
  echo ""
  echo "ERROR - Usage: \"./connect.sh <slice> <hostFile>\""
  echo ""
else
  echo "Connect to servers"
  machines=$2
  slice=$1
  declare -a hostnames

  exec<$machines
  i=0
  while read line
  do
    hostnames[$i]=$line
    echo ${hostnames[$i]}
    i=$((i+1))
  done
  hostnames[$i]=$line;
  echo ${hostnames[$i]};
  srv=${#hostnames[*]}
  echo Total: $srv

  eval $(ssh-agent)
  ssh-add

  t=""
  for (( i = 0; $i < $srv; i++ ));
  do
    t+=" --tab -e 'bash -c \"echo ssh -l $slice -i ~/.ssh/id_rsa ${hostnames[$i]};ssh -l $slice
-i ~/.ssh/id_rsa ${hostnames[$i]};bash\"'"
  done
  VAR="/usr/bin/gnome-terminal"$t
  eval $VAR
fi
#!/bin/bash
#uploadFiles.sh
#Upload all files from SourceFileDir
#to home directory of all nodes included in hostFile

if [ $# -ne 3 ]; then
  echo ""
  echo "ERROR - Usage: \"./sendFiles.sh <slice> <hostFile> <SourceFileDir>\""
  echo ""
else
  echo "Send files to hosts"
  slice$1
  machines=$2
  dir=$3

  declare -a hostnames

  exec<$machines
  i=0
  while read line
```

```
  do
    hostnames[$i]=$line
    echo ${hostnames[$i]}
    i=$((i+1))
  done
  hostnames[$i]=$line;
  echo ${hostnames[$i]};
  srv=${#hostnames[*]}
  echo Total: $srv

  eval $(ssh-agent)
  ssh-add

  t=""
  for (( i = 0; $i < $srv; i++ ));
  do
    rsync -avz -e ssh $dir $slice@${hostnames[$i]}:~/
  done
  ssh-agent -k
  mplayer File_Transfered.mp3
fi
```

```
#!/bin/bash
#downloadFiles.sh
#Download log files from nodes included in hostFile
#To DestinationFileDir

if [ $# -ne 2 ]; then
  echo ""
  echo "ERROR - Usage: \"./rcvFiles.sh <hostFile> <DestinationFileDir>\""
  echo ""
else
  echo "Receive files from hosts"
  machines=$1
  dir=$2

  slice="cyprus_CFW"
  declare -a hostnames

  exec<$machines
  i=0
  while read line
  do
    hostnames[$i]=$line
    echo ${hostnames[$i]}
    i=$((i+1))
  done
  hostnames[$i]=$line;
  echo ${hostnames[$i]};
  srv=${#hostnames[*]}
  echo Total: $srv

  eval $(ssh-agent)
  ssh-add

  t=""
  for (( i = 0; $i < $srv; i++ ));
  do
    rsync -avz -e ssh $slice@${hostnames[$i]}:~/log* $dir
  done
  ssh-agent -k
  mplayer File_Transfered.mp3
fi
```

```
#!/bin/bash
#uploadScr.sh
#Check if all arguments were given
if [ $# -ne 11 ]; then
  echo ""
  echo "ERROR - Usage: \"./uploadSrv.sh <slice> <hostFile> <AlgType> <quorSysFile> <serversNo>
<portsNum> <failFreq> <wrtNo> <wOpFreq> <rdrNo> <rOpFreq>"
  echo " <failFreq> = [0,101]"
  echo ""
  echo "Example: ./uploadScr.sh uconn_exp hostsUconn.txt 1 25 qs-c25.txt 10001 80"
  echo ""
  # We have 11 arguments
else
  echo "Upload servers file"
  # Save arguments
  slice=$1              # Slice
  machines=$2           # Hosts File
  algType=$3            # Algorithm Type
  qsFile=$4             # Quorum System File
  srv=$5                # Number of servers
  ports=$6              # Beginning of Ports number
  fail=$7               # Fail Frequency
  wrt=$8        # Number of writers
  wrtFr=$9              # Frequency of operations for writers
  shift 2
  rdr=$8        # Number of readers
  rdrFr=$9              # Frequency of operationf for readers
  opNo=200
  ###############################################################################
  # Choose algorithm to run
  if [ $algType -eq 0 ]; then
    echo "Simple Algorithm"
    server="sserver"
    writer="swriter"
    reader="sreader"
  else
    echo "Classic Algorithm (fast)"
    server="fserver"
    writer="fwriter"
    reader="freader"
  fi
  ###############################################################################
  # Fill servers file
  name=(`echo $qsFile | tr '-' ' ' | cut -d ' ' -f 2`)
  echo $srv > srv-$name
  exec<$machines
  for (( i = 0; $i < $srv; i++ ));
  do
    read line
    if [ $((i%10)) -eq 0 ]; then
        exec<$machines
        read line
    fi
    host=$(echo $line);
    echo "$((i+1)) $((ports+i)) $host $fail" >> srv-$name
  done
  ###############################################################################
  # Start session - Give password
  eval $(ssh-agent)
  ssh-add
  ###############################################################################
  # Save nodes for servers
  declare -a hostnames

  exec<srv-$name
  read line     #ignore first line
  for (( i = 0; $i < $srv; i++ ));
  do
    read line
    hostnames[$i]=$(echo $line | cut -d " " -f3);
    echo "hostnames[$i] = ${hostnames[$i]}"
  done
  ###############################################################################
```

```
# Save rest of nodes
# Read machines file, skip the first x hosts which are used for servers (1<x<11)
exec<$machines
servers=10
i=0

if [ $srv -lt $servers ]; then
  servers=$srv;
fi

while read line
do
  i=$((i+1))
  if [ $i -le $servers ]; then
      continue;
  fi
  hostnames[$((srv-servers+i-1))]=$(echo $line);
  echo "-hostnames[$((srv-servers+i-1))] = ${hostnames[$((srv-servers+i-1))]}"
done
#############################################################################
# Upload scripts for Servers
j=0
t=""
for (( i = 0; $i < $srv; i++ ));
do
  echo "./$server $((i+1)) $((ports+i)) $fail" > "run-h$((i+1)).sh"
  chmod 711 "run-h$((i+1)).sh"
  rsync -avz -e ssh ./run-h$((i+1)).sh $slice@${hostnames[$j]}:~/
  t+=" --tab -e 'bash -c \"echo ssh -l $slice -i ~/.ssh/id_rsa ${hostnames[$i]};\
     ssh -l $slice -i ~/.ssh/id_rsa ${hostnames[$i]};\bash\"'"
  j=$((j+1))
done
VAR="/usr/bin/gnome-terminal"$t
eval $VAR
#############################################################################
# Upload file to all client hosts
exec<$machines
servers=10
i=0

if [ $srv -lt $servers ]; then
  servers=$srv;
fi

while read line
do
  i=`expr $i + 1`;

  if [ $i -le $servers ]; then
      continue;
  fi
  host=$(echo $line);
  rsync -avz -e ssh ./srv-$name $slice@$host:~/
done
#############################################################################
# Upload scripts for Writers

wnodes=`expr $nodes - $servers`;
if [ $wrt -lt $wnodes ]; then
  wnodes=$wrt;
fi
j=$srv
t=""
for (( i = $srv; $i < $((srv+wrt)); i++ ));
do
  echo "./$writer $((i+1)) srv-$name $qsFile $opNo $wrtFr" > "run-w$((i+1)).sh"
  chmod 711 "run-w$((i+1)).sh"
  if [ $((j%35)) -eq 0 ]; then
      j=$srv
  fi
  rsync -avz -e ssh ./run-w$((i+1)).sh $slice@${hostnames[$j]}:~/
  t+=" --tab -e 'bash -c \"echo ssh -l $slice -i ~/.ssh/id_rsa ${hostnames[$j]};\
     ssh -l $slice -i ~/.ssh/id_rsa ${hostnames[$j]};\bash\"'"
```

```
    j=$((j+1))
  done
  VAR="/usr/bin/gnome-terminal"$t
  eval $VAR
  ##########################################################################
  # Upload scripts for Readers
  rnodes=`expr $nodes - $servers`;
  if [ $rdr -lt $rnodes ]; then
    rnodes=$rdr;
  fi
  j=$srv
  t=""
  for (( i = $((srv+wrt)); $i < $((srv+wrt+rdr)); i++ ));
  do
    echo "./$reader $((i+1)) srv-$name $qsFile $opNo $rdrFr" > "run-r$((i+1)).sh"
    chmod 711 "run-r$((i+1)).sh"
    if [ $((j%35)) -eq 0 ]; then
        j=$srv
    fi
    rsync -avz -e ssh ./run-r$((i+1)).sh $slice@${hostnames[$j]}:~/
    t+=" --tab -e 'bash -c \"echo ssh -l $slice -i ~/.ssh/id_rsa ${hostnames[$j]};\
       ssh -l $slice -i ~/.ssh/id_rsa ${hostnames[$j]};\bash\"'"
    j=$((j+1))
  done
  VAR="/usr/bin/gnome-terminal"$t
  eval $VAR
  ##########################################################################
fi
```

Παράρτημα Γ

```
#!/bin/bash
#parseLogfile.sh
#Parse the logfile situated in logsDirectory, depending on the ProgramIdentifier
#which can be a “w” for a writer’s logfile or a “r” for a reader’s logfile

if [ $# -ne 2 ]; then
  echo ""
  echo "ERROR - Usage: \"./parseLogfile.sh <logsDirectory> <ProgramIdentifier>\""
  echo "<ProgramIdentifier>: w for writer, r for reader"
  echo ""
else
  echo "Collect data from readers"
  #Save Parameters
  dir=$1
  id=$2
  declare -a lat
  declare -a fast
  declare -a slow
  latAvg=0.0
  fastAvg=0.0
  slowAvg=0.0
  i=0

  cd $dir
  total=`ls *$id* | wc -w`;
  echo total=$total

  for file in `ls *$id*`;
  do
    echo $file;
    lat[$i]=`cat $file | grep Lattency | cut -d " " -f4 | awk 'BEGIN {sum=0; counter=0;} { sum
+= $0; counter++;} END {print sum/counter}'`;
    echo "lattency = ${lat[$i]}";
    latAvg=`echo $latAvg + ${lat[$i]} | bc`;

    if [ $id == 'r' ];then
        fast[$i]=`cat $file | grep Fast | cut -d " " -f4`;
        slow[$i]=`cat $file | grep Slow | cut -d " " -f4`;
        echo "fast     = ${fast[$i]}";
        echo "slow     = ${slow[$i]}";
        fastAvg=`echo $fastAvg + ${fast[$i]} | bc`;
        slowAvg=`echo $slowAvg + ${slow[$i]} | bc`;
    fi
    ((i=i+1));
  done

  latAvg=`echo $latAvg / $total | bc -l`;
  echo "lattency-average = $latAvg";

  if [ $id == 'r' ];then
    fastAvg=`echo $fastAvg / $total | bc -l`;
    slowAvg=`echo $slowAvg / $total | bc -l`;
    echo "fast-average     = $fastAvg";
    echo "slow-average     = $slowAvg";
  fi
fi
```