Ατομική Διπλωματική Εργασία

ΥΛΟΠΟΙΗΣΗ ΚΑΤΑΝΕΜΗΜΕΝΟΥ ΣΥΣΤΗΜΑΤΟΣ ΑΡΧΕΙΩΝ ΒΑΣΙΣΜΕΝΟ ΣΕ ΑΛΓΟΡΙΘΜΟΥΣ ΓΡΑΦΗΣ ΚΑΙ ΑΝΑΓΝΩΣΗΣ ΑΤΟΜΙΚΩΝ ΑΝΤΙΚΕΙΜΕΝΩΝ

Ηλίας Σπανός

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2016

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΥΛΟΠΟΙΗΣΗ ΚΑΤΑΝΕΜΗΜΕΝΟΥ ΣΥΣΤΗΜΑΤΟΣ ΑΡΧΕΙΩΝ ΒΑΣΙΣΜΕΝΟ ΣΕ ΑΛΓΟΡΙΘΜΟΥΣ ΓΡΑΦΗΣ ΚΑΙ ΑΝΑΓΝΩΣΗΣ ΑΤΟΜΙΚΩΝ ΑΝΤΙΚΕΙΜΕΝΩΝ

Ηλίας Σπανός

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 2016

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή της Ατομικής Διπλωματικής Εργασίας Δρ. Χρύση Γεωργίου, ο οποίος υπήρξε ο επιβλέπων καθηγητής της διπλωματικής μου εργασίας και για τη συνεχή υποστήριξη και καθοδήγηση που μου παρείχε κατά τη διάρκεια της εκπόνησης της.

Θα ήθελα, επίσης, να ευχαριστήσω τον Δρ. Νικόλα Νικολάου για τον χρόνο που αφιέρωσε για να μου δώσει πολύτιμες συμβουλές έτσι ώστε να ολοκληρώσω τη υλοποίηση μου επιτυχώς.

Επίσης, θα ήθελα να ευχαριστήσω την οικογένεια μου , και ιδιαίτερα τους γονείς μου , για τις συμβουλές και τις καθοδηγήσεις.

Περίληψη

Τα κατανεμημένα συστήματα στις μέρες μας είναι πολύ σημαντικά. Πολλές υφιστάμενες εφαρμογές όπως υπηρεσίες διαδικτύου και peer-to-peer εφαρμογές χρησιμοποιούν την έννοια των κατανεμημένων συστημάτων. Στη παρούσα Διπλωματική Εργασία, έχει γίνει η υλοποίηση ενός κατανεμημένου συστήματος αρχείων το οποίο είναι βασισμένο στους αλγορίθμους ABD [1] και LDR [5,6]. Η κύρια ιδέα του αλγορίθμου LDR είναι να διατηρεί τα πραγματικά δεδομένα σε ξεχωριστό χώρο σε σχέση με τις πληροφορίες που αφορούν την τοποθεσία των πραγματικών δεδομένων. Έτσι εκμεταλλεύεται το γεγονός αυτό για να εκτελεί τις περισσότερες λειτουργίες στα μεταδεδομένα παρά στα πραγματικά δεδομένα. Με αυτό τον τρόπο μειώνεται το κόστος επικοινωνίας και το σύστημα μας γίνεται πολύ πιο αποδοτικό. Επίσης, ένα σημαντικό πλεονέκτημα του αλγορίθμου LDR είναι ότι προσφέρει αυστηρή συνέπεια δεδομένων. Συνεπώς, οι χρήστες του συστήματος θα διαβάζουν πάντα το πιο πρόσφατο αρχείο. Επίσης, έχει υλοποιηθεί ο διαχειριστής αρχείων, ο οποίος είναι υπεύθυνος να δημιουργεί μοναδικά αναγνωριστικά για κάθε καινούργιο αρχείο που δημιουργείται στο σύστημα. Παράλληλα γνωρίζει ποια είναι τα διαθέσιμα αρχεία στο σύστημα και από ποιους χρήστες έχουν δημιουργηθεί. Επιπλέον, έχει υλοποιηθεί η διεπαφή συστήματος έτσι ώστε να προσφέρει ευκολότερη πρόσβαση στους χρήστες του συστήματος. Η υλοποίηση του ΚΣΑ είναι κατάλληλη τόσο σε LAN αλλά και σε WAN δίκτυα και δίνει τη δυνατότητα στους χρήστες να διαβάζουν ή να γράφουν οποιοδήποτε τύπο αρχείου.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός και Κίνητρο

Ένα κατανεμημένο σύστημα αρχείων (Κ.Σ.Α) [14,19] προσφέρει στους χρήστες του συστήματος διάφορες υπηρεσίες οι οποίες καθιστούν δυνατή την αποθήκευση, οργάνωση, διαχείριση αρχείων και ανάκτηση τους από το λειτουργικό σύστημα του υπολογιστή. Το σύστημα εμφανίζεται στους χρήστες του σαν ένα ενιαίο σύστημα και ο αποθηκευτικός χώρος μπορεί να βρίσκεται τοπικά ή σε απομακρυσμένες περιοχές.

Παρόλο που υπάρχουν ποικίλες υλοποιήσεις κατανεμημένων συστημάτων αρχείων, κανένα τέτοιο σύστημα μέχρι στιγμής δεν υποστηρίζει ισχυρή συνέπεια δεδομένων. Μια ενδιαφέρουσα ερευνητική περιοχή ασχολείται με την ισχυρή συνέπεια δεδομένων σε τέτοια συστήματα. Πιο συγκεκριμένα, το πεδίο έρευνας της συγκεκριμένης περιοχής είναι να διασφαλίζει ισχυρή συνέπεια δεδομένων σε κατανεμημένα συστήματα αποθήκευσης. Παρόλο που υπάρχουν αλγόριθμοι που παρέχουν ισχυρή συνέπεια δεδομένων και αυστηρότυπα επιβεβαιωμένες εγγυήσεις, μέχρι στιγμής δεν έχουν οδηγήσει στην υλοποίηση ενός Κ.Σ.Α με ισχυρή συνέπεια δεδομένων. Σκοπός της παρούσα Διπλωματική Εργασίας είναι η υλοποίηση ενός Κ.Σ.Α το οποίο εγγυάται ισχυρή συνέπεια δεδομένων με αυστηρότυπα επιβεβαιωμένες εγγυήσεις και είναι βασισμένο σε αλγορίθμους που προτάθηκαν στα πλαίσια της μελέτης όπου είναι βασισμένοι σε λειτουργίες γραφής/ανάγνωσης κατανεμημένων ατομικών αντικειμένων [1,5].

1.2 Μεθοδολογία Υλοποίησης

Η μεθοδολογία που ακολουθήθηκε για την υλοποίηση της διπλωματικής εργασίας είναι η εξής: Αρχικά, έγινε μελέτη άρθρων και εργασιών σχετικά με τους αλγόριθμους ABD [1] και LDR [5,6] που αποτελούν τη βάση της εργασίας μου. Επιπλέον, για απόκτηση θεωρητικού υπόβαθρου έγινε περεταίρω μελέτη σχετικά με τα κατανεμημένα συστήματα , κατανεμημένο υπολογισμό και κατανεμημένη μνήμη. Ακολούθως, έγινε εξοικείωση με προγραμματισμό υποδοχών για την επικοινωνία μεταξύ διαδικτυακών διεργασιών. Επίσης, έγινε μελέτη και εξοικείωση νημάτων τα οποία έπαιξαν σημαντικό ρόλο στην υλοποίηση του συστήματος αρχείων. Έπειτα, έγινε η υλοποίηση των αλγορίθμων ABD και LDR. Έτσι άρχισε να υλοποιείται το κατανεμημένο σύστημα αρχείων έχοντας ως βάση τους δυο προαναφερθέντες αλγόριθμους. Επομένως, χρειάστηκε να μελετηθεί ο τρόπος που θα μπορούσαμε να μεταφέρουμε μεγάλα αρχεία αποδοτικά, έτσι μελετήθηκαν διάφορες συναρτήσεις της γλώσσας προγραμματισμού C.

Η υλοποίηση του κατανεμημένου συστήματος αρχείων έγινε σε γλώσσα προγραμματισμού C κάνοντας χρήση προγραμματισμού υποδοχών και νημάτων. Το επόμενο βήμα ήταν η υλοποίηση του διαχειριστή αρχείων ο οποίος είναι υπεύθυνος να αποθηκεύει ποια αρχεία υπάρχουν στο σύστημα και ποιοι χρήστες τα έχουν δημιουργήσει. Επίσης, σχεδιάστηκε και αναπτύχθηκε η διεπαφή για εύκολη χρήση του συστήματος από χρήστες που δεν είναι απαραίτητα εξοικειωμένοι με κατανεμημένα συστήματα. Η υλοποίηση και η αποσφαλμάτωση έγινε τόσο στο τοπικό δίκτυο του Πανεπιστημίου αλλά και σε ευρύ κλίμακα.

1.3 Οργάνωση Διπλωματικής Εργασίας

Το έγγραφο της ατομικής διπλωματικής μου εργασίας αποτελείται από 7 Κεφάλαια. Στο Κεφάλαιο 1 δίνεται ο σκοπός, τα κίνητρα και η μεθοδολογία της εργασίας μας. Στο Κεφάλαιο 2 περιέχονται κάποιες βασικές έννοιες, ενώ στο Κεφάλαιο 3 γίνεται περιγραφή των βασικών αλγορίθμων ABD [1] και LDR [6,7] του ΚΣΑ που έχουμε υλοποιήσει. Ακολούθως, στο Κεφάλαιο 4 γίνεται αναλυτική περιγραφή της υλοποίησης του ΚΣΑ. Έπειτα, στο Κεφάλαιο 5 γίνεται μια ανασκόπηση σχετικά με την χρήση του συστήματος. Στο Κεφάλαιο 6 περιγράφεται η πειραματική αξιολόγηση του συστήματος και στο τελευταίο κεφάλαιο αναφέρονται τα συμπεράσματα, προβλήματα και παραδοχές καθώς επίσης και κάποιες μελλοντικές εργασίες που μπορούν να πραγματοποιηθούν.

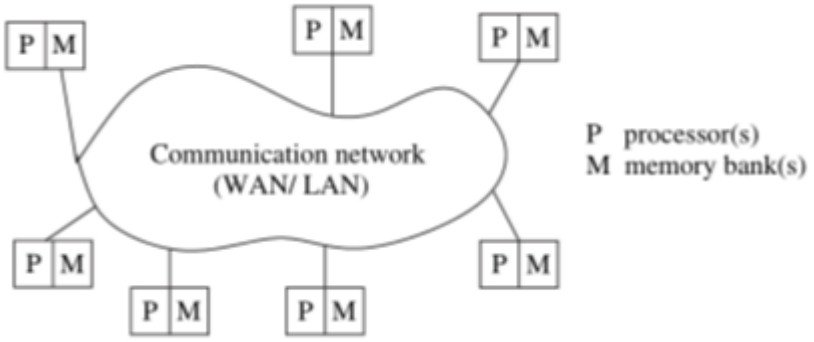
Κεφάλαιο 2

Υπόβαθρο

2.1 Κατανεμημένα Συστήματα

Ένα κατανεμημένο σύστημα [2,19] είναι μια συλλογή αυτόνομων επεξεργαστών οι οποίοι συνεργάζονται για την επίλυση ενός προβλήματος. Η επικοινωνία γίνεται αποκλειστικά μέσω ενός δικτύου, ανταλλάζοντας μηνύματα. Οι επεξεργαστές μπορεί να είναι ασύγχρονοι , να βρίσκονται σε γεωγραφική απομακρυσμένη απόσταση και δεν μοιράζονται κοινή μνήμη. Επίσης, παρουσιάζονται στους χρήστες ως μια μοναδική οντότητα. Ένα απλό κατανεμημένο σύστημα φαίνεται στο Σχήμα 2.1. Όπως φαίνεται και ένα δίκτυο επικοινωνίας ενώνει όλους τους κόμβους και ο κάθε κόμβος έχει τη δική του μνήμη και επεξεργαστή.

Τα κατανεμημένα συστήματα παρέχουν αρκετά πλεονεκτήματα σε σύγκριση με άλλα συστήματα γι’ αυτό και είναι τόσο δημοφιλή. Για παράδειγμα, οι υπολογισμοί κατανέμονται μεταξύ αρκετών κόμβων με αποτέλεσμα η απόδοση να αυξάνεται. Επιπλέον, υπάρχουν αρκετά αντίγραφα των δεδομένων και σε περίπτωση σφάλματος το σύστημα θα συνεχίσει να λειτουργεί κανονικά. Τέλος, ακόμα ένας σημαντικός παράγοντας είναι η επεκτασιμότητα. Καθώς αυξάνεται ο αριθμός των κόμβων του συστήματος η απόδοση αυξάνεται.

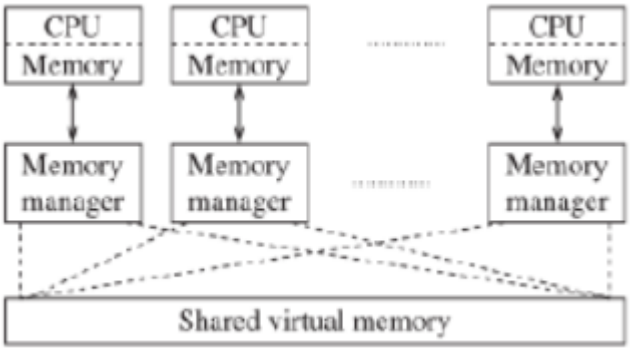


Σχήμα 2.1

Κατανεμημένο Σύστημα [19]

2.2 Κατανεμημένη Κοινόχρηστη Μνήμη

Η κατανεμημένη κοινόχρηστη μνήμη (Distributed Shared Memory) [1,2] είναι μια αφηρημένη έννοια στα κατανεμημένα συστήματα. Δίνει την αίσθηση μιας μοναδικής μνήμης με ένα χώρο διευθύνσεων. Στην πραγματικότητα, όπως φαίνεται και στο Σχήμα 2.2, ο κάθε κόμβος προσφέρει ένα μέρος της μνήμης του για την κοινή μνήμη η οποία απεικονίζεται ως μια κοινή εικονική μνήμη ενός χώρου διευθύνσεων. Με άλλα λόγια, δημιουργείται ένας μεγάλος εικονικός χώρος διευθύνσεων από τις εικονικές μνήμες του κάθε κόμβου. Κάθε κόμβος έχει πρόσβαση στην δική του προσωπική μνήμη και στην κοινή μνήμη διαμέσου του δικτύου με την βοήθεια των συναρτήσεων read και write.



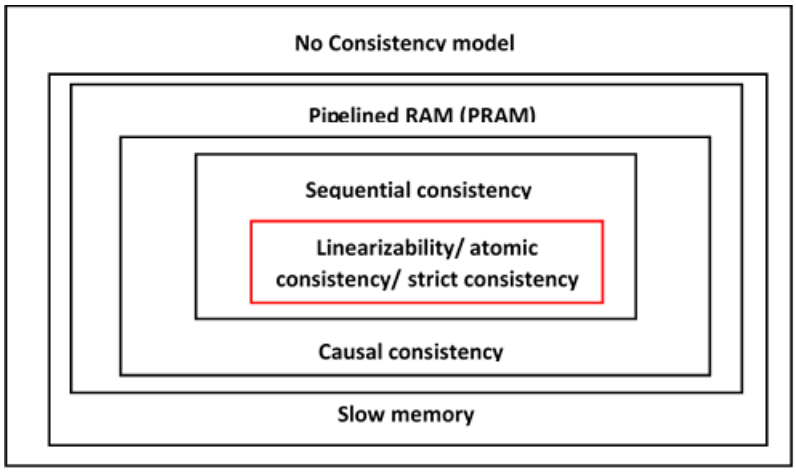
Σχήμα 2.2

Απεικόνιση της κοινής κατανεμημένης μνήμης [1,2]

2.3 Συνέπεια Μνήμης

Συνέπεια μνήμης, είναι η ικανότητα του συστήματος να εκτελεί ορθά λειτουργίες μνήμης. Έστω ότι έχουμε *N* διεργασίες όπου η κάθε διεργασία μπορεί να εκτελεί διαφορετικές λειτουργίες στην μνήμη. Συνεπώς, ο αριθμός των διαφορετικών συνδυασμών εκτέλεσης των λειτουργιών είναι πολύ μεγάλος. Για να υπάρξει συνέπεια μνήμης πρέπει να αφαιρεθούν οι συνδυασμοί που την παραβιάζουν. Με άλλα λόγια πρέπει να ορίσουμε τη ορθότητα έτσι ώστε να γνωρίζουμε ποιοι συνδυασμοί την παραβιάζουν. Ένας κατάλληλος ορισμός της ορθότητας θα όριζε ότι κάθε λειτουργία ανάγνωσης επιστρέφει την τιμή που αποθηκεύτηκε στην πιο πρόσφατη λειτουργία εγγραφής. Εντούτοις η έννοια του πιο πρόσφατου καθίσταται δύσκολο να οριστεί με την παρουσία ταυτόχρονων προσβάσεων στην μνήμη και πολλαπλών αντιγράφων του αντικειμένου. Επομένως, πρέπει να προσαρμοστεί ο ορισμός της ορθότητας και στόχος είναι να απορρίπτονται κάποιες ταυτόχρονες προσβάσεις, χωρίς όμως να μειώνουμε υπερβολικά τους επιτρεπτούς συνδυασμούς.

Το Σχήμα 2.3 απεικονίζει τα 6 διαφορετικά μοντέλα συνέπειας μνήμης. Στη παρούσα διπλωματική εργασία θα ασχοληθούμε μόνο με το μοντέλο της ατομικής ή γραμμικής μνήμης (Linearizability/atomic consistency) [20].



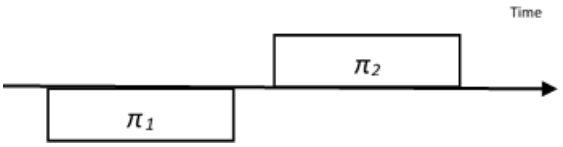
Σχήμα 2.3

Ιεραρχία των μοντέλων συνέπειας μνήμης [6].

2.4 Ορισμός Ατομικότητας

Ένα ατομικό αντικείμενο είναι ένα αντικείμενο που ορίζεται από ένα σύνολο πιθανών τιμών και υποστηρίζεται από λειτουργίες ανάγνωσης (read) και γραφής (write). Μπορούμε να έχουμε ταυτόχρονη πρόσβαση σε τέτοιου είδους αντικείμενα, έχοντας την αίσθηση πως οι λειτουργίες εκτελούνται σε μια διαδοχική σειρά. Έτσι κάποιες φορές ονομάζονται γραμμικά αντικείμενα.

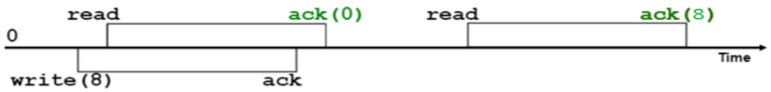
Ας εξηγήσουμε πιο αυστηρά την έννοια της ατομικότητας για να την κατανοήσουμε καλύτερα. Αρχικά πρέπει να ορίσουμε την έννοια της προτεραιότητας μεταξύ των λειτουργιών. Έστω ότι έχουμε δύο λειτουργίες την π_1 και π_2 , η π_1 προηγείται της π_2 εάν η απάντηση της π_1 συμβαίνει χρονικά πριν από την αίτηση της π_2.



Σχήμα 2.4

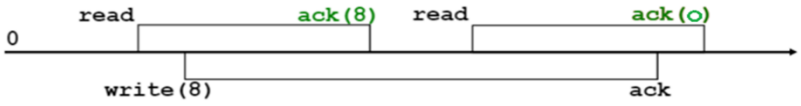
Οι συνθήκες που πρέπει να ισχύουν για να ικανοποιείται η ατομικότητα είναι οι εξής :

- Εάν μια λειτουργία ανάγνωσης δεν είναι ταυτόχρονη με μια λειτουργία γραφής, τότε η ανάγνωση επιστρέφει την τιμή που γράφτηκε από την τελευταία εγγραφή που προηγείται.
- Εάν μια ανάγνωση είναι ταυτόχρονη με μια γραφής, τότε η ανάγνωση επιστρέφει είτε την τιμή της αμέσως προηγούμενης εγγραφής, είτε την τιμή που αναγράφεται από την ταυτόχρονη εγγραφή.
- Αν έχουμε δύο διαδοχικές αναγνώσεις (μη επικαλυπτόμενες), τότε η δεύτερη ανάγνωση θα επιστρέψει την ίδια ή μια πιο πρόσφατη τιμή.



Σχήμα 2.6

Στο Σχήμα 2.6 βλέπουμε αρχικά μια λειτουργία ανάγνωσης να εκτελείται ταυτόχρονα με μια λειτουργία γραφή. Συνεπώς, η λειτουργεία ανάγνωσης θα μπορούσε να επιστρέψει την τιμή 0 ή 8 σύμφωνα με της συνθήκες οι οποίες έχουν οριστεί προηγουμένως. Ορθά επιστρέφει την τιμή 0. Ακολούθως, η δεύτερη ανάγνωση η οποία αρχίζει ξεκάθαρα μετά την πρώτη ανάγνωση δεν θα μπορούσε να επιστρέψει μια παλιότερη τιμή από την πρώτη ανάγνωση. Επομένως, ορθά επιστρέφει την τιμή 8 και μπορούμε να πούμε ότι το Σχήμα 2.6 διασφαλίζει την συνέπεια δεδομένων.



Σχήμα 2.7

Στο Σχήμα 2.7 βλέπουμε δυο αναγνώσεις να επικαλύπτονται με μια γραφή. Αρχικά, σύμφωνα με τις συνθήκες ατομικότητας που έχουμε ορίσει προηγουμένως, η πρώτη ανάγνωση μπορεί να διαβάσει μια τιμή που έχει γραφτεί από την προηγούμενη ολοκληρωμένη γραφή ή μια τιμή που γράφετε τώρα. Η πρώτη ανάγνωση επιστρέφει ορθά την τιμή 8 σύμφωνα με τις συνθήκες τις οποίες έχουμε ορίσει και μπορούμε να συμπεράνουμε ότι η λειτουργία γραφής έχει γράψει την τιμή 8. Επομένως η δεύτερη ανάγνωση δεν θα μπορούσε να επιστρέψει μια πιο παλιά τιμή. Στο συγκεκριμένο παράδειγμα η δεύτερη ανάγνωση επιστρέφει μια πιο παλιά τιμή και καταλήγουμε στο συμπέρασμα ότι το παράδειγμα μας παραβιάζει την ατομικότητα.

2.5 Συστήματα Απαρτίας

Ένα σύστημα απαρτίας (Quorum System) [11] είναι μια τεχνική που χρησιμοποιείται για τη διασφάλιση της συνέπειας και της διαθεσιμότητας των αντιγράφων των δεδομένων , ακόμη και στη παρουσία σφαλμάτων. Το σύστημα απαρτίας αποτελείται από μια συλλογή από σύνολα $Q=\{Q_1,...Q_n\}$, γνωστά ως quorums , από τα οποία το κάθε ένα περιέχει ένα σύνολο εξυπηρετητών $S= \{S_1,....,S_i\}$ για κάθε σύνολο Q_i. Το σύστημα απαρτίας είναι ένα υποσύνολο του S, έτσι ώστε κάθε δύο υποσύνολα του Q έχουν μια κοινή τομή.

Υπάρχουν ποικίλα είδη συστημάτων απαρτίας, ανάλογα με της ανάγκες που απαιτεί το κάθε σύστημα. Για την συγκεκριμένη διπλωματική θα μας απασχόληση το σύστημα απαρτίας της πλειοψηφίας (Majority) (Σχήμα 2.8 (β)) χωρίς όμως να υπονοούμε ότι είναι και το καλύτερο. Υπάρχουν επίσης εξίσου ή και καλύτερα συστήματα απαρτίας, ένα από αυτά είναι το σύστημα απαρτίας Matrix (Σχήμα 2.8 (α)).

(α) (β)

Σχήμα 2.8: Συστήματα Απαρτίας

(α) Matrix , (β) Majority

Στα συστήματα απαρτίας της πλειοψηφίας (Majority) κάθε quorum αποτελείται από τουλάχιστο $\lceil (n+1)/2 \rceil$ εξυπηρετητές. Μπορούμε εύκολα να αντιληφθούμε ότι και για τα δύο σύνολα του Σχήματος 2.8 (β) ισχύει ο περιορισμός. Από την άλλη, στα Matrix συστήματα απαρτίας , οι εξυπηρετητές σχηματίζουν εικονικά ένα ορθογώνιο πλέγμα και το κάθε πλέγμα αποτελείται από μια γραμμή και μία στήλη όπως μπορούμε να αντιληφθούμε από το Σχήμα 2.8 (α) και έχει μέγεθος $2\sqrt{n} - 1$ εξυπηρετητές.

Κεφάλαιο 3

Περιγραφή Αλγορίθμων

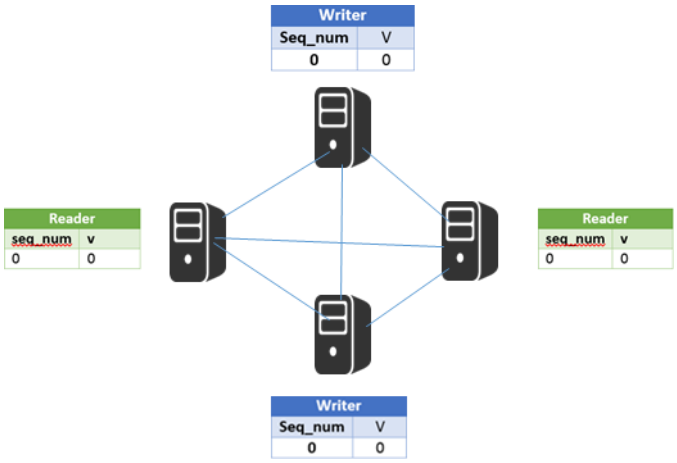
Σε αυτό το Κεφάλαιο θα εξηγήσουμε με λεπτομέρεια τους αλγόριθμους ABD [1] και LDR [5,6] . Επίσης θα εξηγήσουμε πως διαφέρει ο τροποποιημένος αλγόριθμος LDR [15].

3.1 Αλγόριθμος ABD

Ένας από τους αλγόριθμους ο οποίος έχει ως στόχο την προσομοίωση κοινόχρηστη μνήμη είναι ο αλγόριθμος γνωστός ως ABD [1]. Ο αλγόριθμος αυτός παρουσιάστηκε από τους Attiya, Bar-Noy και Dolev και αφορούσε την προσομοίωση μιας single-writer/multi-reader κοινόχρηστης μνήμης μέσα σε ένα δίκτυο ανταλλαγής μηνυμάτων. Έπειτα, ο αλγόριθμος αυτός επεκτάθηκε από τους Lynch και Shvartsman [13] σε μοντέλο multi-writer/multi-read και στην παρούσα διπλωματική εργασία έχουμε υλοποιήσει αυτόν τον αλγόριθμο. Συνεπώς, θα αναφερόμαστε στην επέκταση του αλγόριθμου ABD. Η κύρια διαφορά του αρχικού αλγορίθμου ABD που αφορούσε την προσομοίωση single-writer/multi-reader κοινόχρηστης μνήμης είναι ότι μια μόνο διεργασία μπορεί να εκτελεί εντολές γραφής και πολλές διεργασίες να διαβάζουν. Σε αντίθεση με τον αλγόριθμο ABD που επεκτάθηκε όπου πολλές διεργασίες μπορούν να διαβάζουν και να γράφουν.

Ο αλγόριθμος υλοποιεί multi-writer/multi-read ατομικά αντικείμενα όπου έχουμε S εξυπηρετητές να διατηρούν από ένα αντίγραφό του αντικειμένου. Μέχρι και οι μισοί εξυπηρετητές μπορούν να καταρρεύσουν ή να χάσουν επικοινωνία. Επίσης οι πελάτες αποτελούνται από R αναγνώστες και P εγγραφείς . Επιπλέον, κάθε λειτουργία γραφής και

ανάγνωσης παίρνει δύο γύρους επικοινωνίας. Ο αλγόριθμος αυτός εκτελεί αντιγραφές βασισμένες σε απαρτίες, οι οποίες δεν χρειάζονται οποιοδήποτε επιπλέον μηχανισμό για έλεγχο ταυτοχρονίας, σε αντίθεση με άλλους αλγορίθμους αντιγραφής. Το κύριο πλεονέκτημα του αλγορίθμου αυτού είναι η απόδοση του, καθώς επίσης και η ανοχή του σε σφάλματα.



Σχήμα 3.1

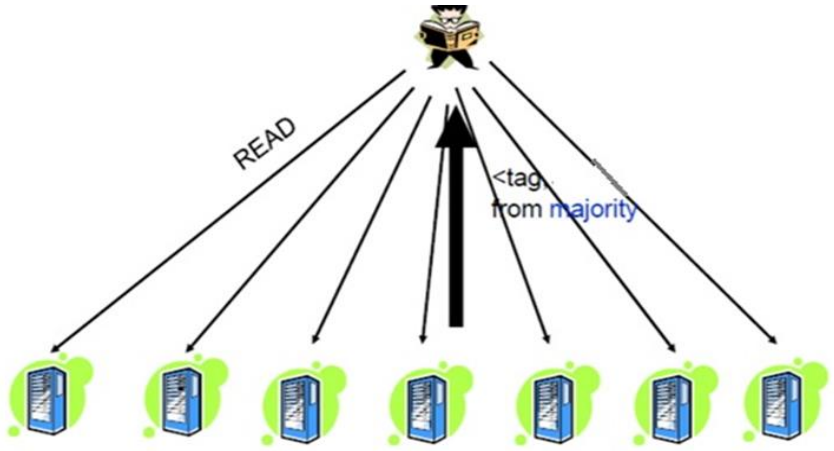
Σχηματισμός συστήματος ABD

Όλες οι διεργασίες κρατάνε από ένα αντίγραφο της τιμής που υπάρχει στην κοινή μνήμη. Επιπλέον, η κάθε τιμή είναι συσχετισμένη με ένα ακέραιο αριθμό όπου ο αριθμός αυτός αυξάνεται κάθε φορά που ο γραφέας γράφει κάποια καινούργια τιμή. Με άλλα λόγια, γίνεται χρήση ζευγών <*tag*,*value*> , όπου tag είναι ένας θετικός ακέραιος αριθμός και *value* η τιμή που αποθηκεύεται. Οι δύο βασικές λειτουργίες του αλγόριθμου ABD είναι η λειτουργία *γραφής* και *ανάγνωσης*. Οι λειτουργίες εκτελούνται πάνω σε πλειοψηφία από διεργασίες του συστήματος και κάθε φορά που εκτελείται η λειτουργία ανάγνωσης και εγγραφής, ακολουθείτε ένας επιπλέον γύρος ενημέρωσης (info). Παρόλο που κάποιες διεργασίες μπορούν να καταρρεύσουν, το σύστημα θα συνεχίσει να λειτουργεί κανονικά.

3.1.1 Λειτουργία Γραφής

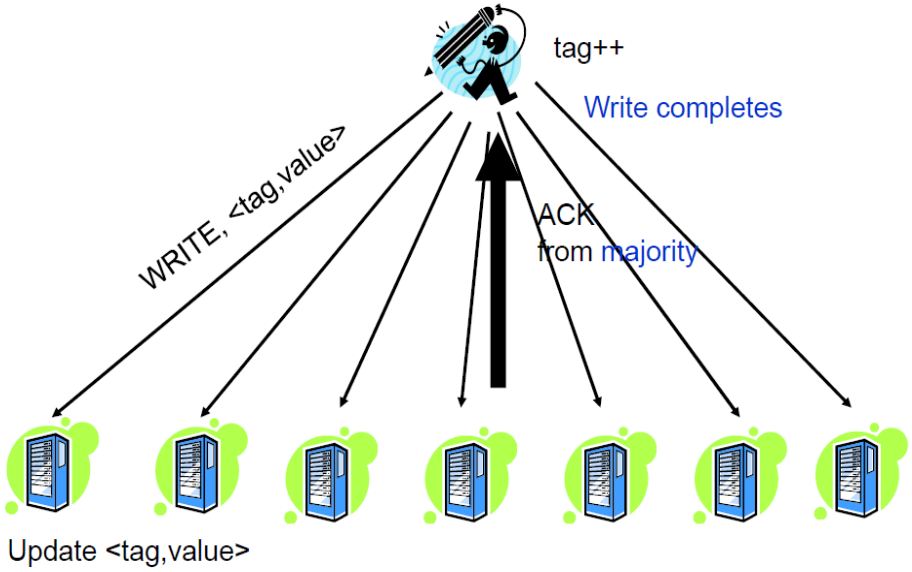
Η λειτουργία γραφής είναι υπεύθυνη για να ανανεώνει την τιμή που υπάρχει στην κοινόχρηστη μνήμη. Οποιαδήποτε διεργασία μπορεί να εκτελεί την λειτουργία γραφής. Η λειτουργία γραφής παίρνει δύο γύρους επικοινωνίας και έχει ως σκοπό να αλλάξει την τιμή που υπάρχει στην κοινόχρηστη μνήμη. Συνεπώς, κάθε φόρα που χρειάζεται να εκτελεστή η λειτουργία γραφής πρέπει πρώτα να υπολογίσει το μεγαλύτερο *tag*. Ο λόγος είναι ότι υπάρχουν πολλές διεργασίες που εκτελούν την λειτουργία γραφής και δεν μπορεί οποιαδήποτε διεργασία να

γνωρίζει αν έχει αλλάξει ή όχι το *tag*, κάποια άλλη διεργασία. Ακολούθως, ο γραφέας αυξάνει το *tag* , προσθέτει την τιμή του (*value*), και το κοινοποιεί στις διεργασίες.



Σχήμα 3.2

Λειτουργία γραφής, γύρος 1 [7]



Σχήμα 3.3

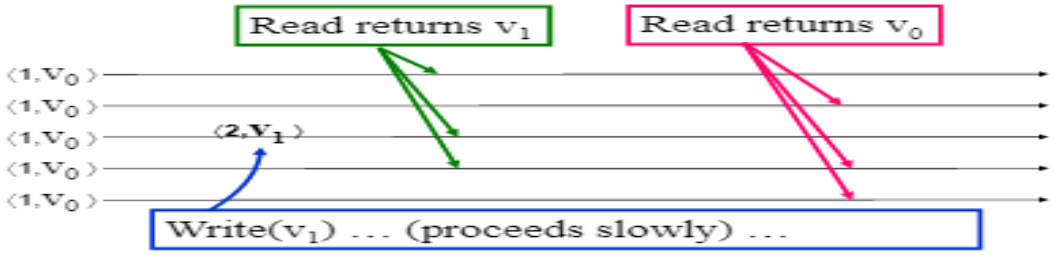
Λειτουργία γραφής, γύρος 2 [7]

3.1.2 Λειτουργία Ανάγνωσης

Η λειτουργεία ανάγνωσης παίρνει δύο γύρους επικοινωνίας. Οποιαδήποτε διεργασία μπορεί να εκτελέσει την λειτουργία ανάγνωσης. Αρχικά , η διεργασία στέλνει ένα αίτημα σε όλες τις διεργασίες του συστήματος για να μάθει πιο είναι το πιο πρόσφατο *tag* στο σύστημα, δηλαδή

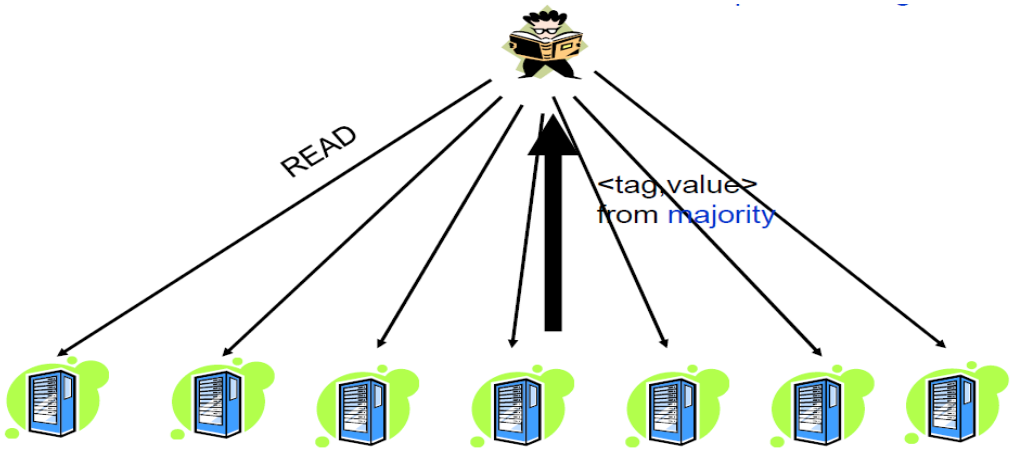
το *tag* με το μεγαλύτερο αριθμό. Ακολούθως, η διεργασία περιμένει να λάβει μια πλειοψηφία από απαντήσεις. Έπειτα, εκτελεί ένα επιπλέον γύρο για να ενημερώσει μια πλειοψηφία από διεργασίες για το τελευταίο tag που έχει βρει.

Για να μην παραβιαστεί η ατομικότητα ο γύρος ενημέρωσης (info) είναι αναγκαίος. Συνεπώς, η διεργασία ανάγνωσης πρέπει να προωθήσει το *tag* που έχει βρει ως μεγαλύτερο. Διαφορετικά η ατομικότητα θα παραβιαζόταν. Στο πιο κάτω παράδειγμα (Σχήμα 3.4) η διεργασία ανάγνωσης (πράσινο) επιστρέφει την τιμή v_1 ενώ η δεύτερη διεργασία ανάγνωσης (κόκκινο) επιστρέφει την τιμή v_0.



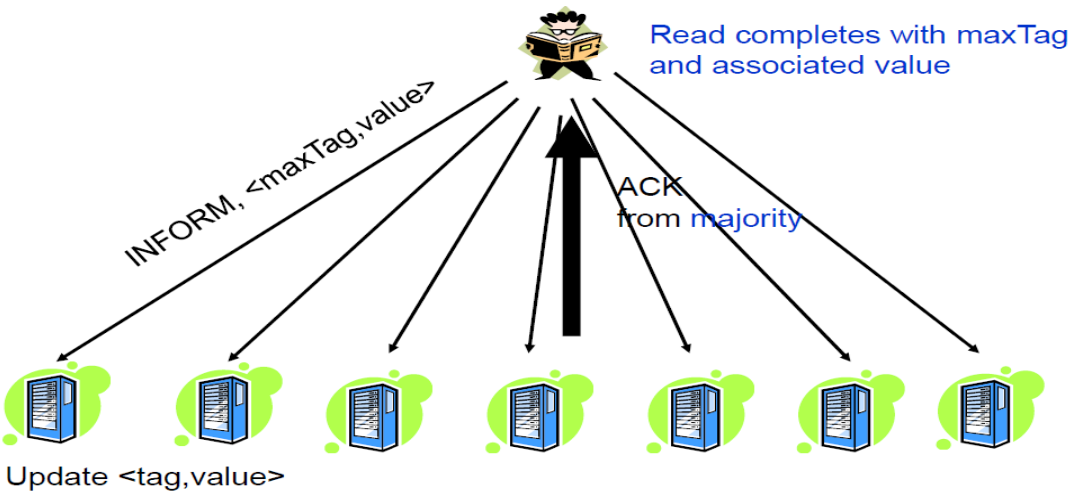
Σχήμα 3.4 [7]

Συνεπώς, παραβιάζεται η ατομικότητα επειδή η δεύτερη διεργασία αρχίζει μετά την ολοκλήρωση της πρώτης και δεν μπορεί να επιστρέψει μια παλιότερη τιμή σύμφωνα με τον ορισμό της ατομικότητας. Για να μην παραβιάζεται η ατομικότητα η δεύτερη διεργασία θα έπρεπε να επιστρέψει την ίδια τιμή με την πρώτη διεργασία ή μια πιο πρόσφατη .



Σχήμα 3.5

Λειτουργία ανάγνωσης, γύρος 1 [7]



Σχήμα 3.6

Λειτουργία ανάγνωσης, γύρος 2 [7]

3.2 Αλγόριθμος LDR

Ο αλγόριθμος Layer Data Replication (LDR) [5,6] έχει δημιουργηθεί με βάση τον αλγόριθμο ABD και ο στόχος του είναι το διάβασμα και γράψιμο μεγάλων αντικειμένων όπως αρχείων. Παράλληλα, διασφαλίζει την ατομικότητα των αντικειμένων. Η βασική ιδέα του αλγορίθμου είναι η αντιγραφή των δεδομένων σε αυθαίρετες τοποθεσίες. Επίσης, ο αλγόριθμος διατηρεί τα πραγματικά δεδομένα ξεχωριστά από τα μεταδεδομένα (metadata) των αρχείων. Με τον όρο μεταδεδομένα εννοούμε πληροφορίες σχετικά με την τοποθεσία των πραγματικών δεδομένων καθώς επίσης, και ποια είναι η πιο πρόσφατη έκδοση του κάθε αρχείου.

Ο αλγόριθμος είναι βασισμένος στο μοντέλο πελάτη-εξυπηρετητή και τρέχει σε οποιοδήποτε ασύγχρονο περιβάλλον ανταλλαγής μηνυμάτων και είναι κατάλληλος σε WAN και LAN δίκτυα. Η αποδοτικότητα του LDR αλγορίθμου βασίζεται στο γεγονός ότι το μέγεθος των μεταδεδομένα είναι πολύ μικρότερο σε σύγκριση με τα πραγματικά δεδομένα. Συνεπώς, οι περισσότερες λειτουργίες εκτελούνται πάνω στα μεταδεδομένα.

Τέλος, ο αλγόριθμος αποτελείται από τρείς διαφορετικές διεργασίες: Πελάτης, Εξυπηρετητής Ευρετηρίου και Εξυπηρετητής Αρχείων. Επίσης η επικοινωνία μεταξύ των διαφόρων

διεργασιών γίνεται μέσω ενός αξιόπιστου καναλιού και χρησιμοποιεί στο μοντέλο πελάτη-εξυπηρετητή.

3.2.1 Πελάτης

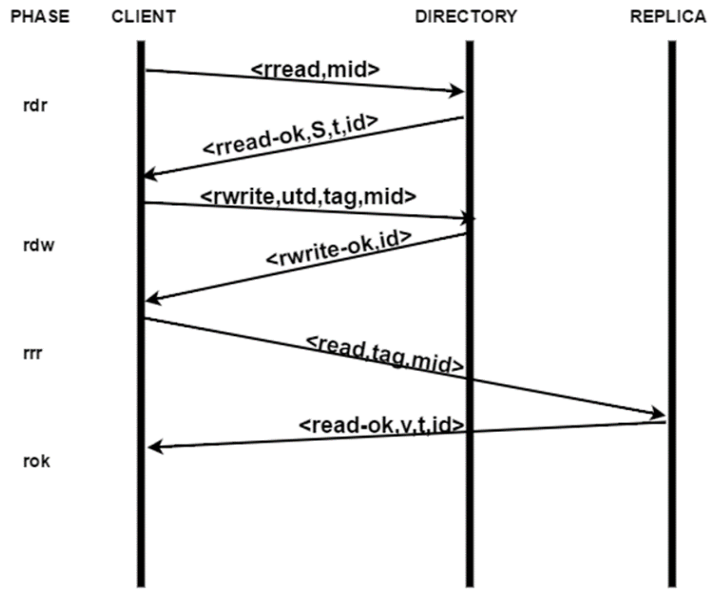
Η υπογραφή του αυτομάτου του πελάτη αποτελείται από τις ενέργειες εισόδου read_i και $\text{write}(v)_i$ και τις ενέργειες εξόδου $\text{read-ok}(v)_i$. Οι μεταβλητές καταστάσεων είναι οι εξής: η μεταβλητή *acc*, η οποία αποθηκεύει τα αναγνωριστικά των διεργασιών που ανταποκρίνονται σε ένα μήνυμα του πελάτη, τη μεταβλητή *phase*, η οποία αντιπροσωπεύει τη φάση στην οποία βρίσκεται η διεργασία για μια συγκεκριμένη στιγμή, τη μεταβλητή *tag* η οποία αντιπροσωπεύει την πιο ενημερωμένη χρονοσφραγίδα. Επιπλέον, στην μεταβλητή *val* αποθηκεύεται η ενημερωμένη τιμή της *x* και ένας φυσικός αριθμός mid που αντιπροσωπεύει τα μηνύματα που στέλνει η διεργασία.

Οι εντολές που δέχεται ο πελάτης είναι από τον χρήστη είτε για να διαβάσει ή να γράψει μια τιμή. Το κάθε μήνυμα που στέλνει ο πελάτης, συσχετίζεται με ένα φυσικό αριθμό έτσι ώστε όταν λαμβάνει κάποια απάντηση να γνωρίζει πόσο πρόσφατη είναι η απάντηση αυτή.

Λειτουργία ανάγνωσης

Η εκτέλεση της λειτουργίας ανάγνωσης ενός πελάτη C_i περνά από τις εξής τέσσερεις φάσεις: Read-directories-read (*rdr*), read-directories-write (*rdw*), read-replicas-read (*rrr*) και read-ok (*rok*). Η κάθε φάση χρειάζεται ένα επικοινωνιακό γύρο για να ολοκληρωθεί. Πρώτα, ο C_i παραλαμβάνει ένα αίτημα για ανάγνωση read_i, και στέλνει ένα μήνυμα της μορφής <*rread,mid*> σε όλους τους εξυπηρετητές ευρετηρίου και μπαίνει στη φάση '*rdr'*. Ακολούθως, ο C_i προσπαθεί να εντοπίσει ένα σύνολο από εξυπηρετητές ευρετηρίου οι οποίοι έχουν τιμή του *x*. Επομένως, οι εξυπηρετητές ευρετηρίου θα απαντήσουν στο αίτημα του πελάτη με ένα μήνυμα της μορφής <*rread-ok,S,t,mid*> , όπου *S* είναι ένα σύνολο από εξυπηρετητές αρχείων, οι οποίοι κρατάνε την πιο πρόσφατη έκδοση του *x*, και το *t* είναι η χρονοσφραγίδα του *x*. Μόλις ο πελάτης C_i πάρει απάντηση από μια απαρτία εξυπηρετητών ευρετηρίου , επιλέγει το ζευγάρι *(S,t)* με τη μεγαλύτερη τιμή χρονοσφραγίδας. Είναι προφανές ότι το σύνολο *S* το οποίο θα επιλέξει το πελάτης C_i θα είναι ένα σύνολο από εξυπηρετητές αρχείων οι οποίοι κρατάνε τη πιο πρόσφατη τιμή του *x*. Έπειτα, ο C_i θέτει το ζευγάρι *(utd,tag)* να ισούται με *(S,t),* και στέλνει σε όλους τους εξυπηρετητές ευρετηρίου το μήνυμα της μορφής <*rwrite,utd,tag,mid*> για να τους ενημερώσει ποια είναι η πιο πρόσφατη χρονοσφραγίδα του *x*. Ακολούθως, εισέρχεται στη φάση '*rdw'* και περιμένει μια απαρτία από εξυπηρετητές ευρετηρίου να απαντήσουν με ένα μήνυμα της μορφής <*rwrite-ok,mid*>. Τέλος, ο C_i στέλνει ένα μήνυμα της μορφής <*read,tag,mid*> σε ένα εξυπηρετητή αρχείων από το σύνολο *utd* για να διαβάσει τη τιμή του

x και εισέρχεται στη φάση ‘*rrr’*. Μόλις ο C_i πάρει απάντηση της μορφής <*rread-ok,v,t,id*> από τον εξυπηρετητή αρχείων, ανταποκρίνεται στο χρήστη με την ενέργεια read-ok(v)$_i$ και εισέρχεται στη φάση ‘*rok’*.



Σχήμα 3.7 [5,6]

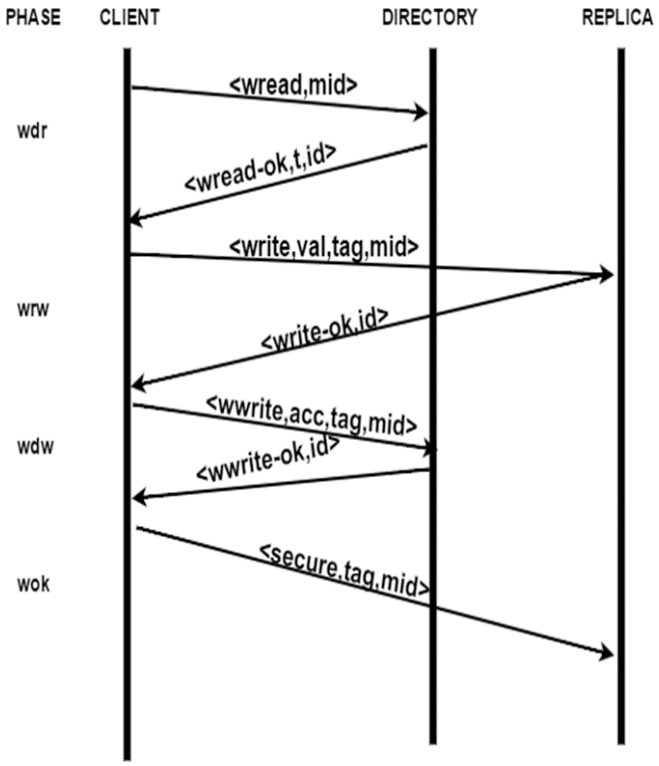
Στο Σχήμα 3.7 παρουσιάζεται η λειτουργία ανάγνωσης του πελάτη. Πιο συγκεκριμένα βλέπουμε τα μηνύματα τα οποία ανταλλάζονται μεταξύ του πελάτης και των δύο εξυπηρετητών.

Λειτουργία γραφής

Η εκτέλεση της λειτουργία γραφής από τον πελάτη C_i ακολουθεί 4 φάσεις:

write-directories-read (*wdr*), write-replicas-write (*wrw*), write-directories-write (*wdr*) και write-ok (*wok*). Οι πρώτες 3 φάσεις διαρκούν ένα επικοινωνιακό γύρο σε αντίθεση με τη τελευταία φάση που διαρκεί μόνο μισό επικοινωνιακό γύρο. Όταν ο πελάτης C_i, παραλαμβάνει ένα αίτημα για να αρχίσει η εγγραφή της τιμής *v* στο *x*, στέλνει ένα μήνυμα της μορφής <*wread,mid*> σε όλους τους εξυπηρετητές ευρετηρίου. Σε αυτό το στάδιο ο C_i μπαίνει στη φάση ‘*wdr’*, και προσπαθεί να εντοπίσει μια χρονοσφραγίδα (*tag*) αρκετά μεγάλη έτσι ώστε να κάνει τη δική του εγγραφή να είναι η πιο πρόσφατη και εισέρχεται στη φάση ‘*wdr’*. Ο C_i περιμένει απάντηση από μια απαρτία εξυπηρετητών ευρετηρίου με το μήνυμα της μορφής <*wread-ok,t,id*>, όπου το *t* αντιπροσωπεύει τη χρονοσφραγίδα. Μόλις λάβει τα μηνύματα, διαλέγει το μεγαλύτερο *t=(n,i)* όπου το *n* αντιπροσωπεύει ένα φυσικό αριθμό και το *i* είναι το αναγνωριστικό του πελάτη. Ακολούθως, επιλέγει μια μεγαλύτερη χρονοσφραγίδα κατά την λειτουργία της γραφής κάνοντας την χρονοσφραγίδα *tag=(n+1,i)* και στέλνει ένα μήνυμα της

μορφής <*wwrite,val,tag,mid*> σε όλους τους εξυπηρετητές αρχείων για να γράψουν την τιμή *(val,tag)* και εισέρχεται στη φάση '*wrw'*. Ο C_i περιμένει τουλάχιστον $f + 1$ απαντήσεις από τους εξυπηρετητές αρχείων έτσι ώστε να εγγυηθεί η γραφή θα επιβιώσει ακόμα και αν f εξυπηρετητές αρχείων καταρρεύσουν. Μόλις παραλάβει τις απαντήσεις από του εξυπηρετητές αρχείων, στέλνει ένα μήνυμα της μορφής <*wwrite,acc,tag,mid*> σε όλους τους εξυπηρετητές ευρετηρίου για να τους ενημερώσει για το νέο ζευγάρι *(val,tag)* και εισέρχεται στη φάση '*wdw'*. Ο πελάτης C_i περιμένει ένα μήνυμα της μορφής <*wwrite-ok*> από μια απαρτία από εξυπηρετητών ευρετηρίου. Μόλις λάβει την απαρτία στέλνει ένα μήνυμα της μορφής <*secure,tag,mid*> σε όλους του εξυπηρετητές αρχείων που υπάρχουν το σύνολο *acc*. Ο λόγος που στέλνει το συγκεκριμένο μήνυμα είναι για να ενημερώσει τους εξυπηρετητές αρχείων ότι μια απαρτία από εξυπηρετητές ευρετηρίου γνωρίζει για την πιο πρόσφατη χρονοσφραγίδα. Συνεπώς, οι εξυπηρετητές αρχείων ποτέ στο μέλλον δεν θα χρειαστεί να επιστρέψουν τη τιμή του *x* με μικρότερη χρονοσφραγίδα. Ως αποτέλεσμα, οι εξυπηρετητές αρχείων μπορούν να διαγράψουν όλες τις τιμές του *x* που είναι μικρότερες της τελευταίας χρονοσφραγίδας. Τέλος, ο πελάτης C_i ανταποκρίνεται στο χρήση με την εκτέλεση της εξωτερικής ενέργειας write-ok$_i$.



Σχήμα 3.8 [5,6]

Στο Σχήμα 3.8 παρουσιάζεται η λειτουργία γραφής του πελάτη. Πιο συγκεκριμένα αντιπροσωπεύει τα μηνύματα τα οποία ανταλλάζονται μεταξύ του πελάτη και των δύο εξυπηρετητών.

3.2.2 Εξυπηρετητής Ευρετηρίου

Η βασική αρμοδιότητα του Εξυπηρετητή ευρετηρίου είναι να κρατά πληροφορίες που αφορούν την τοποθεσία των ενημερωμένων (up-to-date) αντιγράφων. Η υπογραφή του αυτομάτου του εξυπηρετητή ευρετηρίου αποτελείται από την ενέργεια εισόδου recv(m)$_{i,j}$ και fail$_i$, και την ενέργεια εξόδου send(m)$_{ij}$. Οι μεταβλητές καταστάσεων του αυτόματου είναι η μεταβλητή *utd*, η οποία αντιπροσωπεύει τους ενημερωμένους εξυπηρετητές αρχείων, η μεταβλητή *tag* στην οποία περιέχεται η ενημερωμένη χρονοσφραγίδα και τέλος, μια ακολουθία μηνυμάτων *msg* στην οποία αποθηκεύονται τα προσωρινά μηνύματα που στέλνει το ευρετήριο.

Λειτουργία Ανάγνωσης

Όπως έχει αναφερθεί και προηγουμένως, ο εξυπηρετητής ευρετηρίου είναι αρμόδιος να κρατά πληροφορίες που αφορούν την τοποθεσία των ενημερωμένων αντιγράφων. Επομένως, όταν κάποιος πελάτης θέλει να διαβάσει την τιμή *x* τότε θα στείλει ένα μήνυμα της μορφής <*rread,mid*> και θα πάρει ως απάντηση το μήνυμα <*rread-ok,utd,tag,mid*>. Ο εξυπηρετητής ευρετηρίου μπορεί να παραλάβει και το μήνυμα της μορφής <*wread,mid*> όπου σε αυτή την περίπτωση ανταποκρίνεται με το μήνυμα της μορφής <*wread-ok,tag,mid*>.

Λειτουργία Γραφής

Όταν ο εξυπηρετητής ευρετηρίου παραλάβει το μήνυμα της μορφής <*rwrite,S,t,mid*> , ελέγχει εάν το $t < tag$, αν η συνθήκη ισχύει δεν κάνει καμία λειτουργία επειδή το write είναι παλιό. Στη περίπτωση όπου το $t=tag$, τότε το *S* είναι ένα σύνολο από εξυπηρετητές αρχείων η οποίοι έχουν την νέα τιμή του *x* και προσθέτει το *S* στο *utd*. Τέλος, στη περίπτωση όπου το $t > tag$, τότε έχουμε μια νέα τιμή του *x* και την αποθηκεύει. Σε όλα τα μηνύματα που στέλνονται από τον πελάτη, ο εξυπηρετητής ευρετηρίου απαντά με ένα μήνυμα της μορφής <*rwrite-ok,mid*>. Η ίδια λογική ακολουθείται και στη περίπτωση όπου ο εξυπηρετητής αρχείων παραλάβει το μήνυμα <*wwrite,S, t, mid* >, μόνο που σε αυτή τη περίπτωση θα απαντήσει στο πελάτη με ένα μήνυμα της μορφής <*wwrite-ok,id*>.

3.2.3 Εξυπηρετητής Αρχείων

Το αυτόματο του εξυπηρετητή αρχείων αποτελείται από τις ενέργειες εισόδου recv(m)$_{i,j}$ και fail$_i$, την ενέργεια εξόδου send(m)$_{i,j}$ και τις εσωτερικές ενέργειες gossip$_i$. Οι μεταβλητές καταστάσεων του αυτομάτου είναι η μεταβλητή *data* η οποία είναι ένα σύνολο της μορφής *[value,tag,bit]* , όπου *value* αντιπροσωπεύει την τιμή του *x* , *tag* η χρονοσφραγίδα που μας δείχνει πόσο πρόσφατη είναι η τιμή του *x*, και *bit* το (security bit) αλλιώς *bit* ασφαλείας που έχει ως πεδίο ορισμού {0,1} και αντιπροσωπεύει αν η εγγραφή έγινε επιτυχώς ή όχι. Ο κύριος

ρόλος του εξυπηρετητή αρχείων είναι να αποθηκεύει τιμές του *x*, τις οποίες οι πελάτες μπορούν να γράφουν και να διαβάζουν. Όμως, ο εξυπηρετητής αρχείων κρατά πολλές τιμές του *χ* όπου κάθε τιμή *χ* έχει μαζί ένα συσχετισμένο *tag* που αντιπροσωπεύει πόσο πρόσφατη είναι η τιμή αυτή και ένα security bit. Συνεπώς, αν έχουμε [*,*t*,1] σημαίνει ότι η εγγραφή ολοκληρώθηκε και είναι ασφαλής. Σε αντίθετη περίπτωση θα έχουμε [*,*t*,0] που σημαίνει ότι η εγγραφή δεν έχει ολοκληρωθεί ακόμη.

Λειτουργία Ανάγνωσης

Όταν ο εξυπηρετητής αρχείων λάβει ένα μήνυμα της μορφής <*read,t,mid*> από κάποιο πελάτη που θέλει να διαβάσει την τιμή με χρονοσφραγίδα *t*, τότε πρέπει να ελέγξει αν υπάρχει μεταβλητή στο σύνολο data [*v,t,**] και την επιστρέφει. Εάν δεν υπάρχει σημαίνει ότι η τιμή *x* με τη συγκεκριμένη χρονοσφραγίδα έχει διαγραφεί από τον garbage-collector. Σε αυτή την περίπτωση ο εξυπηρετητής αρχείων επιστρέφει τη μεγαλύτερη ασφαλή τιμή . Η μεγαλύτερη ασφαλής τιμή ,είναι η μεγαλύτερη τιμή που έχει το ασφαλές *bit* της να ισούται με 1. Η τιμή maxst(*data*) ορίζεται ως εξής :

$$maxst(data) = \begin{cases} (v,t)\big((v,t,1) \in data\big) \wedge \big((v',t',1) \in data \Rightarrow t \geq t' \\ (v0,t0) \nexists\ ,\ \ t\colon (v,t,1) \in data \end{cases}$$

Λειτουργία Εγγραφής

Στη περίπτωση όπου ο εξυπηρετητής αρχείων παραλαμβάνει το μήνυμα της μορφής <*write,t,mid*> από κάποιο πελάτη που ζητά να γράψει μια νέα τιμή *x* , τότε προσθέτει στη μεταβλητή data την πλειάδα [*v,t*,0].

Λειτουργία Περισυλλογής

Μια από τις σημαντικότερες λειτουργίες είναι η λειτουργία της περισυλλογής. Κατά τη λειτουργία εγγραφής, αρχικά γράφονται όλες οι νέες τιμές *x* μέχρι να γίνει το secure bit 1. Επομένως, θέλουμε ένα μηχανισμό ο οποίος να διαγράφει τις παλιές τιμές του *x* που δεν χρειάζονται το σύστημα. Ο μηχανισμός αυτός ονομάζεται garbage collector και είναι υπεύθυνος να διαγράφει τιμές οι οποίες είναι πλέον παλιές. Ο λόγος που δίνεται αυτή η διαδικασία είναι επειδή όταν ο εξυπηρετητής αρχείων παραλάβει ένα *secure* μήνυμα, γνωρίζει ότι δεν θα χρειαστεί να ξανά επιστρέψει τιμή με μικρότερη χρονοσφραγίδα. Ως αποτέλεσμα, οι παλιές τιμές να πρέπει να διαγραφούν.

3.3 Τροποποιημένος Αλγόριθμος LDR

Παρόλο που ο αλγόριθμος LDR [5,6] ήταν αρκετά αποδοτικός για την μεταφορά μεγάλων αντικειμένων όπως αρχείων δεν λαμβάνει υπόψη αν το αρχείο έχει τροποποιηθεί από κάποιον άλλο πελάτη. Με άλλα λόγια, δεν λαμβάνει υπόψη αν ο πελάτης έκανε αλλαγές πάνω στο πιο πρόσφατο αρχείο. Όπως έχει αναφερθεί και προηγουμένως ο αλγόριθμος LDR είναι βασισμένος στον ABD [13] και πιο συγκεκριμένα η επικοινωνία με τους εξυπηρετητές ευρετηρίου είναι ο αλγόριθμος ABD. Ο τροποποιημένος αλγόριθμος LDR [14] έρχεται να επιλύσει το πρόβλημα αυτό και κατά την λειτουργία της γραφής λαμβάνει υπόψη αν ο πελάτης έχει κάνει τις αλλαγές του πάνω στο πιο πρόσφατο αρχείο. Πιο κάτω περιγράφεται η λειτουργία γραφής του τροποποιημένου αλγόριθμου LDR.

Η λειτουργία γραφής ενός πελάτη ακολουθεί της εξής φάσης:

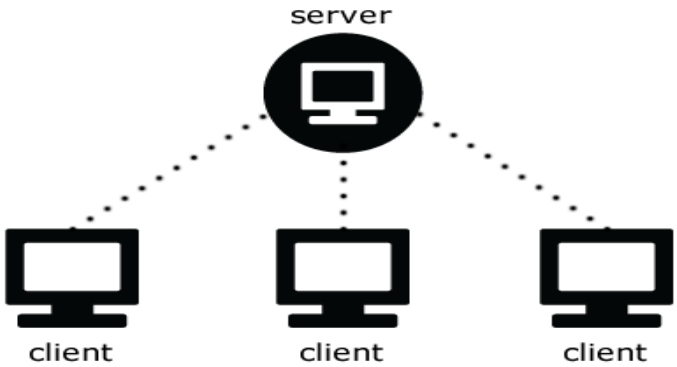
Αρχικά ο πελάτης C_i εκτελεί την λειτουργία γραφής και στέλνει αίτημα σε όλους τους εξυπηρετητές ευρετηρίου για να μάθει ποια είναι η πιο πρόσφατη έκδοση του αρχείου. Όπως και στον αλγόριθμο ABD, ο πελάτης C_i περιμένει απάντηση από μια πλειοψηφία (majority) εξυπηρετητές ευρετηρίου. Μόλις πάρει τις απαντήσεις διαλέγει την μεγαλύτερη χρονοσφραγίδα (*tag*) που υπάρχει στο σύστημα. Αν η χρονοσφραγίδα που υπάρχει στο σύστημα ισούται με την χρονοσφραγίδα του αρχείου που έχει τότε συνεχίζει κανονικά την λειτουργία γραφής όπως ο κανονικός αλγόριθμος LDR. Σε αντίθετη περίπτωση ανακοινώνει στους εξυπηρετητές ευρετηρίου την πιο πρόσφατη χρονοσφραγίδα που έχει εντοπίσει όπως είναι η δεύτερη φάση της λειτουργίας ανάγνωσης του αλγορίθμου ABD και ζήτα από τους εξυπηρετητές αρχείων να του στείλουν το αρχείο με την συγκεκριμένη χρονοσφραγίδα.

Κεφάλαιο 4

Υλοποίηση Αλγορίθμων

4.1 Μοντέλο Πελάτη – Εξυπηρετητή

Μια από τις δημοφιλέστερες αρχιτεκτονικές δικτύου είναι το μοντέλο πελάτη – εξυπηρετητή [18] το οποίο αποτελείται από δύο είδη διεργασιών, η διεργασία του πελάτη (Client) και η διεργασία του εξυπηρετητή (server). Ευρέως χρησιμοποιημένο μοντέλο σε δίκτυα WAN (διαδίκτυο) καθώς επίσης, και σε τοπικά δίκτυα (LAN). Σκοπός της κάθε διεργασίας είναι να εκτελεί μια συγκεκριμένη δουλειά. Επιπλέον, κύριος σκοπός των εξυπηρετητών είναι να παρέχουν διάφορες υπηρεσίες στους πελάτες. Υπάρχουν αρκετά παραδείγματα εφαρμογών που χρησιμοποιούν το μοντέλο Πελάτη – Εξυπηρετητή. Μερικές σημαντικές εφαρμογές είναι ο web browser - web server , mail server.



Σχήμα 4.1

Μοντέλο Πελάτη – Εξυπηρετητή

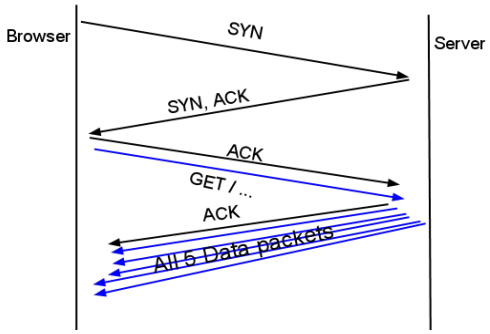
Η βασική φιλοσοφία αυτού του μοντέλου είναι αρχικά η εγκαθίδρυση σύνδεσής του πελάτη μαζί με τον εξυπηρετητή. Έπειτα, ο πελάτης κάνει αίτηση στον εξυπηρετητή να του επιστρέψει της απαιτούμενές πληροφορίες. Ο εξυπηρετητής θα αποδεχτεί την αίτηση του πελάτη , και ακολούθως θα απαντήσει πίσω με τις αιτούμενες πληροφορίες.

Υπάρχουν δύο είδη επεξεργασίας όπου κάποιος επεξεργαστής μπορεί να εκτελεί , η σειριακή επεξεργασία και η ταυτόχρονη επεξεργασία. Οι επεξεργαστές οι οποίοι εκτελούν σειριακή επεξεργασία (iterative), επεξεργάζονται με την σειρά τις αιτήσεις , τη μια μετά την άλλη. Ενώ, οι εξυπηρετητές που εκτελούν ταυτόχρονη επεξεργασία, επεξεργάζονται της αιτήσεις ταυτόχρονα. Για κάθε αίτημα που λαμβάνει ο εξυπηρετητής δημιουργεί ένα νήμα (thread) το οποίο είναι αρμόδιο να εξυπηρετήσει το κάθε πελάτη. Το κάθε νήμα ολοκληρώνει μόνο όταν εξυπηρετήσει το πελάτη που έχει αναλάβει.

4.2 Πρωτόκολλο TCP

Στην καθημερινότητα μας, πρωτόκολλο είναι ένα σύνολο από συμβάσεις που καθορίζουν το πως πρέπει να πραγματοποιηθεί μια διαδικασία. Στα δίκτυα υπολογιστών, πρωτόκολλο είναι

ένα σύνολο από συμβάσεις που καθορίζουν το πως ανταλλάσσουν μεταξύ τους δεδομένα οι υπολογιστές του δικτύου. Στον κόσμο των δικτύων, το πρωτόκολλο είναι αυτό που επιβεβαιώνει την αξιόπιστη αποστολή και λήψη δεδομένων, επίσης το πως γίνεται ο έλεγχος και πως αντιμετωπίζονται ενδεχόμενα λάθη. Οι περισσότερες σύγχρονες υπηρεσίες στο Διαδίκτυο βασίζονται στο TCP [17]. Ένα από τα πιο γνωστά παραδείγματα που βασίζεται στο TCP είναι το HTTP , γνωστό ως υπηρεσίες World Wide Web (Παγκόσμιος Ιστός).



Σχήμα 4.2

Πρωτόκολλο TCP [10]

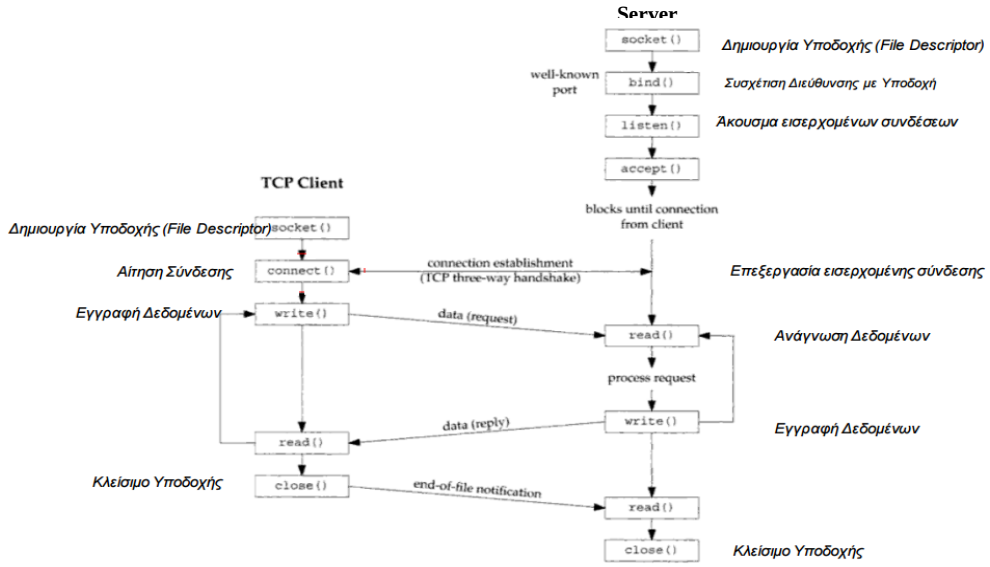
Μια σύνδεση TCP παρέχει αμφίδρομη μεταφορά δεδομένων μεταξύ δυο διεργασιών όπου, με τον όρο διεργασίες εννοούμε υπολογιστές. Με άλλα λόγια, μπορούν να μεταφέρονται δεδομένα από τον πρώτο υπολογιστή στον δεύτερο και ταυτόχρονα από το δεύτερο στον πρώτο υπολογιστή. Μια σύνδεση TCP, είναι πάντα από σημείο προς σημείο , ανάμεσα σε ένα μόνο αποστολέα και ένα μόνο δέκτη.

4.3 Προγραμματισμός με Υποδοχές Ροής

Σε αυτό το υποκεφάλαιο εξετάζουμε τον τρόπο με τον οποίο έγινε η επικοινωνία μεταξύ των διάφορων διεργασιών. Αν αναλογιστούμε ότι οι διεργασίες σε ένα κατανεμημένο σύστημα αρχείων βρίσκονται σε γεωγραφικά απομακρυσμένες τοποθεσίες, τότε ο μόνος τρόπος επικοινωνίας είναι μέσω δικτύου. Επομένως η επικοινωνία έγινε με την βοήθεια προγραμματισμού υποδοχών [10,11] ροής σε γλώσσα προγραμματισμού C.

Η υλοποίηση του κατανεμημένου συστήματος αρχείων αποτελείται από 4 είδη διεργασιών, εκ των οποίων μόνο μια διεργασία έχει το ρόλο πελάτη και οι υπόλοιπες το ρόλο του εξυπηρετητή. Αρχικά, ο κάθε εξυπηρετητής δημιουργεί μια βασική διεργασία η οποία είναι υπεύθυνη να ακούει σε μια συγκεκριμένη υποδοχή, έτσι ώστε να γνωρίζει πότε κάποιος πελάτης θέλει να ενωθεί στην συγκεκριμένη υποδοχή. Έπειτα, για να μπορέσει ο πελάτης να

επικοινωνήσει με τον κάθε εξυπηρετητή πρέπει να δημιουργήσει μια υποδοχή ροής (stream socket). Κατά την δημιουργία της υποδοχής ροής ο πελάτης πρέπει να καθορίσει την διεύθυνση και υποδοχή του εξυπηρετητή. Από το Σχήμα 4.3 μπορούμε να αντιληφθούμε την διαδικασία σύνδεσης και επικοινωνίας μεταξύ πελάτη και εξυπηρετητή.



Σχήμα 4.3

Διαδικασία σύνδεσης και επικοινωνίας μεταξύ πελάτη και εξυπηρετητή [7].

4.3.1 Είδη Υποδοχών

Τα δύο πιο συνηθισμένα είδη υποδοχών [11] είναι οι υποδοχές ροής(stream sockets) για συνδέσεις TCP και οι τηλεγραφικές υποδοχές (datagram sockets) για συνδέσεις UDP. Αξίζει εδώ να αναφέρουμε ότι οι υποδοχές ροής είναι ευρέως χρησιμοποιούμενες γιατί προσφέρουν αξιοπιστία στην επικοινωνία. Επίσης, εγγυόνται ότι θα φτάσουν όλα τα μηνύματα και με την σωστή σειρά. Από την άλλη, οι τηλεγραφικές υποδοχές δεν προσφέρουν αξιόπιστη σύνδεση και να μηνύματα μπορεί να παραλειφθούν με οποιαδήποτε σειρά χωρίς να είναι απαραίτητα κακό. Για παράδειγμα, υπάρχουν εφαρμογές που απαιτούν τα μηνύματα να αποστέλνονται γρήγορα έτσι θα προτιμούσαν να χρησιμοποιήσουν τις τηλεγραφικές υποδοχές. Στο κατανεμημένο σύστημα αρχείο που θα αναλύσουμε όλες οι επικοινωνίες έχουν γίνει με υποδοχές ροής , διότι απαιτείται αξιόπιστη μεταφορά δεδομένων.

4.3.2 Δημιουργία Υποδοχής

Όπως και στην δημιουργία αρχείων χρειαζόμαστε ένα περιγραφέα αρχείων έτσι και στην δημιουργία υποδοχών χρειάζεται να δημιουργήσουμε ένα περιγραφέα υποδοχής. Για την δημιουργία υποδοχής χρειάζεται να καλέσουμε την συνάρτηση socket() η οποία επιστρέφει

το περιγραφέα της υποδοχής. Οι παράμετροι της συνάρτησης socket() είναι η δομή της διεύθυνσης, το είδος της υποδοχής , και το πρωτόκολλο. Για την παρούσα διπλωματική χρησιμοποιήθηκαν οι παράμετροι: *AF_INT* , *SOCK_STREAM* και 0. Το *SOCK_STREAM* υποδηλώνει ότι θα χρησιμοποιηθεί το πρωτόκολλο TCP.

```
#include  <sys/socket.h>

int socket( int domain, int type, int protocol);
```

Σε περίπτωση επιτυχία επιστρέφει 0, διαφορετικά -1.

4.3.3 Δέσμευση Θύρας Υποδοχής

Η συνάρτηση bind() είναι υπεύθυνη για τη συσχέτιση της υποδοχής με κάποια διεύθυνση και θύρα. Για την διεργασία του εξυπηρετητή θα καθορίσουμε εμείς μια γνωστή διεύθυνση στην οποία θα καταφθάνουν οι αιτήσεις από τους πελάτες. Από την άλλη, για τη διεργασία του πελάτη θα αφήσουμε το σύστημα να αποφασίσει την διεύθυνση που θα του συσχετίσει εφόσον δεν χρειάζεται να την γνωρίζουμε.
Το *sockfd* είναι ο περιγραφέας υποδοχής που επιστράφηκε από την συνάρτηση socket(). Το *addr* είναι δείκτης σε δομή struct sockadd που περιέχει της πληροφορίες για τη διεύθυνση και *len* είναι για το μέγεθος.

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

Επιστρέφει -1 σε περίπτωση λάθους, διαφορετικά 0.

4.3.4 Άκουσμα αιτήσεων σύνδεσης

Το επόμενο βήμα μετά την δημιουργία και τη συσχέτιση της υποδοχής είναι να ανακοινωθεί από τον εξυπηρετητή ότι είναι έτοιμος να δεχτεί αιτήσεις σύνδεσης στη θύρα *sockfd*. Αυτό επιτυγχάνεται με τη συνάρτηση listen().

Η παράμετρος *sockfd* είναι ο περιγραφέας υποδοχής που δημιουργήθηκε από την socket(). Η *maxcli* δηλώνει το μέγιστο αριθμό αιτήσεων που μπορούν να περιμένουν στην ουρά για

εξυπηρέτηση. Στην περίπτωση όπου ο αριθμός αιτήσεων ξεπεράσει το μέγιστο αριθμό , τότε οι επιπρόσθετες αιτήσεις απορρίπτονται , μέχρι να εξυπηρετηθούν κάποιες από την ουρά έτσι ώστε να δημιουργηθεί χώρος στην ουρά για καινούργιες αιτήσεις.

```
#include  <sys/socket.h>

int listen(int sockfd,  int maxcli);
```

Σε περίπτωση επιτυχία, επιστρέφει 0, διαφορετικά -1.

4.3.5 Σύνδεση με Εξυπηρετητή

Στο μοντέλο πελάτη – εξυπηρετητή για να υπάρξει επικοινωνία μεταξύ δύο διεργασιών πρέπει πρώτα να προηγηθεί η τριμερής χειραψία την οποία αρχίζει ο πελάτης. Επομένως, για να εγκαθιδρυθεί η σύνδεση μεταξύ της υποδοχής του πελάτη και του εξυπηρετητή πρέπει να καλέσει την συνάρτηση connect().
Το *sockfd* είναι ο περιγραφέας υποδοχής του πελάτη. Στην περίπτωση όπου ο περιγραφέας υποδοχής του πελάτη δεν είναι συσχετισμένος με κάποια διεύθυνση , θα γίναι αυτόματα από την συνάρτηση.

```
#include  <sys/socket.h>

int connect(int sockfd, const struct sockaddr *addr, socklen_t len);
```

Σε περίπτωση επιτυχίας επιστρέφει 0, διαφορετικά -1.

4.3.6 Αποδοχή Αίτησης Σύνδεσης

Όταν ο εξυπηρετητής είναι πρόθυμος να δεχτεί αιτήσεις σύνδεσης με την συνάρτηση listen() το επόμενα βήμα είναι να αποδεχθεί κάποια εισερχόμενη αίτηση με τη συνάρτηση accept().

Κατά την εκτέλεση της συνάρτηση listen() επιστρέφεται ο νέος περιγραφέας υποδοχής της σύνδεσης. Στις παραμέτρους *addr* και *len* αποθηκεύονται πληροφορίες σχετικά με την

ταυτότητα του πελάτη όπου στην συγκεκριμένη περίπτωση είναι η διεύθυνση του. Επιπλέον, στη παράμετρο len αποθηκεύεται το μέγεθος της διεύθυνσης του.

```
#include  <sys/socket.h>

int connect(int sockfd, const struct sockaddr *addr, socklen_t len);
```

Σε περίπτωση επιτυχία, η συνάρτηση επιστρέφει το νέο περιγραφέα υποδοχής της σύνδεσης, διαφορετικά επιστρέφει -1.

4.3.7 Ανταλλαγή Δεδομένων

Οι συναρτήσεις send() και recv() είναι δυο από τις πιο σημαντικές συναρτήσεις στον προγραμματισμό υποδοχών. Μας παρέχουν την δυνατότητα ανταλλαγή μηνυμάτων μεταξύ δύο διεργασιών.

Η συνάρτηση send() στέλνει *n* bytes από τη συμβολοσειρά που δείχνει το *buf*, στην υποδοχή που δείχνει ο περιγραφέας υποδοχής *fd*. Η συνάρτηση recv() διαβάζει μέχρι *n* bytes από το περιγραφέα υποδοχής *fd* και τα αποθηκεύει στη συμβολοσειρά που δείχνει το *buf*.

```
#include  <sys/socket.h>
 #include <unistd.h>
ssize_t send(int sockfd, const void *buf, size_t nbytes, int flags);
ssize_t recv(int sockfd, void *buf, size_t nbytes, int flags);
```

Επιστρέφουν τον πραγματικό αριθμό δεδομένων που έχουν σταλεί ή διαβαστεί, διαφορετικά επιστρέφεται -1.

4.3.8 Μεταφορά αρχείων

Η συνάρτηση sendfile() είναι κατάλληλη για τη αποδοτική μεταφορά αρχείων σε προγραμματισμό υποδοχών. Παρόλο που είναι δυνατό να μεταφέρουμε αρχεία χρησιμοποιώντας τη συνάρτηση send, η κύρια διαφορά της sendfile είναι ότι αντιγράφει τα δεδομένα κατευθείαν στη υποδοχή επικοινωνίας χωρίς να χρειάζεται να τα μεταφέρει πρώτα στο χώρο του χρήστη. Με αυτό τον τρόπο γίνεται πολύ πιο αποδοτική η μεταφορά αρχείων.
Η παράμετρος *out_fd* αντιπροσωπεύει το περιγραφέα υποδοχή όπου θα γραφτούν τα δεδομένα, η παράμετρος *in_fd* είναι ο περιγραφέας αρχείου και η παράμετρος *offset* είναι για να

γνωρίζουμε μέχρι πιο σημείο του αρχείου έχει διαβάσει μέχρι στιγμής. Τέλος, το *count* είναι ο αριθμός των bytes που πρέπει να γραφτούν στο περιγραφέα υποδοχής.

```
#include  <sys/sendfile.h>

ssize_t sendfile(int out_fd, int in_fd,  off_t *offset ,  size_t count);
```

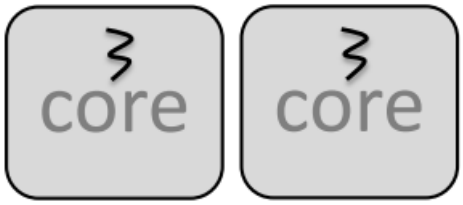
Επιστρέφουν τον πραγματικό αριθμό δεδομένων που έχουν σταλεί ή διαβαστεί, διαφορετικά επιστρέφεται -1.

4.4 Νήματα

Η ύπαρξη πολυνηματικού προγραμματισμού για τις περισσότερες εφαρμογές είναι σχεδόν απαραίτητη. Σχεδόν όλες οι σύγχρονες υπολογιστικές οντότητες έχουν περισσότερους από ένα επεξεργαστές ως αποτέλεσμα να υπάρχει η ανάγκη να αξιοποιηθούν. Στην παρούσα διπλωματική εργασία υπήρχαν πολλές περιπτώσεις που υπήρχε η ανάγκη για ταυτόχρονη εξυπηρέτηση.

Γενικά η ταυτόχρονη εξυπηρέτηση μπορεί να επιτευχθεί με διάφορους τρόπους. Οι πιο συνηθισμένοι είναι η χρήση διεργασιών ή νημάτων. Εντούτοις τα νήματα έχουν αρκετά πλεονεκτήματα σε σχέση με τις διεργασίες. Η δημιουργία διεργασιών γίνεται με την εντολή fork() η οποία δεσμεύει αρκετή μνήμη. Επίσης, η εναλλαγή διεργασιών είναι 5 φορές πιο αργή από την <<εναλλαγή>> νημάτων. Ακόμη σε συστήματα *N* επεξεργαστών τα νήματα εκτελούνται πραγματικά παράλληλα.

Επομένως για τη ταυτόχρονη εξυπηρέτηση των πελατών έγινε η χρήση νημάτων με την βοήθεια της βιβλιοθήκης pthread.h. Για κάθε νέα αίτηση που δέχεται ο εξυπηρετητής δημιουργείται ένα νήμα το οποίο είναι υπεύθυνο να εξυπηρετήσει το κάθε πελάτη.



Σχήμα 4.4

Απεικόνιση Νημάτων

Το Σχήμα 4.4 απεικονίζει δύο νήματα τα οποία κάνουν δυο διαφορετικές εργασίες παράλληλα. Για να μεταγλωττίσουμε μια εφαρμογή, η οποία χρησιμοποιεί νήματα πρέπει να συμπεριλάβουμε τη πιο κάτω βιβλιοθήκη.

```
#include  <pthread.h>
```

Επίσης για την μεταγλώττιση πρέπει να χρησιμοποιηθεί η επιλογή του GCC –lpthread.

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

4.4.1 Δημιουργία Νήματος

Κατά την δημιουργία ενός νήματος πρέπει να του ορίσουμε κάποια χαρακτηριστικά καθώς επίσης να καθορίσουμε ποια θα είναι η εργασία του. Η πιο κάτω συνάρτηση δημιουργεί ένα καινούργιο νήμα που εκτελεί την συνάρτηση thread_f. Η παράμετρος *thread επιστρέφει το *thread_id* του νήματος που δημιουργείται. Επίσης μπορούμε να ορίσουμε κάποια χαρακτηριστικά για το νήμα που θα δημιουργηθεί με την παράμετρο **attr*. Με την παράμετρο thread_f καθορίζουμε την συνάρτηση που εκτελεί το νήμα και ποιες παραμέτρους θα πάρει στο *arg*.

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

4.4.2 Τερματισμός Νήματος

Όταν κάποιο νήμα τελειώσει αυτό που έπρεπε να κάνει τότε πρέπει να τερματιστεί. Αυτό επιτυγχάνεται με την πιο κάτω συνάρτηση. Σε περίπτωση όπου η αρχική διεργασία-νήμα καλέσει την πιο κάτω συνάρτηση τότε περιμένει μέχρι να τερματίσουν όλα τα νήματα παιδιά την εκτέλεση τους.

```
#include  <pthread.h>
 void pthread_exit(void *retval);
```

4.4.3 Συνέπεια Μνήμης σε Νήματα

Εφόσον τα νήματα έχουν κοινή μνήμη μπορεί να δημιουργηθούν προβλήματα ασυνέπειας μνήμης. Για παράδειγμα λάθη μνήμης τα οποία δημιουργούνται όταν ένα νήμα πραγματοποιεί κάποια λειτουργία πάνω σε κάποια περιοχή μνήμης ενώ ένα άλλο νήμα προσπαθεί να μεταβάλει αυτή την περιοχή μνήμης.

Στην περίπτωση μας όταν πολλοί πελάτες επικοινωνούν με το εξυπηρετητή ταυτόχρονα τότε έχουμε πολλά νήματα να έχουν πρόσβαση στη κοινή μνήμη. Επομένως χρειαζόμαστε ένα μηχανισμό ο οποίος θα αποφεύγει προβλήματα ασυνέπεια μνήμης. Στην παρούσα διπλωματική για το συγχρονισμό μεταξύ νημάτων έγινε η χρήση της βιβλιοθήκης posix η οποία παρέχει μια απλοποιημένη εκδοχή σηματοφόρων , τους δυαδικούς σηματοφόρους (Mutexs).

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

4.5 Βιβλιοθήκη Glib

Η γλώσσα προγραμματισμού C είναι μια από τις αποδοτικότερες γλώσσες προγραμματισμού. Ο κύριος λόγος βασίζεται στο γεγονός ότι η διαχείριση μνήμης γίνεται από μεριάς του προγραμματιστή. Παρ΄ όλα αυτά, δεν υποστηρίζει έτοιμες συναρτήσεις για διαχείριση δεδομένων, πχ. δομές δεδομένων.

Η βιβλιοθήκη Glib μας δίνει τη δυνατότητα να προγραμματίζουμε στην γλώσσα προγραμματισμού C και παράλληλα να χρησιμοποιήσουμε ποικίλες δομές δεδομένων, όπως συνδεδεμένες λίστες, δένδρα και πίνακες κατακερματισμού.

Για να μπορέσουμε να χρησιμοποιήσουμε τη βιβλιοθήκη GLib πρέπει να συμπεριλάβουμε την πιο κάτω εντολή κατά την μεταγλώττιση του προγράμματος.

```
`pkg-config --cflags --libs glib-2.0`
```

Επίσης θα πρέπει να προσθέσουμε την πιο κάτω βιβλιοθήκη στο πρόγραμμα μας.

4.5.1 Συνδεδεμένες Λίστες

```
#include <glib.h>
```

Μια από τις βασικότερες δομές δεδομένων είναι η συνδεδεμένη λίστα. Μας δίνει τη δυνατότητα να αποθηκεύουμε οποιοδήποτε τύπο δεδομένων χωρίς να πρέπει να καθορίσουμε εξαρχής το μέγεθος της λίστας.

```
#include <glib.h>
int main(int argc, char** argv) {
 GSList* list = NULL;
 printf("The list is now %d items long\n", g_slist_length(list));
 list = g_slist_append(list, "test.txt");
 list = g_slist_append(list, "eng.mp3");
 printf("The list is now %d items long\n", g_slist_length(list));
 g_slist_free(list);
 return 0;
}
```

Το πιο πάνω παράδειγμα δημιουργεί μια συνδεδεμένη λίστα και προσθέτει δυο ονόματα αρχείων. Αρχικά, πρέπει να δηλώσουμε τη λίστα μας που θα είναι τύπου GSList*.

Για να εισάγουμε κάποιο στοιχείο στην λίστα χρησιμοποιούμε τη συνάρτηση g_slist append. Η συνάρτηση αυτή παίρνει δύο παραμέτρους και επιστρέφει την καινούργια αρχή της λίστας μας. Η πρώτη παράμετρος είναι το όνομα της λίστας μας και η δεύτερη το στοιχείο που πρόκειται να προσθέσουμε στη λίστα.

Η συνάρτηση g_slist_length επιστέφει το μέγεθος της λίστας. Με άλλα λόγια, πόσα στοιχεία έχουν εισαχθεί στη λίστα. Τέλος, η συνάρτηση g_slist_free χρησιμοποιείται στην περίπτωση που θέλουμε να διαγράψουμε τη λίστα μας. Διαγράφει όλη τη μνήμη που έχει δεσμευθεί για τη συγκεκριμένη λίστα.

```
#include <glib.h>
int main(int argc, char** argv) {
 GSList* list = NULL;
 list = g_slist_append(list, " test.txt ");
 list = g_slist_prepend(list, "file1.pdf");
 printf("The list is now %d items long\n", g_slist_length(list));
 list = g_slist_remove(list, "test.txt ");
 printf("The list is now %d items long\n", g_slist_length(list));
 g_slist_free(list);
 return 0;
}
***** Output *****
The list is now 2 items long
```

Το πιο πάνω παράδειγμα παρουσιάζει πως μπορούμε να προσθέσουμε δύο στοιχεία στη λίστα μας και ακολούθως να διαγράψουμε ένα από τα δύο αρχεία. Για την διαγράφει κάποιου στοιχείου της λίστας μας, χρησιμοποιείται η συνάρτηση g_list_remove η οποία παίρνει δύο παραμέτρους. Η πρώτη παράμετρος είναι το όνομα της λίστας μας και η δεύτερη το στοιχείο που θέλουμε να διαγράψουμε.

Το πιο κάτω παράδειγμα δείχνει πως μπορούμε να εισάγουμε δεδομένα τύπου δομής στη λίστα μας. Μερικές φορές είναι σημαντικό να εισάγουμε ένα στοιχείο στη λίστα μας που να περιέχει περισσότερα του ενός πεδία. Αρχικά δηλώνουμε τα πεδία της δομής που θέλουμε να εισάγουμε στη λίστα μας. Επίσης, πρέπει να δημιουργήσουμε και τη λίστα μας που θα είναι τύπου GSList*. Χρησιμοποιώντας την συνάρτηση g_list_append μπορούμε να εισάγουμε οποιοδήποτε δομή στη λίστα μας.

```
#include <glib.h>
typedef struct {
 char* filename;
 int tag;
} FILE;
int main(int argc, char** argv) {
 GSList* list = NULL;
 FILE * file = (FILE *)malloc(sizeof(FILE));
 file ->name = "test.txt";
 file ->tag = 12;
 list = g_slist_append(list, file);
 g_slist_free(list);
 free(file);
 return 0;
```

4.5.2 Πίνακες Κατακερματισμού

Μια από τις σημαντικότερες δομές δεδομένων είναι οι πίνακες κατακερματισμού. Ο λόγος που είναι τόσο δημοφιλείς είναι λόγο της αποδοτικής αναζήτησης δεδομένων. Οι πίνακες κατακερματισμού χρησιμοποιούν μια συνάρτηση κατακερματισμού για να αποθηκεύσουν και να αναζητήσουν δεδομένα σε χρόνο O(1).

```
#include <glib.h>
int main(int argc, char** argv)
{

 GHashTable* hash = g_hash_table_new(g_str_hash, g_str_equal);
 g_hash_table_insert(hash, "filename1", "tag1");

 printf("There are %d keys in the hash\n", g_hash_table_size(hash));
 printf("The capital of Texas is %s\n", g_hash_table_lookup(hash, "filename1"));


 g_hash_table_destroy(hash);
 return 0;
}

***** Output *****

There are 1 keys in the hash
The Tag of filename1 is tag1
```

Το πιο πάνω απλό παράδειγμα δείχνει πως μπορούμε να δημιουργήσουμε και να εισάγουμε στοιχεία στον πίνακα κατακερματισμού.

Αρχικά πρέπει να δημιουργήσουμε ένα πίνακα κατακερματισμού με τη συνάρτηση g_hash_table_new η οποία παίρνει δύο παραμέτρους. Ο τύπος του πίνακα που θα δημιουργήσουμε θα είναι GHashTable*. Η πρώτη και η δεύτερη παράμετρος αφορά τη συνάρτηση κατακερματισμού. Στη παρούσα διπλωματική θα αποθηκεύουμε χαρακτήρες οπόταν θα χρησιμοποιήσουμε την ανάλογη συνάρτηση.

Κάθε στοιχείο που εισάγουμε πρέπει να είναι συσχετισμένο με μια τιμή. Με άλλα λόγια το κάθε στοιχείο που εισάγουμε είναι της μορφής <*κλειδί, τιμή*>. Για να εισάγουμε κάποιο στοιχείο πρέπει να χρησιμοποιήσουμε την συνάρτηση g_hash_table_insert η οποία παίρνει τρείς παραμέτρους. Η πρώτη είναι το όνομα του πίνακα κατακερματισμού που έχουμε δημιουργήσει και οι υπόλοιπες δύο είναι το κλειδί και η τιμή του στοιχείου μας.

4.5.3 Δυναμικοί Πίνακες

Μια από τις πιο συνηθισμένες δομές δεδομένων είναι οι πίνακες. Το κύριο μειονέκτημα των πινάκων είναι ότι πρέπει να οριστεί εξαρχής το μέγεθος του πίνακα. Με την βοήθεια της βιβλιοθήκης Glib μπορούμε να ορίσουμε δυναμικούς πίνακες οι οποίοι αυξάνονται αυτόματα σε περίπτωση που έχει γεμίσει.

```
#include <glib.h>
int main(int argc, char** argv) {
 GArray* array = g_array_new(FALSE, FALSE, sizeof(char*));
 char* first = "img.png", *second = "img2.png "
 g_array_append_val(array, first);
 g_array_append_val(array, second);
 printf("There are now %d items in the array\n", array->len);
 printf("The first item is '%s'\n", g_array_index(array, char*, 0));
 g_array_remove_index(a, 1);
 printf("There are now %d items in the array\n", array ->len);
 g_array_free(array, FALSE);
 return 0;
}

***** Output *****
There are now 2 items in the array
The first item is 'img.png'
The third item is 'img2.png'
There are now 1 items in the array
```

Το πιο πάνω παράδειγμα παρουσιάζει πως μπορούμε να δημιουργήσουμε ένα δυναμικό πίνακα και να εισάγουμε διάφορα στοιχεία. Αρχικά πρέπει να ορίσουμε ένα πίνακα τύπου GArray*. Ακολούθως χρησιμοποιούμε την συνάρτηση g_array_append_val η οποία παίρνει δύο παραμέτρους, για να εισάγουμε οποιοδήποτε στοιχείο. Η πρώτη παράμετρος είναι το όνομα του πίνακα που έχουμε δημιουργήσει και η δεύτερη παράμετρος είναι το στοιχείο που θέλουμε να εισάγουμε. Η συνάρτηση g_array_remove_index παίρνει δυο παραμέτρους και χρησιμοποιείται για να διαγράψουμε ένα στοιχείο από τον πίνακα, σε μια συγκεκριμένη θέση. Η πρώτη παράμετρος είναι το όνομα του πίνακα και η δεύτερη η θέση που βρίσκεται το στοιχείο που θέλουμε να διαγράψουμε.

4.6 Βιβλιοθήκη GTK+

Η βιβλιοθήκη GTK+ είναι μια από τις δημοφιλέστερες βιβλιοθήκες για την ανάπτυξη διεπαφών σε προγράμματα γραμμένα στη γλώσσα προγραμματισμού. Δίνει τη δυνατότητα στο προγραμματιστή να χρησιμοποιήσει συναρτήσεις για την δημιουργία σοφιστικών και πολύπλοκων διεπαφών.

Για να μπορούμε να χρησιμοποιήσουμε την συγκεκριμένη βιβλιοθήκη πρέπει να την προσθέσουμε στον πηγαίο κώδικα ως ακολούθως:

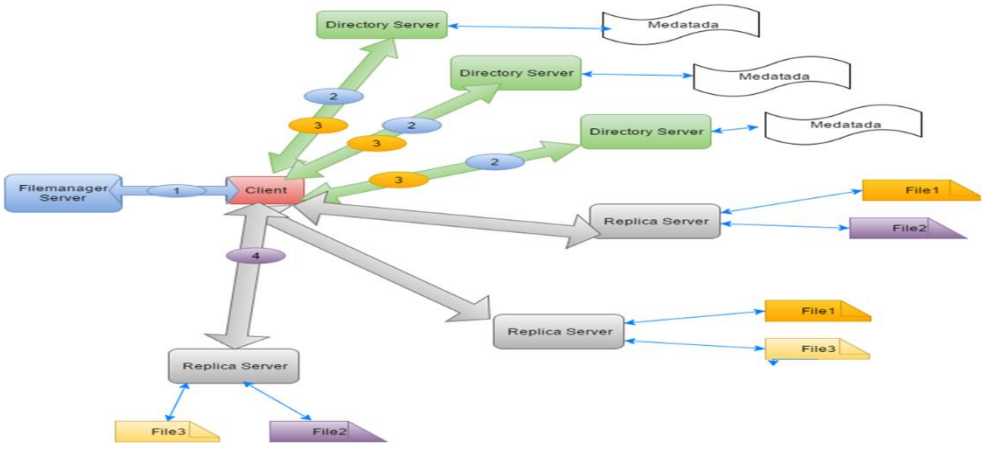
```
#include <gtk/gtk.h>
```

Επίσης κατά την μεταγλώττισης πρέπει να προσθέσουμε την πιο κάτω εντολή στον μεταγλωττιστή.

```
‘gtk+-2.0 glib-2.0`
```

4.7 Λεπτομέρειες Υλοποίησης

Η υλοποίηση του κατανεμημένου συστήματος αρχείων έγινε στο περιβάλλον Linux με την γλώσσα προγραμματισμού C και ο κώδικας του συστήματος βρίσκεται στο Παράρτημα Α. Το σύστημα αποτελείται από 4 διαφορετικές διεργασίες: Πελάτης, Directory, Replica και FileManager. Η επικοινωνία έγινε με την χρήση C sockets IP/TCP στο μοντέλο πελάτη-εξυπηρετητή. Η εφαρμογή του πελάτη επικοινωνεί με όλες τις διεργασίες. Σε αντίθεση, με τις διεργασίες Directory, Replica και FileManager που επικοινωνούν μόνο με την διεργασία του πελάτη.



Σχήμα 4.10

Αρχιτεκτονική Κατανεμημένου Συστήματος Αρχείων

Στο Σχήμα 4.10 απεικονίζεται η αρχιτεκτονική του κατανεμημένου συστήματος αρχείων. Αρχικά ο πελάτης εγκαθιδρύει σύνδεση με όλες τις διεργασίες. Αξίζει εδώ να αναφέρουμε ότι η σύνδεση με όλες τις διεργασίες παραμένει ενεργή μέχρι ο πελάτης να εγκαταλείψει το σύστημα. Ο κύριος λόγος που ακολουθείται η προσέγγιση αυτή είναι για λόγους επίδοσης.

Ο πελάτης επικοινωνεί πρώτα με το filemanager για να ενημερωθεί για τα διαθέσιμα αρχεία του συστήματος. Ακολούθως, ζητά από το filemanager να μάθει πιο είναι το αναγνωριστικό του αρχείου που θέλει να διαβάσει. Η επικοινωνία με τις υπόλοιπες διεργασίες γίνεται με το αναγνωριστικό του αρχείου.

Στη περίπτωση όπου ο πελάτης θέλει να διαβάσει κάποιο αρχείο επικοινωνεί πρώτα με όλους τους directories και περιμένει ένα quorum από απαντήσεις. Ο λόγος που περιμένει ένα quorum από απαντήσεις είναι για να μπορούμε να εγγυηθούμε ότι θα διαβάσει το πιο πρόσφατο αρχείο. Το μήνυμα που λαμβάνει από τους directories είναι στη πραγματικότητα ποιοι replica κρατάνε το τελευταίο αρχείο. Πριν ζητήσει όπως το πραγματικό αρχείο πρέπει να ξανά επικοινωνήσει με όλους τους directories και να τους ενημερώσει πιο είναι το πιο πρόσφατο αρχείο που έχει λάβει. Ο λόγος που γίνεται αυτό το επιπλέον βήμα είναι για λόγους συνέπειας όπως έχει εξηγηθεί προηγουμένως κατά την περιγραφή των αλγορίθμων ABD και LDR.

Όταν λάβει ένα quorum από απαντήσεις κατά τη δεύτερη φάση επικοινωνίας με τους directories μπορεί τώρα να επικοινωνήσει με τους replicas και να τους ζητήσει το αρχείο. Οι πληροφορίες σχετικά με το πιο πρόσφατο αρχείο καθώς επίσης και σε ποιους replica είναι αποθηκευμένο, βρίσκονται στο μήνυμα που έχει λάβει από τους directories.

Ο πελάτης θα επικοινωνήσει μόνο με ένα από τους διαθέσιμους replica που έχουν το πιο πρόσφατο αρχείο. Σε περίπτωση όπου κάποιος replica έχει καταρρεύσει τότε ζητά από τον επόμενο διαθέσιμο replica το αρχείο.

4.7.1 Βιβλιοθήκες

Οι δομές δεδομένων που έχουν χρησιμοποιηθεί στο σύστημα έχουν γίνει με την βοήθεια της βιβλιοθήκης glib. Η βιβλιοθήκη glib παρέχει βασικές δομές δεδομένων υλοποιημένες στη γλώσσα προγραμματισμού C, όπως συνδεδεμένες λίστες και πίνακες κατακερματισμού. Οπόταν για να γίνει η μεταγλώττιση του συστήματος πρέπει να υπάρχει εγκατεστημένη η πιο κάτω βιβλιοθήκη.

```
>> sudo apt-get install libgtk-3-dev
```

Κατά την μεταγλώττιση του συστήματος πρέπει να εισαχθεί η πιο κάτω παράμετρος.

```
`pkg-config --cflags --libs glib-2.0`
```

4.7.2 Εγκαθίδρυση επικοινωνίας

Κατά την σύνδεση του πελάτη στο Κ.Σ.Α, όλες οι απαραίτητες πληροφορίες που πρέπει να γνωρίζει το πρόγραμμα του πελάτη βρίσκονται στο ‘config.txt’. Το δίκτυο είναι στατικό, επομένως όλες οι διευθύνσεις και θύρες δεν χρειάζεται να τροποποιηθούν. Ο πελάτης συνδέεται με τον διαχειριστεί αρχείων, εξυπηρετητή ευρετηρίου και αρχείων. Επίσης, η σύνδεση με τους εξυπηρετητές θα γίνει μόνο μια φορά για κάθε πελάτη και η αποσύνδεση θα γίνει μόνο στη περίπτωση όπου ο πελάτης θα εγκαταλείψει το σύστημα.

4.7.3 Πελάτης

Ο ρόλος του πελάτη στο σύστημα είναι να διαβάζει , γράφει και τροποποιεί τα αρχεία του συστήματος. Για να διαβάσει ένα αρχείο πρέπει πρώτα να επικοινωνήσει με τον διαχειριστή αρχείων έτσι ώστε να μάθει πιο είναι το αναγνωριστικό του αρχείου. Ακολούθως, επικοινωνεί με τους εξυπηρετητές ευρετηρίου για να ενημερωθεί για την τελευταία έκδοση του αρχείου και για το σύνολο των εξυπηρετητών αρχείων που έχουν την τελευταία έκδοση. Για την ολοκλήρωση αυτής της φάσης, πρέπει να περιμένει μέχρι να πάρει απαντήσεις από μία πλειοψηφία από εξυπηρετητές, έτσι ώστε να είναι σίγουρος ότι θα παραλάβει την τελευταία ολοκληρωμένη γραφή. Σε αντίθετη περίπτωση, επιστρέφει στον χρήστη ότι δεν μπορεί να εκτελεστεί η λειτουργία που ζήτησε. Έπειτα, επικοινωνεί με ένα από τους εξυπηρετητές αρχείων οι οποίοι κρατάνε το αρχείο και ζητά να του στείλει το αρχείο.

Για το γράψιμο ενός αρχείου, ο πελάτης επικοινωνεί με τον διαχειριστή αρχείων για να μάθει το αναγνωριστικό του αρχείου που θέλει να γράψει στο σύστημα. Στην περίπτωση όπου είναι η πρώτη φορά που θα γράψει αυτό το αρχείο, τότε ο διαχειριστής αρχείων θα δημιουργήσει ένα καινούργιο μοναδικό αναγνωριστικό για το αρχείο και θα το στείλει στον πελάτη. Ακολούθως, πρέπει να επικοινωνήσει με τους διαχειριστές αρχείων για να μάθει το πιο πρόσφατο αναγνωριστικό του αρχείο και να το αυξήσει κατά ένα. Ακολούθως, επικοινωνεί με ένα σταθερό αριθμό από εξυπηρετητές αρχείων για να γράψει το αρχείο. Ο σταθερός αριθμός από εξυπηρετητές αρχείων ορίζεται με βάση το ποσοστό f και είναι μια πρόβλεψη πόσοι επεξεργαστές ενδέχεται να καταρρεύσουν.

4.7.4 Εξυπηρετητής Ευρετηρίου

Ο εξυπηρετητής ευρετηρίου έχει επικοινωνία μόνο με τον πελάτη και σε καμία περίπτωση δεν επικοινωνεί με κάποιο άλλο εξυπηρετητή. Επίσης, είναι υπεύθυνος να κρατά τα μεταδεδομένα των αρχείων. Όλες οι πληροφορίες σχετικά με τα αρχεία είναι αποθηκευμένες σε ένα πίνακα. Με τον όρο πληροφορίες εννοούμε πεδία όπως: αναγνωριστικό αρχείου , όνομα αρχείου, αναγνωριστικό έκδοσης αρχείου και δικαιώματα αρχείων. Παράλληλα, γνωρίζει σε ποιους εξυπηρετητές αρχείων βρίσκονται τα πραγματικά δεδομένα.

Κατά την εκκίνηση του εξυπηρετητή ευρετηρίου γίνονται κάποιες αρχικοποιήσεις μεταβλητών. Έπειτα, καθορίζονται οι συνδέσεις στις οποίες θα δέχεται αιτήσεις από τους πελάτες. Ο εξυπηρετητής ευρετηρίου μπορεί να εξυπηρετεί ένα μεγάλο αριθμό από πελάτες ταυτόχρονα. Για να μπορεί όμως να γίνεται η ταυτόχρονη εξυπηρέτηση αποδοτικά, γίνεται η χρήση νημάτων (pthreads). Για κάθε ένα νέο αίτημα, δημιουργεί ένα καινούργιο νήμα το οποίο είναι υπεύθυνο να εξυπηρέτηση ένα πελάτη.

Στο πιο κάτω σχήμα παρουσιάζεται ένας απλός ψευδοκώδικας για τη εξυπηρέτηση πολλαπλών πελατών ταυτόχρονα από τον εξυπηρετητή ευρετηρίου.

```
//Ανάμενε αιτήσεις από πελάτες
while(1)
{
    clientPtr=(struct sockaddr *) &client_addr;
    clientlen= sizeof(client_addr);

  //Περιμένει μέχρι να δεκτή κάποια αίτηση από κάποιο πελάτη
 if((newsock=accept(sock , clientPtr  , &clientlen )) < 0)
    {
        perror("accept() failed"); exit(1);
    }
      //Για κάθε νέο πελάτη δημιουργείται ένα καινούργιο νήμα το
    // οποίο εκτελεί τη συνάρτηση accept_thread
       if(err=pthread_create(&tid , NULL , &accept_thread ,  (void *)(intptr_t) newsock))
    {
        perror2("pthread_create for accept_thread" , err);
        exit(1);
    }

}//While(1)
```

Σχήμα 4.11

Ψευδοκώδικας για εξυπηρέτηση πελατών

Το πιο πάνω Σχήμα 4.11 αντιπροσωπεύει τον ψευδοκώδικα για την εξυπηρέτηση των πελατών. Όπως έχει αναφερθεί και προηγουμένως, για να επιτευχθεί η ταυτόχρονη εξυπηρέτηση των πελατών γίνεται η χρήση νημάτων. Αρχικά, η συνάρτηση accept() μπλοκάρεται προσωρινά μέχρι κάποιος πελάτης να επικοινωνήσει με τον εξυπηρετητή ευρετηρίου. Έπειτα, με τη αίτηση κάποιου πελάτη για να επικοινωνία με τον εξυπηρετητή , δημιουργείται ένα νήμα με την συνάρτηση pthread_create(). Η συνάρτηση pthread_create() καλεί την συνάρτηση accept_thread() , η οποία είναι υπεύθυνη για την εξυπηρέτηση του πελάτη. Το δεύτερο νήμα επιστρέφει στη συνάρτηση accept() για να μπορέσει να εξυπηρετήσει ενδεχόμενους πελάτες που θα προκύψουν.

Στο πιο κάτω σχήμα παρουσιάζεται ένας απλός ψευδοκώδικας για την εύρεση του πιο πρόσφατου αναγνωριστικού για ένα συγκεκριμένο αρχείο.

```
//Συνάρτηση IsMaxTag
int IsMaxTag(struct message *tag2)
{
            //Εύρεση θέσης πληροφοριών αρχείου
            int index=findByFileID(tag2 , &funfreepos);

            //Ανάκτηση πληροφοριών του αρχείου
            point2metadata = (struct metadata *) g_ptr_array_index(metatable , index );

            if ((point2metadata->tag.num < tag2->tag.num) \
                                                        || (point2metadata->tag.num == tag2-
                        >tag.num && point2metadata->tag.id < tag2->tag.id ))
            {
                point2metadata->tag.num=tag2->tag.num;
                point2metadata->tag.id=tag2->tag.id;

                //Remove old replica set because now is unnecessary due to
                //the tag is old one
                g_slist_free ( point2metadata->replicaSet );
                point2metadata->replicaSet=NULL;

                //Point to new replica set with the latest tag
                point2metadata->replicaSet=tag2->replicaSet;

               isFound=1;
            }

                 else if(point2metadata->tag.num == tag2->tag.num && point2metadata->tag.num ==  tag2->tag.id
                 )
            {
                point2metadata->replicaSet = insertList( point2metadata->replicaSet , tag2->replicaSet);
            }

    return isFound;
}
```

Σχήμα 4.12

Ψευδοκώδικας Εύρεσης του πιο πρόσφατου αναγνωριστικού αρχείου

Ο ψευδοκώδικας στο πιο πάνω Σχήμα 4.12 αντιπροσωπεύει τη συνάρτηση IsMaxTag() η οποία συγκρίνει αν το καινούργιο αναγνωριστικό αρχείου είναι μεγαλύτερο από το υφιστάμενο. Στην περίπτωση όπου είναι μεγαλύτερο τότε το παλιό πρέπει αν αντικατασταθεί με το καινούργιο. Ο πίνακας metatable είναι ένας πίνακας στον οποίο αποθηκεύονται όλες οι πληροφορίες σχετικά με τα αρχεία. Αρχικά, η συνάρτηση αναζητά τη θέση του αρχείου μέσα στο πίνακα. Μόλις βρει τη θέση ανακτά τις πληροφορίες αυτές για να ελέγξει αν η έκδοση του αρχείου είναι μεγαλύτερη.

Οι τύποι μηνύματος όπου ένας εξυπηρετητής αρχείων μπορεί να παραλάβει είναι ένας από τους εξής:

- *<rread,msgid,filename,fileid,permission>*
- *<rread-ok,tag.num,tag.id,repliset,msgid,fileid>*
- *<rwrite,repliset,tag.num,tag.clientid,msgid,filename>*
- *<rread,repliset,tag.num,tag.clientid,msgid,filename>*

Κατά την λειτουργία ανάγνωσης (read) ο πελάτης αρχικά στέλνει ένα μήνυμα της μορφής *<rread,msgid,filename,fileid,permission>*, όπου *msgid* είναι ένας αύξων αριθμός για να γνωρίζει ο πελάτης ότι έχει πάρει απάντηση για το σωστό μήνυμα που έχει σταλθεί. Είναι σημαντικό εδώ να αναφέρουμε ότι το σύστημα μας τρέχει και σε ασύγχρονο δίκτυο , άρα είναι λογικό κάποια μηνύματα να χάνονται. Το *filename* αντιπροσωπεύει το όνομα του αρχείου και το *fileid* είναι ένας αύξων μοναδικός αριθμός για κάθε αρχείο. Για να μπορεί να επιστρέψει σε ποιους εξυπηρετητές αρχείων βρίσκεται το αρχείο και πιο είναι το πρόσφατο συσχετισμένο αναγνωριστικό έκδοσης, πρέπει πρώτα να ελεγχθεί ότι ο συγκεκριμένος πελάτης έχει δικαίωμα ανάγνωσης για το συγκεκριμένο αρχείο. Αυτό επιτυγχάνεται με το να συγκρίνει το permissions του χρήστη μαζί με του αρχείου. Αν έχει δικαίωμα, τότε επιστρέφει ένα μήνυμα της μορφής *<rread-ok,tag.num,tag.id,repliset,msgid,fileid>* όπου *tag.num* και *tag.id* είναι τα αναγνωριστικά του αρχείου. Το *replicaset* αντιπροσωπεύει ποιοι εξυπηρετητές αρχείων κρατάνε το πιο πρόσφατο αρχείο. Τέλος, το *msgID* και το *fileID* είναι για να γνωρίζει ο πελάτης για πιο μήνυμα και αρχείο είναι το μήνυμα.

4.7.5 Εξυπηρετητής Αρχείων

Ο ρόλος του εξυπηρετητή αρχείων στο σύστημα είναι να αποθηκεύει πολλαπλές εκδόσεις των πραγματικών αρχείων. Κάθε αρχείο συνοδεύεται με ένα αναγνωριστικό αρχείου για να γνωρίζει ο εξυπηρετητής πόσο πρόσφατο είναι το κάθε αρχείο. Επίσης, μπορεί να παραλαμβάνει ή να στέλνει αρχεία.

Η μεταφορά αρχείων έγινε με την χρήση του πρωτοκόλλου TCP/IP και με την χρήση της συνάρτησης ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count). Η συνάρτηση sendfile είναι κατάλληλη για την μεταφορά μεγάλων αρχείων, καθώς είναι πιο αποδοτική σε σύγκριση με τις read και write συναρτήσεις. Η αποδοτικότητα της sendfile είναι στο γεγονός

ότι δεν αντιγράφει το αρχείο που πρέπει να σταλεί στο χώρο του χρήστη, αλλά το στέλνει απευθείας στην υποδοχή. Αξίζει εδώ να αφιερώσουμε λίγο χρόνο για να εξηγήσουμε πως έχει γίνει η μεταφορά των αρχείων αποδοτικά. Υπάρχουν διάφορα πρωτόκολλα που θα μπορούσαν να χρησιμοποιηθούν για τη μεταφορά αρχείων. Μερικά από αυτά είναι το FTP (File transfer protocol) ή το HTTP. Το FTP πρώτο πρωτόκολλο, είναι πολύ αποδοτικό για τη μεταφορά μεγάλων αρχείων. Το δεύτερο είναι το βασικό πρωτόκολλο που χρησιμοποιείται στο παγκόσμιο ιστό. Τόσο για τη μεταφορά σελίδων καθώς επίσης και για τα περιεχόμενα που υπάρχουν σε μια σελίδα όπως πχ. αρχεία. Στη παρούσα διπλωματική εργασία έχει σχεδιαστεί ένα πρωτόκολλο το οποίο τρέχει πάνω στο TCP/IP. Ο λόγος που δεν έχει γίνει χρήση των προαναφερθέντων πρωτοκόλλων είναι λόγο του ότι υπήρχε η ανάγκη να προσαρμοστεί σύμφωνα με τις ανάγκες του συστήματος.

```
//Ανάκτηση μεγέθους αρχείου από τον πελάτη
remain_data = file_stat.st_size;

 while (((sent_bytes = sendfile(newsock, fd, &offset, BUFSIZE)) > 0) && (remain_data > 0))
 {
     remain_data -= sent_bytes;
    fprintf(stdout, "Server sent %d bytes from file's data, offset is now : %d and
remaining                 data = %d\n", sent_bytes, offset, remain_data);
 }

 //Κλείσιμο περιγραφέα αρχείου
 close(fd);
```

Σχήμα 4.13

Ψευδοκώδικας Αποστολής Αρχείων

Στο Σχήμα 4.13 παρουσιάζεται ο ψευδοκώδικας για την αποστολή αρχείων. Αρχικά, πρέπει να καθορίσουμε τον περιγραφέα αρχείου. Έπειτα, χρησιμοποιώντας τη συνάρτηση sendfile αρχίζουμε να στέλνουμε το αρχείο στην υποδοχή που έχουμε ανοιχτή με τον πελάτη, μέχρι να σταλεί όλο το μέγεθος του αρχείου.

Στην τοπική του μνήμη ο εξυπηρετητής έχει αποθηκευμένο για κάθε αρχείο: το όνομα του αρχείου, το αναγνωριστικό αρχείου, τη έκδοση του αρχείου, και το secure bit που είναι συσχετισμένο με το κάθε αρχείο. Το πεδίο ορισμού του secure bit είναι {0,1}. Στην περίπτωση όπου μια πλειοψηφία από εξυπηρετητές ευρετηρίου γνωρίζει για το αρχείο αυτό τότε το secure bit του συγκεκριμένου αρχείου είναι 1. Σε αντίθετη περίπτωση είναι 0. Αξίζει εδώ να αναφερθεί ότι παρόλο που ο κάθε εξυπηρετητής έχει αποθηκευμένες πολλαπλές εκδόσεις του κάθε αρχείου, όταν πάρει ένα μήνυμα της μορφής <*secure, tag.num,tag.id,msgid,fileid,filename,filetype*> κάνει το secure bit του αρχείου με το συγκεκριμένο όνομα αρχείου να ισούται με 1. Ακολούθως, διαγράφει όλες τις παλαιότερες εκδόσεις του αρχείου που έχουν μικρότερη αναγνωριστικό έκδοση αρχείου.

Κατά την εκκίνηση της εφαρμογής, δέχεται σαν ορίσματα γραμμής εντολών τα πιο κάτω:

```
./replica [port number]
```

όπου replica είναι το όνομα του εκτελέσιμου αρχείου και port number είναι η θύρα που θα ακούει για εισερχόμενες αιτήσεις σύνδεσης.

4.7.6 Διαχειριστής Αρχείων

Η κύρια αρμοδιότητα του διαχειριστή αρχείων είναι να δημιουργεί για κάθε καινούργιο αρχείο στο σύστημα ένα μοναδικό αναγνωριστικό. Ακολούθως, ενημερώνει τον πελάτη ποιο είναι το αναγνωριστικό έτσι ώστε ο πελάτης να επικοινωνεί με τους υπόλοιπους εξυπηρετητές χρησιμοποιώντας το αναγνωριστικό αυτό. Επιπλέον, αναθέτει ένα μοναδικό αναγνωριστικό για κάθε πελάτη έτσι ώστε να αναγνωρίζει μοναδικά τον κάθε πελάτη και να αποθηκεύει τον δημιουργό του κάθε αρχείου.

Κατά την εκκίνηση της εφαρμογής, δέχεται σαν ορίσματα γραμμής εντολών τα πιο κάτω:

```
./fil [port number]
```

Όπου fil είναι το όνομα του εκτελέσιμου αρχείου και port number είναι η θύρα που θα ακούει για εισερχόμενες αιτήσεις σύνδεσης.

Οι τύποι μηνύματος πού παραλαμβάνει ο διαχειριστής αρχείων είναι οι εξής:

- <*reqclientid,username,msgid*>
- <*reqcreate,filename,owner,msgid*>
- <*reqfileid,filename,owner,msgid*>
- <*reqfilelist, msgid*>

Κατά την παραλαβή του μηνύματος *reqclientid* ο διαχειριστής αρχείων δημιουργεί ένα μοναδικό αναγνωριστικό για το συγκεκριμένο username και ενημερώνει τον πελάτη πιο είναι το αναγνωριστικό που του έχει αναθέσει.

Κατά την παραλαβή του μηνύματος *reqcreate* δημιουργεί ένα μοναδικό αναγνωριστικό για το συγκεκριμένο όνομα αρχείου. Ακολούθως, ενημερώνει τον πελάτη με το πιο είναι το αναγνωριστικό του αρχείου για να μπορεί να επικοινωνεί με τους υπόλοιπους εξυπηρετητές χρησιμοποιώντας το αναγνωριστικό αυτό. Κατά την παραλαβή του μηνύματος *reqfileid*, ο διαχειριστής αρχείων εντοπίζει το αναγνωριστικό για το συγκεκριμένο όνομα αρχείου και το επιστρέφει στον πελάτη.

Τέλος, κατά την παραλαβή του μηνύματος *reqfilelist* ενημερώνει τον πελάτη με το ποια αρχεία υπάρχουν στο σύστημα μαζί με τα σχετιζόμενα αναγνωριστικά τους.

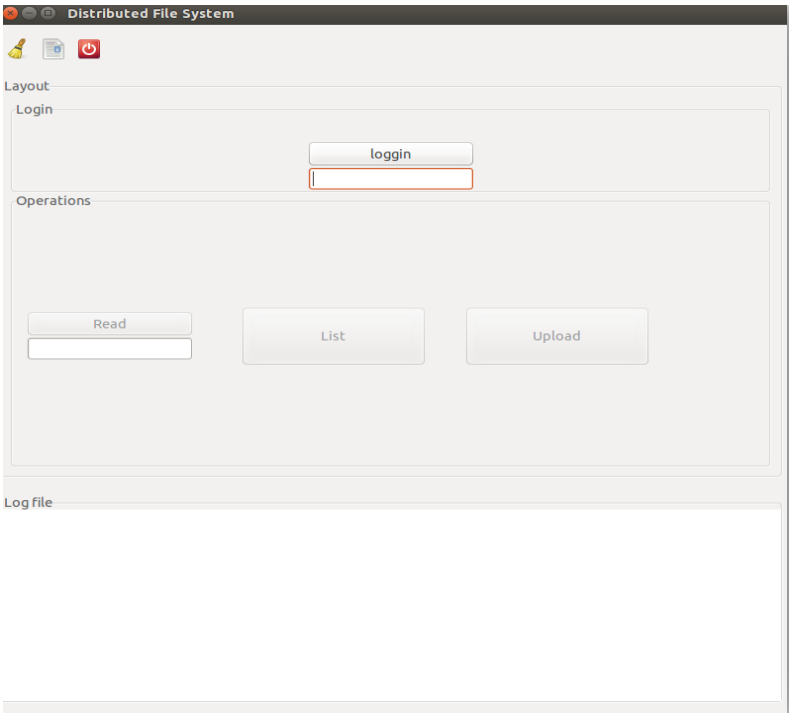
Κεφάλαιο 5

Χρήση Συστήματος: Μια Ανασκόπηση

Σε αυτό το κεφάλαιο θα παρουσιάσουμε την βασική διεπαφή του συστήματος. Επίσης θα δείξουμε πως μπορούμε να εκτελέσουμε βασικές λειτουργίες του συστήματος χρησιμοποιώντας την διεπαφή.

5.1 Έναρξη συστήματος

Αρχικά πρέπει να γίνει η εκκίνηση όλων των συστατικών του συστήματος. Η εκκίνηση των συστατικών του συστήματος, μπορεί να γίνει τοπικά ή απομακρυσμένα. Ακολούθως θα μπορεί να γίνει η χρήση της διεπαφής για την επιτυχή χρήση του συστήματος.

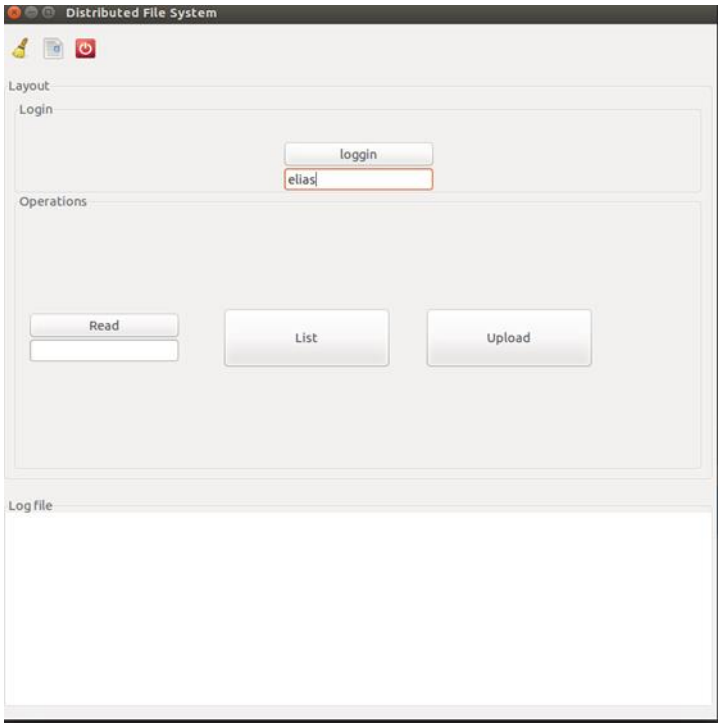


Σχήμα 5.1

Βασική διεπαφή συστήματος

5.2 Εκκίνηση Διεπαφής

Για να γίνει η εκκίνηση της διεπαφής πρέπει να τρέξουμε το εκτελέσιμο αρχείο ‘interf’ που βρίσκεται στο φάκελο ‘Interface’. Όλες οι παράμετροι σχετικά με τους εξυπηρετητές του συστήματος βρίσκονται το αρχείο ‘conf.txt’.

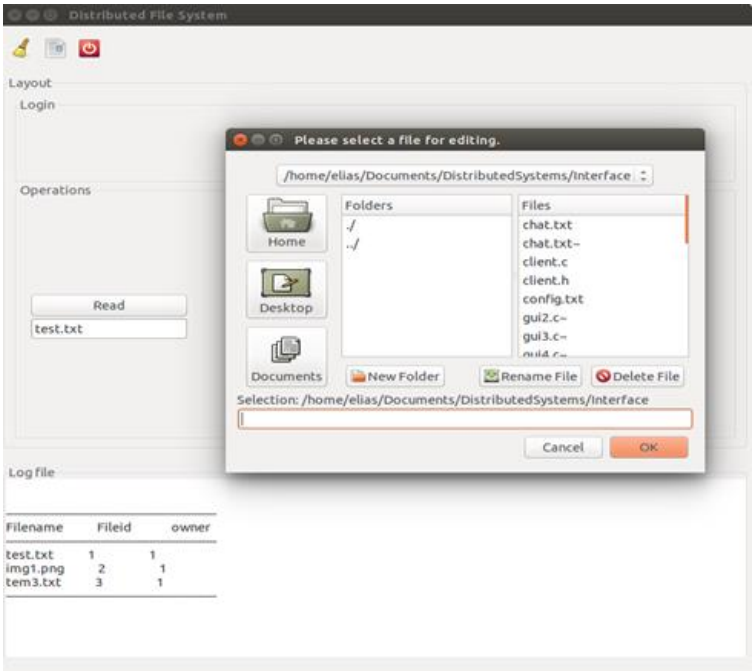


Σχήμα 5.2

Αρχική διεπαφή συστήματος

Αρχικά για να γίνει η σύνδεση του χρήστη στο σύστημα πρέπει να γίνει η εισαγωγή κάποιου username.

5.3 Μεταφόρτωση αρχείων

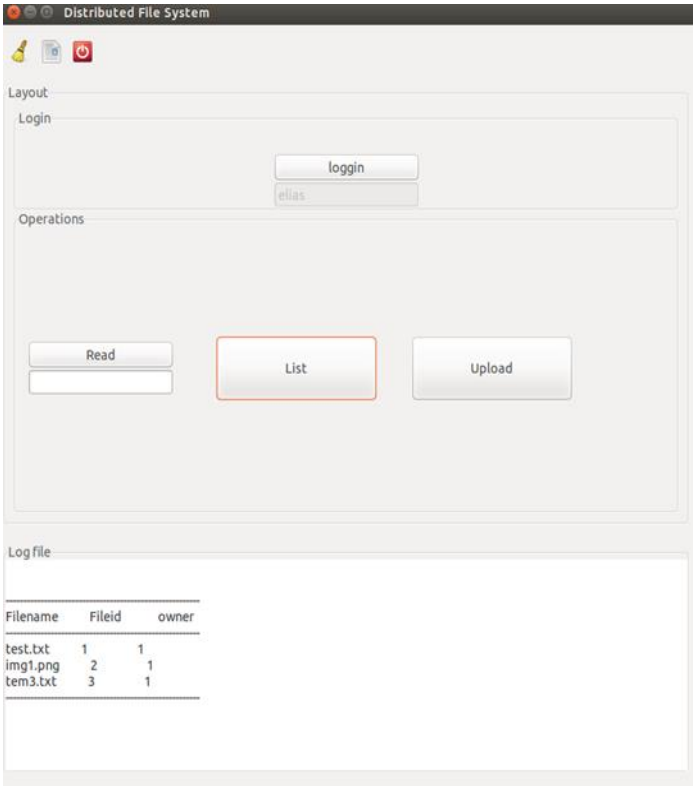


Σχήμα 5.3

Μεταφόρτωση αρχείου

Για να γίνει η μεταφόρτωση κάποιου αρχείου στο σύστημα πρέπει να επιλεχθεί το κουμπί upload. Ακολούθως, θα εμφανιστεί ένα παράθυρο για να επιλέξουμε το αρχείου το οποίο θα ανεβάσουμε. Όταν διαλέξουμε το αρχείο πρέπει να επιλέξουμε το κουμπί ok για να ανεβεί στο σύστημα. Θα πάρει κάποιο χρόνο, ανάλογα με το μέγεθος του αρχείου για να ανεβεί στο σύστημα. Μπορούμε να ελέγξουμε αν το αρχείο έχει ανεβεί επιτυχώς ή αν υπάρχει κάποια σύγκρουση σε περίπτωση όπου προσπαθήσαμε να ανεβάσουμε μια παλιά έκδοση του αρχείου. Τα αντίστοιχα μηνύματα θα εμφανιστούν στο log file view της διεπαφής.

5.4 Αναζήτηση Αρχείων

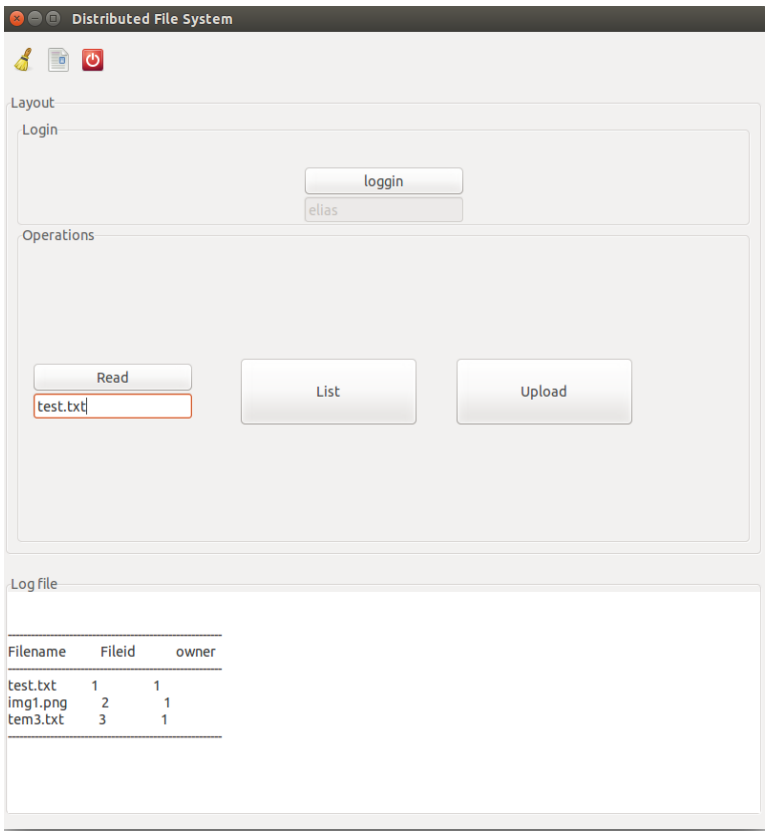


Σχήμα 5.4

Αναζήτηση Αρχείων

Για να δούμε πια αρχεία υπάρχουν στο σύστημα πρέπει να επιλέξουμε το κουμπί list. Έπειτα, θα εμφανιστή μια λίστα με τα διαθέσιμα αρχεία του συστήματος στο log file view της διεπαφής.

5.5 Λειτουργία Ανάγνωσης Αρχείου



Σχήμα 5.5

Ανάγνωση Αρχείου

Αν γνωρίζουμε πιο αρχείο θέλουμε να διαβάσουμε από το σύστημα, μπορούμε να εισάγουμε το όνομα του αρχείου στο πλαίσιο κειμένου που βρίσκεται κάτω από το κουμπί read. Έπειτα, πρέπει να επιλέξουμε το κουμπί read για να εκτελεστή η λειτουργεία read. Το αρχείο θα μεταφερθεί από το σύστημα αρχείων στο φάκελο που βρίσκεται η διεπαφή.

Κεφάλαιο 6

Πειραματική Αξιολόγηση

Η παρούσα πειραματική αξιολόγηση γίνεται περισσότερο ως “Proof of Concept”. Το σύστημα έχει ελεγχθεί και αποσφαλματωθεί σε μεγάλο βαθμό κατά τη διάρκεια της ανάπτυξης του, οπότε αυτή η πειραματική αξιολόγηση είναι περισσότερο για να δώσει μια προκαταρκτική εκτίμηση της απόδοσης του συστήματος σε ένα πραγματικό περιβάλλον.

6.1 Προετοιμασία

Κατ’ αρχήν για να ελεγχθεί η ορθή λειτουργία του κατανεμημένου συστήματος αρχείων έχουν δοκιμαστεί διάφορα σενάρια, με διαφορετικό αριθμό ταυτόχρονων γραφέων και αναγνωστών. Για την συλλογή των δεδομένων έγιναν τα αντίστοιχα scripts για να υλοποιηθούν τα σενάρια και έχει γίνει η χρονομέτρηση του χρόνου ολοκλήρωσης της κάθε λειτουργίας. Το κάθε σενάριο εκτελείται 15 φορές έτσι ώστε να υπολογίσουμε τον μέσο όρο. Οι λεπτομέρειες του κάθε σεναρίου όπως ποιες παράμετροι χρησιμοποιήθηκαν αναφέρονται λεπτομερώς στα επόμενα κεφάλαια.

6.2 Πλατφόρμα Υλοποίησης

Το Κατανεμημένο σύστημα αρχείων τρέχει σε ηλεκτρονικούς υπολογιστές του πανεπιστημίου Κύπρου με Intel® 3.40 GHz, σε λειτουργικό Centos 6.7 και 4GB RAM. Αξίζει να αναφέρουμε το πρόγραμμα του πελάτη έχει τρέξει σε ένα ηλεκτρονικό υπολογιστή με Intel® 3.40 GHz, λειτουργικό σύστημα Ubuntu 15.04 , 12GB RAM. Επίσης να αναφέρουμε ότι τα πρώτα δύο σενάρια έχουν πραγματοποιηθεί σε δίκτυο WAN (Wide Area Network) δηλαδή στο Διαδίκτυο όπου η ταχύτητα αποστολής ήταν 8Mbps και η ταχύτητα παραλαβής 4Mbps. Το τρίτο σενάριο που αφορούσε διάφορα μεγέθη αρχείων έγινε σε δίκτυο LAN (Local Area Network) δηλαδή σε τοπικό δίκτυο όπου η ταχύτητα αποστολής ήταν 100Mbps και η ταχύτητα παραλαβής 80Mbps. Να αναφέρουμε επίσης ότι οι εξυπηρετητές του συστήματος τόσο για τα σενάρια που πραγματοποιήθηκαν σε δίκτυο WAN και LAN ήταν οι ίδιοι. Η μόνη διαφορά ήταν ότι ο

πελάτης κατά τα σενάρια σε δίκτυο WAN ήταν ενωμένος στο σύστημα διαμέσου του Διαδικτύου ενώ στο σενάριο που έτρεξε σε δίκτυο LAN ο πελάτης ήταν συνδεδεμένος στο σύστημα διαμέσου του τοπικού δικτύου, συνεπώς η ταχύτητα ήταν πολύ μεγαλύτερη.

6.3 Σενάρια

Πιο κάτω παρουσιάζονται τα χαρακτηριστικά των τριών σεναρίων που έχουν πραγματοποιηθεί. Πιο συγκεκριμένα αναφέρεται από πόσους εξυπηρετητές αποτελείται το Κ.Σ.Α και πόσοι αναγνώστες ή εγγραφείς εμπλέκονται στο κάθε σενάριο.

Σενάριο1- Αναγνωστών

Το σενάριο αυτό εκτελέστηκε με 10, 20, 30 και 40 ταυτόχρονους αναγνώστες. Σε αυτό το σενάριο, το σύστημα αποτελείτε από 1 διαχειριστή αρχείων και 3 εξυπηρετητές ευρετηρίου και αρχείων. Το αρχείο είναι τύπου '.png' και ήταν μεγέθους 150KB. Ο παράγοντας *f* είναι 2. Η λογική που έγινε αυτό το σενάριο ήταν για να ελεγχθεί η συμπεριφορά του συστήματος στην περίπτωση που εξυπηρετεί ταυτόχρονα πολλούς πελάτες. Επίσης, για να ελεγχθεί η συνέπεια του συστήματος. Για το λόγο ότι αρχικά το σύστημα δεν έχει οποιοδήποτε αρχείο αποθηκευμένο θα εκτελεστεί μια εγγραφή για να αποθηκεύσει το αρχείο που θα διαβάσουν οι αναγνώστες στο σύστημα.

Σενάριο 2 – Εγγραφείς

Στο σενάριο αυτό ο στόχος είναι να ελέγξουμε αν επηρεάζεται η συνέπεια και η απόδοση του συστήματος όταν υπάρχουν ταυτόχρονοι εγγραφείς που προσπαθούν να γράψουν το ίδιο αρχείο. Στο σενάριο αυτό χρησιμοποιήθηκαν 10, 20, 30 και 40 ταυτόχρονοι εγγραφείς. Η διαμόρφωση του συστήματος αποτελείται από 1 διαχειριστή αρχείων, 3 εξυπηρετητές αρχείων και ευρετηρίων. Ο παράγοντας *f* ήταν σταθερός και ίσος με 2.

Σενάριο 3 – Μέγεθος Αρχείων

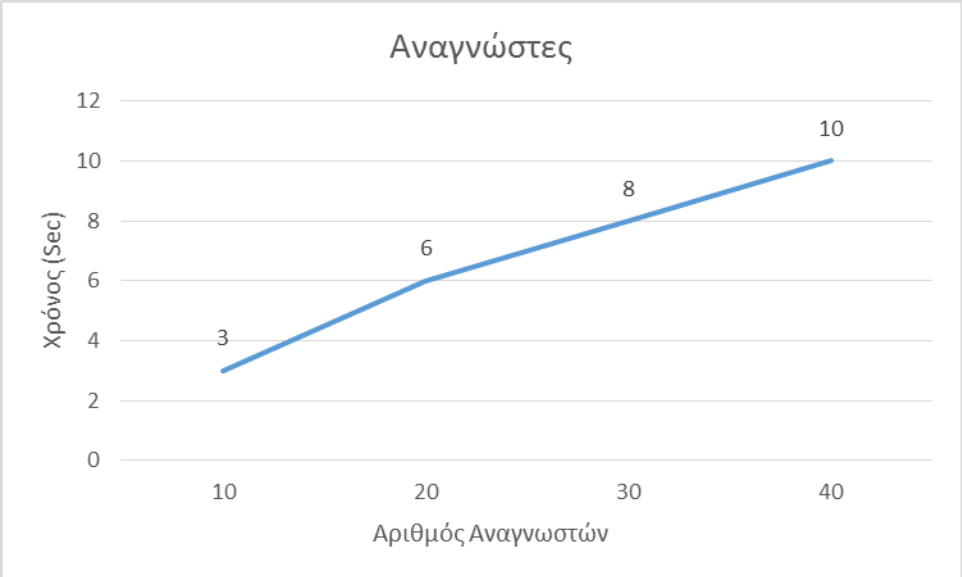
Στο σενάριο αυτό χρησιμοποιούνται δύο εγγραφείς και σκοπός μας είναι να ελέγξουμε την συνέπεια και απόδοση του συστήματος, τόσο για μικρού μεγέθους αρχεία αλλά και για μεγάλου μεγέθους. Σε αντίθεση με τα προηγούμενα σενάρια που η αξιολόγηση δοκιμάστηκε σε δίκτυο WAN, σε αυτό το σενάριο η αξιολόγηση έγινε σε δίκτυο LAN. Σε αυτό το σενάριο μας ενδιέφερε να δούμε πως θα συμπεριφερθεί το σύστημα σε μεγάλα αρχεία. Η διαμόρφωση του συστήματος αποτελείται από 1 διαχειριστή αρχείων , 3 εξυπηρετητές ευρετηρίων και αρχείων.

6.4 Αποτελέσματα

Πιο κάτω παρουσιάζονται τα αποτελέσματα του κάθε σεναρίου σε μορφή γραφικής παράστασης. Επιπλέον γίνεται επεξήγηση των αποτελεσμάτων για το κάθε σενάριο.

Σενάριο 1 - Αναγνώστες

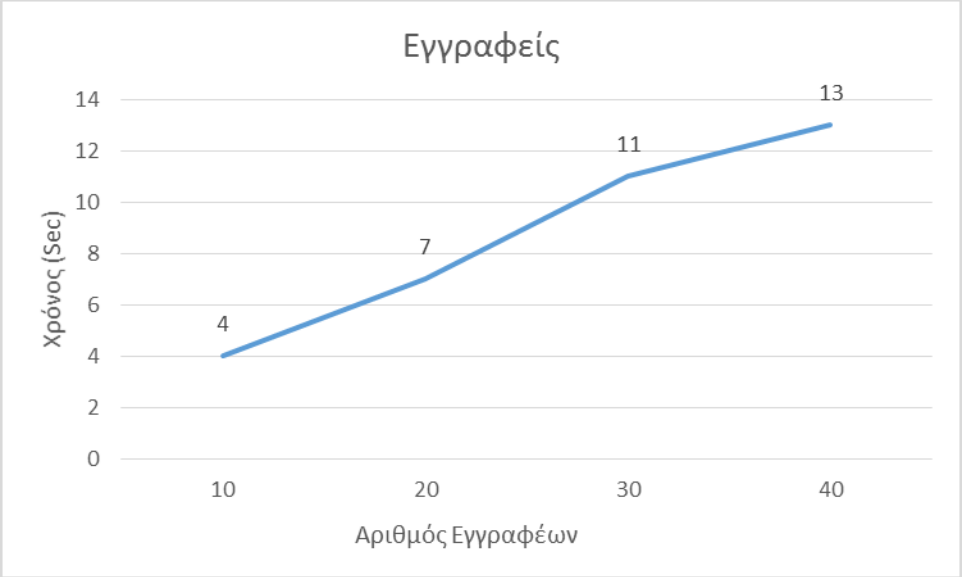
Αναμένεται πως ο χρόνος ολοκλήρωσης μιας λειτουργίας ανάγνωσης θα αυξάνεται όσο αυξάνονται οι αναγνώστες στο σύστημα. Αυτό είναι απόλυτα φυσιολογικό αφού με περισσότερους πελάτες, αυξάνεται ο φόρτος στους εξυπηρετητές. Συνεπώς, οι εξυπηρετητές θα χρειάζονται περισσότερο χρόνο για να εξυπηρετούν το κάθε πελάτη. Στο πιο κάτω σχήμα παρουσιάζεται ο μέσος χρόνος που χρειάζεται να ολοκληρωθεί μια λειτουργία ανάγνωσης. Στον οριζόντιο άξονα βλέπουμε τον αριθμό των αναγνωστών που λαμβάνουν μέρος κάθε φορά και στον κάθετο άξονα διακρίνουμε τον αριθμό των δευτερολέπτων που χρειάζεται η λειτουργία ανάγνωσης για να ολοκληρωθεί.



Σχήμα 6.1

Σενάριο 2 – Εγγραφείς

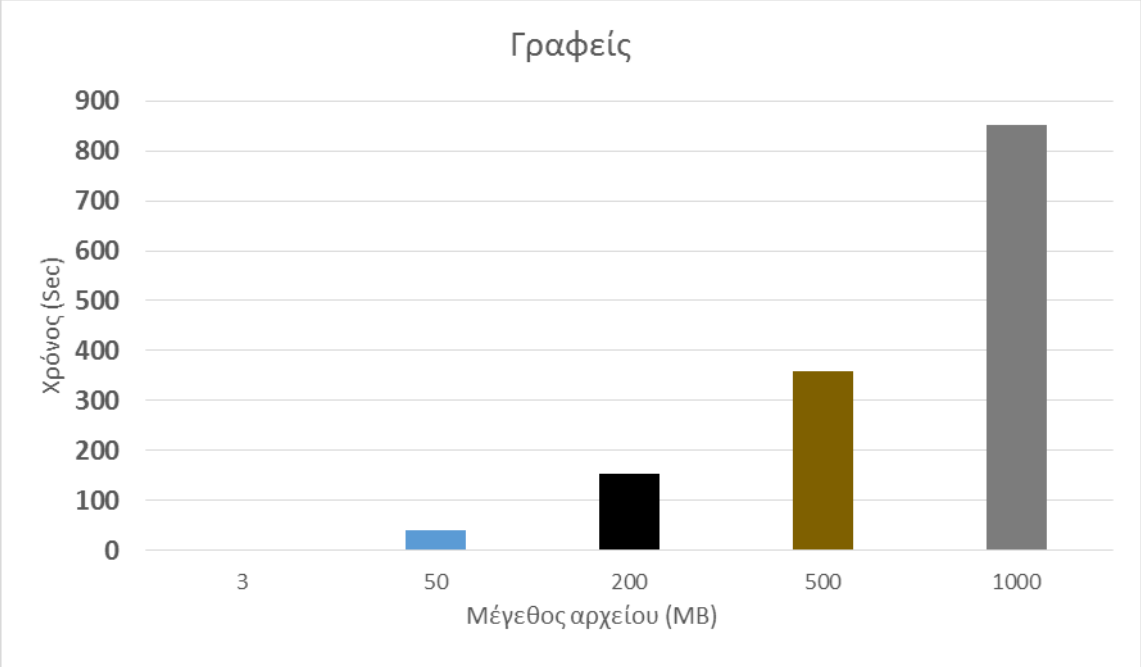
Σε αυτό το σενάριο έχουμε 10 , 20, 30 και 40 εγγραφείς να προσπαθούν να γράψουν το ίδιο αρχείο στο σύστημα. Αξίζει να αναφέρουμε ότι κάποιοι εγγραφείς μπορεί να προλάβουν να ολοκληρώσουν την γραφή τους πριν κάποιοι άλλοι αρχίσουν να γράφουν το αρχείο. Ως αποτέλεσμα κάποιες λειτουργίες γραφής να γίνουν λειτουργίες ανάγνωσης. Επίσης, αναμένουμε ότι ο χρόνος ολοκλήρωσης της λειτουργίας γραφής θα αυξάνεται καθώς αυξάνονται οι πελάτες στο σύστημα. Αυτό είναι απόλυτα λογικό επειδή οι εξυπηρετητές θα χρειάζονται περισσότερο χρόνο για να εξυπηρετούν αρκετούς πελάτες ταυτόχρονα.



Σχήμα 6.2

Σενάριο 3 – Μέγεθος Αρχείου

Στο Σχήμα 6.3 βλέπουμε το σενάριο 2 όπου έχουμε 2 εγγραφείς να προσπαθούν να γράψουν ένα αρχείο. Επίσης σε αυτό το σενάριο έχουμε ως στόχο να ελέγξουμε τη συμπεριφορά του συστήματος για μικρού και μεγάλου μεγέθους αρχεία. Είναι προφανές ότι όσο αυξάνεται το μέγεθος του αρχείου η καθυστέρηση θα αυξάνεται. Ο λόγος είναι ότι όσο αυξάνεται το μέγεθος των αρχείων, αυξάνονται και τα πακέτα των αρχείων που στέλνονται. Στο Σχήμα 6.3 παρουσιάζεται ο μέσος χρόνος που χρειάζεται για να ολοκληρωθεί μια λειτουργία γραφής. Το γενικό συμπέρασμα το οποίο βγάζουμε από την πιο κάτω γραφική παράσταση είναι ότι παρόλο που το μέγεθος των αρχείων αυξάνεται η καθυστέρηση είναι ανάλογη με το μέγεθος του αρχείου. Με άλλα λόγια δεν υπάρχει επιπλέον επιβάρυνση των εξυπηρετητών ασχέτως του μεγέθους. Καταλήγουμε ότι σύμφωνα με την γραφική του Σχήματος 6.3 είναι μια καλή ένδειξη ότι το Κ.Σ.Α έχει καλή επεκτασιμότητα.


Γραφείς
900
800
700
600
500
400
300
200
100
0
Χρόνος (Sec)
3
50
200
500
1000
Μέγεθος αρχείου (MB)

Σχήμα 6.3

Κεφάλαιο 7

Συμπεράσματα

7.1 Γενικά Συμπεράσματα

Σε αυτή τη διπλωματική εργασία υλοποιήθηκε ένα κατανεμημένο σύστημα αρχείων βασισμένο στον αλγόριθμο LDR [5,6]. Η αξιολόγηση του συστήματος έγινε τόσο σε τοπικούς υπολογιστές αλλά και σε απομακρυσμένους υπολογιστές του Πανεπιστημίου Κύπρου. Γενικά η συμπεριφορά του συστήματος ήταν όπως αναμενόταν. Συνεπώς, πολλοί χρήστες μπορούν να αποθηκεύουν οποιοδήποτε τύπο αρχείου ταυτόχρονα και αποδοτικά. Επίσης, έχει εξεταστεί κατά πόσο τηρεί αυστηρή συνέπεια όπως ήταν οι προδιαγραφές του.

Αξίζει να αναφέρουμε κάποιες σχετικές προηγούμενες διπλωματικές εργασίες και να αναφέρουμε σε τι διαφέρει η παρούσα διπλωματική μας εργασία. Μια σχετική διπλωματική εργασία πραγματοποιήθηκε από την φοιτήτρια Σώτια Ιωάννου η οποία έχει υλοποιήσει τον αλγόριθμο LDR στην γλώσσα προγραμματισμού Java. Έχει προδιαγράψει τον αλγόριθμο LDR με τη γλώσσα TIOA (Timed Input/Output Automata) [9] και με τη χρήση του εργαλείου TEMPO [12]. Ο κύριος σκοπός του εργαλείου TEMPO είναι να μετατρέπει το TIOA σε εκτελέσιμο κώδικα στην γλώσσα προγραμματισμού Java. Ο λόγος που δεν έχει γίνει επέκταση του κώδικα της φοιτήτριας Σώτιας Ιωάννου είναι επειδή ο κώδικας του αλγόριθμου που παράγεται από την γλώσσα TIOA σε Java δεν είναι τόσο αποδοτικός. Επομένως, προτιμήσαμε να υλοποιήσουμε εξαρχής τον αλγόριθμο LDR χωρίς την χρήση κάποιου έτοιμου εργαλείου, έτσι ώστε να είναι όσο το δυνατό πιο αποδοτική η υλοποίηση μας. Επίσης η Μαριλένα Δεμετή [3] έχει υλοποιήσει τον αλγόριθμο LDR σε γλώσσα προγραμματισμού C. Ο λόγος ο οποίος ούτε σε αυτή την περίπτωση δεν επεκτείναμε τον κώδικα της φοιτήτριας Μαριλένας Δεμετή ήταν επειδή υπήρχε περιορισμός στην υλοποίηση του μεγέθους του αρχείου του οποίου μπορούσε να σταλεί. Πιο συγκεκριμένα μπορούσε να σταλεί μέχρι μόνο 1MB. Ακόμη, και στις δύο προαναφερθέντες διπλωματικές εργασίες για να τρέξουμε τους αλγόριθμους που

υλοποιήθηκαν χρειαζόταν να το πράξουμε μέσω του τερματικού. Στην παρούσα διπλωματική εργασία μελετήθηκε και υλοποιήθηκε ο τροποποιημένος αλγόριθμος LDR όπως έχει εξηγηθεί στο Κεφάλαιο 3.3. Ακολούθως έγινε ο σχεδιασμός και η ανάπτυξη της διεπαφής του συστήματος. Επίσης έχει υλοποιηθεί ο διαχειριστής αρχείων ο οποίος είναι υπεύθυνος να δημιουργεί και να αποθηκεύει τα αναγνωριστικά των αρχείων, καθώς και να γνωρίζει ποια αρχεία υπάρχουν στο σύστημα και από ποιους χρήστες έχουν δημιουργηθεί.

7.2 Προβλήματα και Παραδοχές

Κατά την εκπόνηση της παρούσας Διπλωματικής μου εργασίας παρουσιάστηκαν κάποια προβλήματα, κυρίως κατά την υλοποίηση. Ένα από τα βασικά προβλήματα ήταν με τον προγραμματισμό υποδοχών. Σε περίπτωση όπου κατέρρεε ένας από τους εξυπηρετητές αρχείων και ευρετηρίων, η υποδοχή έμενε σε λειτουργία. Ως αποτέλεσμα , να μην μπορεί να κάνει ξανά άκουσμα στη συγκεκριμένη υποδοχή. Το πρόβλημα αντιμετωπίστηκε με την συνάρτηση *setsockopt* η οποία μας επιτρέπει το άμεσο άκουσμα σε μια συγκεκριμένη πόρτα ασχέτως αν έχει μείνει ανοιχτή λόγο κάποιου προβλήματος. Επίσης, κατά την χρήση νημάτων υπήρχαν περιπτώσεις όπου έπρεπε να χρησιμοποιηθούν σηματοφόροι έτσι ώστε να μην έρχεται σε ασυνεπή κατάσταση το σύστημα. Τέλος, κατά την ανάπτυξη της διεπαφής δεν υπήρχε κάποιο έτοιμο εργαλείο για το σχεδιασμό έτσι έπρεπε να γίνει εξαρχής η σχεδίαση της καλώντας συναρτήσεις με την βοήθεια της βιβλιοθήκης GTK+.

7.3 Μελλοντική Εργασία

Το κατανεμημένο σύστημα αρχείων έχει υλοποιηθεί σε περιβάλλον Linux. Συνεπώς, κάποιες συναρτήσεις του πελάτη δεν μπορούν να τρέξουν στα υπόλοιπα λειτουργικά συστήματα. Ως μελλοντική εργασία, θα μπορούσε να είναι η τροποποίηση του πελάτη, έτσι ώστε να μπορεί να τρέχει σε όλα τα λειτουργικά συστήματα. Επίσης, ο διαχειριστής αρχείων στην παρούσα διπλωματική είναι κεντρικοποιημένος και ως μελλοντική εργασία θα μπορούσε να είναι κατανεμημένος. Επιπλέον, τα αρχεία θα μπορούσαν να σπάζουν ώστε κάθε πελάτης να δουλεύει σε διαφορετικά κομμάτια ταυτόχρονα, αλλά κάθε κομμάτι να έχει ατομική πρόσβαση.

Βιβλιογραφία

[1] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev, "Sharing Memory Robustly in Message-Passing Systems", Journal of the ACM, vol. 42, no. 1, pp. 124-142, January 1995.

[2] Ajay D. Kshemkalyani, Mukesh Singhal, “Distributed Computing Principles, Algorithms, and Systems”, First Edition, Cambridge University Press, 2008.

[3] Marilena Demeti, Implementation and Experimental Evaluation of the Layered Data Replication Algorithm, Master Thesis, Department of Computer Science University of Cyprus, December 2012.

[4] Hall, B Beej’s Guide to Network Programming Using Internet Sockets, Version, 2005.

[5] R. Fan. Efficient Replication of Large Data Objects. Master Thesis, Computer Science and Engineering, Massachussetts Institute of Technology. February 2003.

[6] R. Fan, N. Lynch. Efficient Replication of Large Data Objects. MIT Computer Science and Artificial Intelligence Laboratory. 2003.

[7] Chryssis Georgiou, "EPL601 Lecture Notes - Chapter 8: Distributed Atomic Objects", Department of Computer Science, University of Cyprus, 2015.

[8] Sotia Ioannou, Specifying and Evaluating a Distributed File System using Tempo, Diploma Project, Department of Computer Science, University of Cyprus, June 2014.

[9] Stephen J. Garland, Dilsum Kaynar, Nancy A. Lynch, Joshua A. Tauber, Mandana Vaziri. TIOA Tutorial. Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, MA, May 22, 2005.

[10] Michael Kerrisk. Linux Programmer's Manual-SOCKET(2). http://man7.org/linux/man-pages/man2/socket.2.html [Accessed 9 February 2016]

[11] Michael Kerrisk. Linux Programmer's Manual - SOCKET(7). http://man7.org/linux/man-pages/man7/socket.7.html [Accessed 9 February 2016]

[12] N. A. Lynch, S. J. Garland, D. Kaynar, L. Michel, A. Shvartsman. The Tempo Language User Guide and Reference Manual. Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology. October 24, 2011.

[13] Nancy Lynch, Alex Shvartsman. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts. In Twenty-Seventh Annual International 103, Symposium on Fault-Tolerant Computing (FTCS'97), pages 272-281, Seattle, Washinghton, USA, June 1997, IEEE.

[14] N. Lynch, "Distributed Algorithms", Morgan Kaufmann Publishers, 1996.

[15] N. Nicolaou, A. Fernandez Anta, and C. Georgiou "Traceable Objects: Consistent Versioning for Concurrent Objects" - arXiv preprint arXiv:1601.07352, 2016

[16] D.Peleg , A.Wool, "Crumbling walls: A class of high availability quorum system". In Proceedings of 14th ACM Symposium on Principles of Distributed Computing (PODC), pages 120-129, 1995.

[17] W.Richard Stevens, "TCP/IP Illustrated, Volume I", Addison Wesley Longman, 1994.

[18] RFC 793, http://www.rfc-editor.org/rfc/rfc793.txt [Accessed 19 January 2016]

[19] Andrew S. Tanenbaum, Maarten Van Steen. Distributed Systems – Principles and Paradigms (2. ed). Pearson Education, 2007.

[20] Ιωάννης Χατζηγιαννάκης, Παύλος Σπυράκης, Χρήστος Ζαρολιάγκης, Σημειώσεις Μαθήματος Κατανεμημένα Συστήματα, Τμήμα Πληροφορικής, Πανεπιστήμιο Πάτρας, 2013.

Παράρτημα Α

Σε αυτό το παράρτημα δίνονται τα προγράμματα του πελάτης, εξυπηρετητή ευρετηρίου, εξυπηρετητή αρχείων και του διαχειριστή αρχείων.

```
/*****************************************************************************/
/*                                                                           */
/*Name: Elias Spanos                                                         */
/*Date: 30/09/2015                                                           */
/*Filename: directory.c                                                      */
/*                                                                           */
/*****************************************************************************/
/*****************************************************************************/
/*                                    LIBRARIES                              */
/*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <strings.h> //For bzero
#include <arpa/inet.h> //To retrieve the ip_add to asci
#include <string.h>
#include "directory.h"
#include <pthread.h>

#define GINT_TO_POINTER(i) ((gpointer) (glong) (i))
#define GPOINTER_TO_INT(p) ((gint)  (glong) (p))

#define MAX_PENDING 5000 // Pending clients in


#define BUFSIZE 256

//Define the array size of metadata table
#define METATABLESIZE 10000

//Format for error for the threads
#define perror2(s,e) fprintf(stderr, "%s:%s\n" , s, strerror(e))

GSList* decode(struct message *msg , char *buf);

/*****************************************************************************/
/*                              GLOBAL VARIABLES                             */
/*****************************************************************************/
struct sockaddr_in serv_addr;

FILE *log_fd = NULL; //File to store the log information of the directory server.
char *logfilename;

/*Lock the strtok*/
pthread_mutex_t locker;

//Lock metadata table
pthread_mutex_t lockermetadata;

//Lock clientCounter
pthread_mutex_t lockercliCoun;


/*****************************************************************************/
/*                              METADATA TABLE                               */
/*****************************************************************************/
//Store all the metadata information by fileID
GPtrArray *metatable=NULL;


//Find the index of metadata for specific fileID
//This function return the position where the data are located
//Otherwise, return -1 if the data does not exist or -2 in case
//where the client does not have the suitable permission to read the data
int findByFileID( struct message *msg , int *freePos)
{

   int index=-1;
   int i=0;
```

```
    //Pointer to the metadata
    struct metadata *point2metadata = NULL;

    for(i=0; i < metatable->len; i++)
    {
        //Retrieve  the data from the specific index
        point2metadata = (struct metadata *) g_ptr_array_index(metatable ,i );

       /* //Check if the filename of the position is what is looking for
        if(strcmp(filename , point2metadata->filename->str)==0)
        {
            index=i;
            break;
        }*/

        //Looking up if the file exist using fileid
        if(point2metadata->file_id == msg->fileID)
        {
            if( (strcmp(point2metadata->permission->str , msg->permission) == 0) || strcmp(msg-
>permission,"L0" ) == 0 )
            {
                index=i;
            }
            else
            {
                index = -2;
            }

            //exit from the for
            break;
        }

    }

    return index;
}


GSList * insertList(GSList *metadata , GSList *curList )
{
    //A temp pointer to go through the curList
    GSList   *iter=NULL;

    //Go through available replicas
    for (iter = curList; iter; iter = iter->next)
    {
        if (g_slist_find(metadata, iter->data ) == NULL)
        {
            metadata = g_slist_prepend(metadata, iter->data );
        }
    }

    return metadata;
}

int IsMaxTag(struct message *tag2)
{
    //Store the error that pthread_mutex_lock failed
    int err;
    int funfreepos;
    int isFound=0;

    //Pointer to the metadata
    struct metadata *point2metadata = NULL;

    //Lock strtok due to is not deadlock free//
    if(err=pthread_mutex_lock(&lockermetadata))
    {
        perror2("Failed to lock()",err);
    }
            //Find the position for the specific filename
            int index=findByFileID(tag2 , &funfreepos);

            //Retrieve  the data from the specific index
            point2metadata = (struct metadata *) g_ptr_array_index(metatable , index );

            if ((point2metadata->tag.num < tag2->tag.num) \
                                        || (point2metadata->tag.num == tag2->tag.num &&
point2metadata->tag.id < tag2->tag.id ))
            {
```

```
                point2metadata->tag.num=tag2->tag.num;
                point2metadata->tag.id=tag2->tag.id;

                //Remove old replica set because now is unnecessary due to
                //the tag is old one
                g_slist_free ( point2metadata->replicaSet );
                point2metadata->replicaSet=NULL;

                //Point to new replica set with the latest tag
                point2metadata->replicaSet=tag2->replicaSet;

               isFound=1;
            }

            else if(point2metadata->tag.num == tag2->tag.num && point2metadata->tag.num == tag2-
>tag.id )
            {
                point2metadata->replicaSet = insertList( point2metadata->replicaSet , tag2-
>replicaSet);
            }

    //Unloack Mutex/
    if(err=pthread_mutex_unlock(&lockermetadata))
    {
        perror2("Failed to lock()",err);
    }

    return isFound;
}

/*****************************************************************************/
/*                        Initialization of variables & Sockets              */
/*****************************************************************************/
void inisializations(int portIn)
{
    //Pointer for g_slist to store the replica that hold
    //the data & repli is which replica hold the data
    gpointer v;
    int      repli;

    //Initialize metadata array
    metatable = g_ptr_array_sized_new(1000);

    //Initialization of mutex for strtok in decode function//
    pthread_mutex_init(&locker,NULL);

    //Initialization of mutex for lock the metadata table
    pthread_mutex_init(&lockermetadata,NULL);

}

/*****************************************************************************/
/*              FUNCTION TO CONFIGURE A CONNECTION FOR THE SERVER             */
/*****************************************************************************/
void *bind_thread(void *port)
{

    int PORT=(intptr_t)port;

    //Socket descriptor for the directory
    int sock;
    //Store server socket.
    int newsock;
    //The size of the details for the accepted client
    int clientlen;
    //Variable to store the thread id
    pthread_t tid;
    //Store the error of pthread_create if exist
    int err;
    //for setsockopt to be possible to reuse the port
    int allow=1;

    //Declare&Initialize sockaddr_in pointer for accept function
    struct sockaddr *clientPtr;
    struct sockaddr_in client_addr;

    /* Initialize socket structure */
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
    serv_addr.sin_port=htons(PORT);
```

```
    //Pointer to Struct for the structure of the TCP/IP socket.
    struct sockaddr *servPtr;
    //Point to struct serv_addr.
    servPtr=(struct sockaddr *) &serv_addr;

    //Create a socket and check if is created correct.
    if((sock=socket(AF_INET,SOCK_STREAM,0)) < 0)
    {
            perror("Socket() failed");
            pthread_exit((void *) 0);
    }

    //Allow to reuse the port
    if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &allow, sizeof(int)) == -1)
    {
        perror("\nsetsockopt\n");
        close(sock);
        pthread_exit((void *) 0);
    }

    //Accosiate address with the sock socket//
    if(bind(sock, servPtr, sizeof(serv_addr)) < 0)
    {
        perror("Bind() failed"); exit(1);
    }

    //Listen for connections//
    //MAX_PENDING:the number of client that can wait in the queue.//
    if(listen(sock ,MAX_PENDING) < 0)
    { /* 15 max. requests in queue*/
        perror("Listen() failed"); exit(1);
    }

    printf("\n*************Start to listen to port:%d******************\n" , PORT);

    //Accept connection from clients forever.
    while(1)
    {
        clientPtr=(struct sockaddr *) &client_addr;
        clientlen= sizeof(client_addr);

        if((newsock=accept(sock , clientPtr  , &clientlen )) < 0)
        {
            perror("accept() failed"); exit(1);
        }

        if(err=pthread_create(&tid , NULL , &accept_thread ,  (void *)(intptr_t) newsock))
        {
            perror2("pthread_create for accept_thread" , err);
            exit(1);
        }

    }//While(1)

}//Function connect

/*****************************************************************************/
/*                            accept_thread                                  */
/*****************************************************************************/
void *accept_thread(void *accept_sock)
{
    //socket descriptor for each client
    int acpt_sock;
    //Buffer to send&receive data//
    char buf[512];
    //Received bytes
    int bytes;

    /*Declare the structure of the sending message in order
     * the client to communicate with server and vice versa*/
    struct message *msg;

    //Allocation memory
    msg = (struct message *) malloc(sizeof(struct message));

    //Retrieve the socket that passed from pthread_create
    acpt_sock = (intptr_t) accept_sock ;

 while(1)
```

```
{
    /*If connection is established then start communicating */
    /*Initialize buffer with zeros */
    bzero(buf, sizeof(buf));
    /*Waiting...to receive data from the client*/
    if (recv(acpt_sock, buf, sizeof(buf), 0) < 0)
    {
        perror("Received() failed!");
        close(acpt_sock);
        pthread_exit((void *) 0);
    }

    //Check if direc received a message
    if(strlen(buf) != 0)
    {
        /*Show the message that received.*/
        printf("---------------------------------\n");
        printf("accept_thread received: %s\n", buf);
    }
    else if (strlen(buf) == 0)
    {
        //printf("Unable to received messsage!\n");
        close(acpt_sock);
        pthread_exit((void *) 0);
    }

    //Decode the message that receive from the client
    // in order to identify the type of the message//
    msg->replicaSet =  decode(msg, buf);

    //Directory ALGORITHM//

    //Check if the message if of type RREAD
    if ((strcmp("RREAD", msg->type) == 0))
    {
        int freepos = -1 , err=0;

        bzero(buf, sizeof(buf));

        //Look up the position of the specific fileid
        //in the metadata.

        //Lock strtok due to is not deadlock free
        if(err=pthread_mutex_lock(&lockermetadata))
        {
            perror2("Failed to lock()",err);
        }
           int index = findByFileID( msg ,  &freepos );

        //Unloack Mutex
        if (err = pthread_mutex_unlock(&lockermetadata))
        {
            perror2("Failed to lock()", err);
        }
        //For any reason if the file which the clint enquired does not
        //exist in the metadata then it is not possible for the client
        //to receive an answer.
        if (index == -1 )
        {
            printf("Unable to find the file in metadata table\n");

        }
        //If the user does not have the appropriates then the directory
        //return <<ACCESSDENIED>> message to the client.
        else  if(index == -2)
        {
            //In case where the client does not have the permission to read the file
            sprintf(buf, "ACCESSDENIED,%ld,%ld", msg->msg_id, msg->fileID );

            //Send response to the client
            if (bytes = send(acpt_sock, buf, sizeof(buf), MSG_DONTWAIT) < 0)
            {
                perror("Send() failed");
                pthread_exit((void *) 0);
            }

        }
        //<RREAD-OK,tag.num , tag.id ,value,  MSG_ID >//
        //In case when the file exist
        else
```

```
        {
            //Lock strtok due to is not deadlock free
            if(err=pthread_mutex_lock(&lockermetadata))
            {
                perror2("Failed to lock()",err);
            }

                //Pointer to the metadata
                struct metadata *point2metadata = NULL;

                //Retrieve  the data from the specific index
                point2metadata = (struct metadata *) g_ptr_array_index(metatable , index );

                int i = 0, len, repliValue;
                len = g_slist_length( point2metadata->replicaSet);
                //Read all replica that hold the data
                if (len != 0)
                {
                    //To point on replica set & go through all the list
                    GSList *iterator = NULL;
                    //To serialize replicaSet to string
                    char strSet[256];
                    char str[40];

                    //Initialize str
                    bzero(strSet, sizeof(strSet));

                    for (iterator = point2metadata->replicaSet; iterator; iterator = iterator->next)
                    {
                        bzero(str, sizeof(str));
                        sprintf(str, ",%d", GPOINTER_TO_INT(iterator->data));
                        strcat(strSet, str);
                    }
                    sprintf(buf, "RREAD-OK,%ld,%ld,%d%s,%ld,%ld", point2metadata->tag.num,
point2metadata->tag.id, len,
                                strSet,
                                msg->msg_id, msg->fileID);
                }
                else
                {

                    printf("Unable to find the file:%s", msg->filename);
                    //sprintf(buf, "RREAD-OK,%d,%d,0,%d,%d", metatable[index].tag.num,
metatable[index].tag.id, msg->msg_id,
                    //      msg->fileID);
                }
            //Unloack Mutex
            if (err = pthread_mutex_unlock(&lockermetadata))
            {
                perror2("Failed to lock()", err);
            }
            // sprintf(buf,"RREAD-
OK,%d,%d,%d,%d,%d",metatable[index].tag.num,metatable[index].tag.id,metatable[index].replicaSet,msg.ms
g_id , msg.fileID );


            //Send response to the client
            if (bytes = send(acpt_sock, buf, sizeof(buf), MSG_DONTWAIT) < 0)
            {
                perror("Send() failed");
                pthread_exit((void *) 0);
            }
        }
    }
    //Check if the message if of type RREAD
    else if ((strcmp("RWRITE", msg->type) == 0))
    {
        //Check if the tag that received if is greater
        if (IsMaxTag(msg) == 1)
        {
            printf("Found max Tag\n");
        }

        bzero(buf, sizeof(buf));
        sprintf(buf, "RWRITE-OK,%ld\n", msg->msg_id);
        if (bytes = send(acpt_sock, buf, sizeof(buf), 0) < 0)
        {
            perror("Send() failed");
            pthread_exit((void *) 0);
        }
```

```
    }//RREAD RESPONSE

    else if (strcmp("WREAD", msg->type) == 0)
    {
                int freepos = -1,err=0;

                /*Lock strtok due to is not deadlock free*/
                if(err=pthread_mutex_lock(&lockermetadata))
                {
                    perror2("Failed to lock()",err);
                }

                //Find the index of the filename
                int index = findByFileID(msg , &freepos);

                if(index == -2)
                {
                    //In case where the client does not have the permission to read the file
                    sprintf(buf, "ACCESSDENIED,%ld,%ld", msg->msg_id, msg->fileID );

                    //Send response to the client
                    if (bytes = send(acpt_sock, buf, sizeof(buf), MSG_DONTWAIT) < 0)
                    {
                        perror("Send() failed");
                        pthread_exit((void *) 0);
                    }

                }

               else  if(index != -1)
                {
                    //Pointer to the metadata
                    struct metadata *point2metadata = NULL;

                    //Retrieve  the data from the specific index
                    point2metadata = (struct metadata *) g_ptr_array_index(metatable, index);

                    int i = 0, len, repliValue;
                    len = g_slist_length(point2metadata->replicaSet);
                    //Read all replica that hold the data
                    if (len != 0)
                    {
                        //To point on replica set & go through all the list
                        GSList *iterator = NULL;
                        //To serialize replicaSet to string
                        char strSet[256];
                        char str[40];

                        //Initialize str
                        bzero(strSet, sizeof(strSet));

                        for (iterator = point2metadata->replicaSet; iterator; iterator = iterator-
>next) {
                            bzero(str, sizeof(str));
                            sprintf(str, ",%d", GPOINTER_TO_INT(iterator->data));
                            strcat(strSet, str);
                        }
                        sprintf(buf, "WREAD-OK,%ld,%ld,%d%s,%ld,%ld", point2metadata->tag.num,
point2metadata->tag.id, len,
                                    strSet,
                                    msg->msg_id, msg->fileID);
                    }
                    //In case when the replica set is NULL
                    else
                    {
                        sprintf(buf, "WREAD-OK,%ld,%ld,0,%ld,%ld", point2metadata->tag.num,
point2metadata->tag.id,
                                 msg->msg_id, msg->fileID);

                    }
                }

                //If the filename does not exist in metadata table store it
                else
                {
                    //Allocate a new entry for the metadata table
                    struct metadata *entry = (struct metadata*) malloc(sizeof(struct metadata));

                    //Initialize new entry
```

```
                    entry->file_id = msg->fileID;
                    entry->tag.num = 1;
                    entry->tag.id  = 0;
                    entry->replicaSet=NULL;
                    entry->filename=g_string_new(NULL);
                    entry->permission=g_string_new(NULL);

                    //Insert the string the metadata table
                    g_string_assign( entry->filename ,  g_strdup( msg->filename ) );

                    //Insert permission in the metadata
                    g_string_assign(entry->permission , g_strdup( msg->permission ));

                    //insert new entry in metadata table
                    g_ptr_array_add(metatable, (gpointer) entry );

                    //Find the index of the new entry
                    index = findByFileID(msg , &freepos);

                    //Store permission for the file
                    g_string_assign( entry->permission ,  g_strdup( msg->permission ) );

                    //Pointer to the metadata
                    struct metadata *point2metadata = NULL;

                    //Retrieve  the data from the specific index
                    point2metadata = (struct metadata *) g_ptr_array_index(metatable , index );

                    bzero(buf, sizeof(buf));
                    if(point2metadata != NULL)
                    {
                        sprintf(buf, "WREAD-OK,%ld,%ld,0,%ld,%ld", point2metadata->tag.num,
point2metadata->tag.id, msg->msg_id,
                                point2metadata->file_id);
                    }

                }

        //Unloack Mutex//
        if(err=pthread_mutex_unlock(&lockermetadata))
        {
            perror2("Failed to lock()",err);
        }


        if (bytes = send(acpt_sock, buf, sizeof(buf), 0) < 0)
        {
            perror("Send() failed");
            pthread_exit((void *) 0);
        }

    }
    else if (strcmp("WWRITE", msg->type) == 0)
    {

        if (IsMaxTag(msg) == 1)
        {
            printf("\nFound new max tag");
        }

        bzero(buf, sizeof(buf));
        sprintf(buf, "WWRITE-OK,%ld\n", msg->msg_id);
        if (bytes = send(acpt_sock, buf, sizeof(buf), 0) < 0)
        {
            perror("Send() failed");
            pthread_exit((void *) 0);
        }
    }

    //print the message that directory sent
     printf("\nSend:%s\n", buf);

 }//while(1)
    close(acpt_sock);
    pthread_exit((void *) 0);

 }

/*****************************************************************************/
/*                            DECODE MSG                                     */
```

```
/*****************************************************************************/
GSList* decode(struct message *msg , char *buf)
{
    //Store the thread error if exist
    int err;

    //Lock strtok due to is not deadlock free//
    if(err=pthread_mutex_lock(&locker))
    {
        perror2("Failed to lock()",err);
    }

        //To store a point to the set of replicas
        GSList *replicaSet=NULL;

        //Use comma delimiter to find the type of
         // send message and retrieve the apropriate
         // fields//
        msg->type=strdup(strtok(buf, ","));

        //Check if the type of the message is RWRITE//
        if( (strcmp(msg->type , "RWRITE" )== 0) || (strcmp(msg->type , "WWRITE" )== 0) )
            {
                int i=0 , len = 0 , repliValue;
                gpointer val;
                len = atoi(strtok(NULL, ","));
                //Read all replica that hold the data
                for(i=0; i<len; i++)
                {
                    repliValue = atoi(strtok(NULL, ","));
                    val=GINT_TO_POINTER(repliValue);
                    msg->replicaSet=g_slist_prepend(msg->replicaSet , val );
                }
                msg->tag.num=atoi(strtok(NULL,","));
                msg->tag.id=atoi(strtok(NULL,","));
                msg->msg_id=atoi(strtok(NULL,","));
                msg->filename=strdup(strtok(NULL,","));
                replicaSet = msg->replicaSet;
            }
        //Check if the type of the message is RREAD//
        else if(strcmp(msg->type , "RREAD" )== 0)
        {
            msg->msg_id=atoi(strtok(NULL,","));
            msg->filename=strdup(strtok(NULL,","));
            msg->fileID = atol(strtok(NULL,","));
            msg->permission = strdup(strtok(NULL,","));
        }
        else if(strcmp(msg->type , "WREAD" )== 0)
        {
            msg->msg_id=atoi(strtok(NULL,","));
            msg->fileID = atol(strtok(NULL,","));
            msg->filename=strdup(strtok(NULL,","));
            msg->permission = strdup(strtok(NULL,","));
        }

    //Unloack Mutex//
    if(err=pthread_mutex_unlock(&locker))
    {
        perror2("Failed to lock()",err);
    }

    return replicaSet;

}//Function decode message

/*****************************************************************************/
/*                               WRITE LOG                                   */
/*****************************************************************************/
int writeLog(char *info)
{
    log_fd=fopen(logfilename , "a+");
    fprintf(log_fd , info , sizeof(info));
    fclose(log_fd);
}//Function writeLog

/*****************************************************************************/
/*                            GET CURRENT DATE                               */
/*****************************************************************************/
/*
char* set_date()
```

```
{
    char namelog[100];
    time_t t = time(NULL);
    struct tm tm=*localtime(&t);
    sprintf(namelog,"directory_%d_%d_%d.log",(tm.tm_mday),(tm.tm_mon+1),(tm.tm_year+1900));
    return namelog;
}//Fucntion setDate
*/

void signal_handler()
{
    printf("Signal Handled here\n");

    int i=0;

    //Pointer to the metadata
    struct metadata *point2metadata = NULL;

    printf("[ Filename , TagNo , ClientID , FILEID , Permission ] \n");
    for(i=0; i < metatable->len; i++)
    {
        //Retrieve  the data from the specific index
        point2metadata = (struct metadata *) g_ptr_array_index(metatable , i );

        //deallocations
        printf("[ %s       ,   %ld  ,    %ld    ,   %ld   ,     %s    ] \n" , point2metadata-
>filename->str , point2metadata->tag.num ,point2metadata->tag.id , point2metadata->file_id,
point2metadata->permission->str);

        //deallocate list
        g_slist_free (point2metadata->replicaSet);

        //deallocate filename
        g_string_free (point2metadata->filename, TRUE);

        //deallocate permission string
        g_string_free (point2metadata->permission, TRUE);

        //deallocate struct
        free(point2metadata);
    }

    //exit the server
    exit(0);
}

/*****************************************************************************/
/*                                MAIN                                       */
/*****************************************************************************/
int main(int argc , char *argv[])
{

/*****************************************/
/*          LOCAL DECLARATION            */
/*****************************************/

   int port=-1; //Store the port that server listen

/*****************************************/
/*          MEMORY ALLOCATIONS           */
/*****************************************/

    //Check if the input parameters are correct. //
    if(argc!=3)
    {
        printf("\nUsage: ./executable -p [port]\n");
        exit(-1);
    }

    printf("\nPID: %d\n", getpid());

    //Store pthread ID
    pthread_t tid;
    //Store threads errors
    int err;

    /*store the port from the inputs*/
    port = atoi(argv[2]);

    //Inisialization
```

```
    inisializations(port);


    // SIGINT is signal name create  when Ctrl+c will pressed
    signal(SIGINT,signal_handler);
    //Handle segmentation corrupt
    signal(SIGSEGV, signal_handler);

    printf("------------------------------------------------------------");
    printf("\nDirectory: starting on port: %d...\n" ,  port);
    printf("------------------------------------------------------------\n");


    //Create a thread for the bind.
    if(err=pthread_create(&tid , NULL ,(void *) &bind_thread , (void *) (intptr_t)port))
    {
        perror2("pthread_create" , err);
        exit(1);
    }

    pthread_exit(NULL);

    return 0;
}//Main Function
```

Directory.h

```
/*****************************************************************************/
/*                                                                           */
/*Name: Elias Spanos                                                        */
/*Date: 09/10/2015                                                           */
/*Filename:directory.h                                                       */
/*                                                                           */
/*****************************************************************************/
#ifndef DISTRIBUTEDALGORITHM_DIRECTORY_H
#define DISTRIBUTEDALGORITHM_DIRECTORY_H

#include <glib.h>

void *bind_thread(void *port);
void *accept_thread(void *accept_sock);

struct tagID
{
    long num;
    long id;
};

struct message
{
    char *type;
    long  msg_id;
    GSList* replicaSet;
    struct tagID  tag;
    long fileID;
    char *filename;
    char *permission;
};

struct metadata
{
    long  file_id;
    GString* permission;
    struct tagID  tag;
    GSList* replicaSet;
    GString *filename;
};

struct sockaddr_in serv_addr;

/*****************************************************************************/
/*                            PROTOTYPES                                     */
/*****************************************************************************/
int findByFileID( struct message *msg , int *freePos);

GSList * insertList(GSList *metadata , GSList *curList );

int IsMaxTag(struct message *tag2);
```

```
void inisializations(int portIn);

void *bind_thread(void *port);

void *accept_thread(void *accept_sock);

GSList* decode(struct message *msg , char *buf);

int writeLog(char *info);

void signal_handler();
#endif //DISTRIBUTEDALGORITHM_DIRECTORY_H
```

Replica.c

```
/*****************************************************************************/
/*                                                                           */
/*Name: Elias Spanos                                                        */
/*Date: 09/10/2015                                                           */
/*Filename:Replica.c                                                         */
/*                                                                           */
/*****************************************************************************/
/*****************************************************************************/
/*                                    LIBRARIES                              */
/*****************************************************************************/
//---------------------------------------------------------------------------//
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h> //To retrieve the ip_add to asci
#include <sys/stat.h>
#include <errno.h>
#include <sys/sendfile.h>
#include <fcntl.h>
#include <pthread.h>



#include "replica.h"

#define BUFSIZE 4098
#define MAXCLIENT 10000

#define SUCCESS  1
#define FAILURE -1

//Format for error for the threads
#define perror2(s,e) fprintf(stderr, "%s:%s\n" , s, strerror(e))

struct sockaddr_in *direc_sockaddr;   //Store all the ips and ports for directories
struct sockaddr_in *replica_sockaddr; //Store all the ips and ports for Replicas
struct serinfo *direcNodes;           //Store IP and port for each directory server
struct serinfo *replicaNodes;         //Store IP and port for each replica server
int    *direcSocks;                   //Store all the sockets for directories
int    *replicaSocks;                 //Store all the sockets for replicas
int    MAX_DIRECTORIES=0;             //Store the number of directories that exist in the system
int    MAX_REPLICAS=0;                //Store the number of replicas that exist in the system
int    NODEID;                        //Store which ID is the replica based on port



/*****************************************************************************/
/*                              DATA STRUCTURES                              */
/*****************************************************************************/
//---------------------------------------------------------------------------//

//Metadata table that holds all tag for the data
GHashTable * metadatatable;

//Lock metadata table
pthread_mutex_t lockermetadata;

/*Lock the strtok*/
```

```
pthread_mutex_t locker;


//Store all the metadata information
GPtrArray *metatable=NULL;


/*****************************************************************************/
/*                              MD5 CHECKSUM                                 */
/*****************************************************************************/
char *checksum_get(char *filename)
{

    int MAX_SIZE=900000;

    GChecksum   *cs;
    guchar       data[MAX_SIZE];
    gsize        size = 0;
    FILE        *input;

    const char *sum;

    cs = g_checksum_new( G_CHECKSUM_MD5 );
    input = fopen( filename, "rb" );
    do
    {
        size = fread( (void *)data, sizeof( guchar ), MAX_SIZE, input );
        g_checksum_update( cs, data, size );
    }
    while( size == MAX_SIZE );
    fclose( input );

    sum = g_checksum_get_string( cs );
    // g_checksum_free( cs );

    return (char*)sum;

}

/*****************************************************************************/
/*                            Establish connections                          */
/*****************************************************************************/
void conn2direc()
{
    int i=0;
    char buf[50];
    bzero(buf,sizeof(buf));
    int bytes;

    for(i=0; i < MAX_DIRECTORIES; i++)
    {
        /*Establish a connection withe the IP address from the struct server*/
        if (connect(direcSocks[i] , (struct sockaddr *) &direc_sockaddr[i], sizeof(direc_sockaddr[i]))
< 0)
        {

            printf("Unable to Connect to : %s ,  PORT: %d\n", inet_ntoa(direc_sockaddr[i].sin_addr),
ntohs(direc_sockaddr[i].sin_port) );
        }
        else
        {
            printf("Connection established to IP: %s ,  PORT: %d\n",
inet_ntoa(direc_sockaddr[i].sin_addr),ntohs(direc_sockaddr[i].sin_port) );
        }

    }//For statment

}//Function establish connections

int connect2Replicas()
{
    int i=0;

    for(i=0; i < MAX_REPLICAS; i++)
    {

        if( NODEID != replicaNodes[i].port  )
        {
            /*Establish a connection withe the IP address from the struct server*/
```

```
            if (connect(replicaSocks[i] , (struct sockaddr *) &replica_sockaddr[i],
sizeof(replica_sockaddr[i])) < 0)
            {
                printf("Unable to Connect to : %s ,  PORT: %d\n",
inet_ntoa(replica_sockaddr[i].sin_addr), ntohs(replica_sockaddr[i].sin_port) );
            }
            else
            {
                printf("Connection established to IP: %s ,  PORT: %d\n",
inet_ntoa(replica_sockaddr[i].sin_addr),ntohs(replica_sockaddr[i].sin_port));
            }
        }
    }
}

struct replicaHeader* decode(char *buf )
{
    //Store the thread error if exist
    int err;

    //Lock strtok due to is not deadlock free//
    if(err=pthread_mutex_lock(&locker))
    {
        perror2("Failed to lock()",err);
    }

        struct replicaHeader *msg;
        msg  = (struct replicaHeader *)malloc(sizeof(struct replicaHeader));
        msg->tag = (TAG *)malloc(sizeof(TAG));

        /*Use comma delimiter to find the type of
             * send message and retrieve the apropriate
             * fields*/
        msg->type=strdup(strtok(buf, ","));
        if( (strcmp(msg->type , "WRITE" )== 0))
        {
            msg->tag->num = atoi(strtok(NULL,","));
            msg->tag->clientID = atoi(strtok(NULL,","));
            msg->fileid = atol(strtok(NULL,","));
            msg->tag->isSecure = 0 ;
            msg->msgID = atoi(strtok(NULL,","));
            msg->fileSize = atoi(strtok(NULL,","));
            msg->checksum = strdup(strtok(NULL,","));
            msg->filename = strdup(strtok(NULL,","));
        }
        else if( (strcmp(msg->type , "READ" )== 0))
        {
            msg->tag->num = atoi(strtok(NULL,","));
            msg->tag->clientID = atoi(strtok(NULL,","));
            msg->msgID = atoi(strtok(NULL,","));
            msg->fileid = atol(strtok(NULL,","));
            msg->filename = strdup(strtok(NULL,","));
        }
        else if((strcmp(msg->type , "SECURE" )== 0))
        {
            msg->tag->num = atoi(strtok(NULL,","));
            msg->tag->clientID = atoi(strtok(NULL,","));
            msg->tag->isSecure=1;
            msg->msgID = atoi(strtok(NULL,","));
            msg->fileid = atol(strtok(NULL,","));
            msg->filename = strdup(strtok(NULL,","));
        }
        //Handle the case where a file does not have
        //file extension
        if(strchr(msg->filename, '.') != NULL)
        {
            char *temp;
            temp= strtok(msg->filename,".");
            msg->filetype=strdup(strtok(NULL," "));
        }
        else
        {
            msg->filetype ="";
        }

    //Unloack Mutex//
    if(err=pthread_mutex_unlock(&locker))
    {
        perror2("Failed to lock()",err);
    }
```

```
    return msg;
}

int send2ftp(int newsock , struct replicaHeader *msg )
{

    int         fd;
    off_t       offset = 0;
    int         remain_data;
    int         sent_bytes;
    /*to retrieve information for the file*/
    struct      stat file_stat;
    int         len;
    int         file_size;
    char        sendheader[512];
    char        filename[255];
    unsigned    int length;

    bzero(sendheader,sizeof(sendheader));

    //
    length=sprintf(filename,"%s_%ld_%d_%d_%d.%s",msg->filename,msg->fileid,msg->tag->num , msg->tag-
>clientID,msg->tag->isSecure,msg->filetype);
    //filename[length-1]='\0';

    printf("FileNAmeIn:%s" , filename);
    fd = open(filename,  O_RDONLY);
    if (fd < 0 )
    {
        fprintf(stderr, "Error opening file --> %s", strerror(errno));
        close(fd);
        return FAILURE;
    }

    //Get file stats //
    if (fstat(fd, &file_stat) < 0)
    {
        printf("Error fstat");
        close(fd);
    }

    // Sending file size //
    file_size=file_stat.st_size;
    printf("\nFile Size: \n %d bytes\n",file_size);


    //Retrieve checksum for the file
    char *filechecksum = checksum_get(filename);

    //
    sprintf(sendheader,"READ-OK,%d,%d,%d,%d,%s", msg->tag->num , msg->tag->clientID ,msg->msgID ,
file_size , filechecksum );

    /* If connection is established then start communicating */
    len = send(newsock, sendheader, sizeof(sendheader), 0);
    if (len < 0)
    {
        perror("send");
        return FAILURE;
    }


    printf("Server sent %d bytes for the size\n" , len);

    remain_data = file_stat.st_size;
    /* Sending file data */
    while (((sent_bytes = sendfile(newsock, fd, &offset, BUFSIZE)) > 0) && (remain_data > 0))
    {
        remain_data -= sent_bytes;
        fprintf(stdout, "Server sent %d bytes from file's data, offset is now : %d and remaining data
= %d\n", sent_bytes, offset, remain_data);
    }

    printf("Finish sending\n");
    //close(newsock);
    close(fd);

    //deallocate
```

```
    free(filechecksum);

    //check if it received the whole file
    //otherwise return an error
    if(remain_data > 0)
    {
        return FAILURE;
    }

    return SUCCESS;

}

int ftp_recv(int sock, struct replicaHeader *msg )
{

    FILE     *received_file;
    int      remain_data;
    int      bytes;
    int      file_size;

    //Store the actual filename
    char     filename[256];

    char     buffer[BUFSIZE];

    //Store the size of the filename
    unsigned int length;

    printf("Received: %d\n" , msg->fileSize);

    bzero(filename,sizeof(filename));

    //Handle the case where file does not have type extension
    if(strcmp(msg->filetype, "") == 0)
    {
        length= sprintf(filename , "%s_%ld_%d_%d_0" , msg->filename , msg->fileid , msg->tag->num ,
msg->tag->clientID);
    }
    else
    {
        length= sprintf(filename , "%s_%ld_%d_%d_0.%s" , msg->filename , msg->fileid , msg->tag->num ,
msg->tag->clientID , msg->filetype);
    }


    //filename[length-1]='\0';

    received_file = fopen(filename, "w");
    if (received_file == NULL)
    {
        fprintf(stderr, "Failed to open file foo --> %s\n", strerror(errno));
        return FAILURE;
    }
    remain_data = msg->fileSize;


    bzero(buffer, sizeof(buffer));
    while (((bytes = recv(sock, buffer, sizeof(buffer), 0)) > 0) && (remain_data > 0))
    {
        fwrite(buffer, sizeof(char), bytes, received_file);
        remain_data -= bytes;
       // fprintf(stdout, "Receive %d bytes and we hope :- %d bytes\n", bytes, remain_data);

       // printf("Received: %d , remain : %d\n",bytes , remain_data);
        if(remain_data <=0 )
        {
            break;
        }

    }//While

    fclose(received_file);

    //Retrieve the checksum from the file that the
    //client receive in order to check if receive
    //the whole file
   // char *filechecksum = checksum_get(filename);

    //check if it received the whole file
```

```
    //otherwise return an error
    if(remain_data > 0)
    {
        return FAILURE;
    }
   /* else if(strcmp(filechecksum , msg->checksum) != 0)
    {
        printf("Invalid checksum\n");

        //Remove the file
        remove(filename);
        return FAILURE;
    }*/

    printf("Received file: %s successful\n" , filename);

    return SUCCESS;
}

GHashTable *  insertMetadata(GHashTable * funmetatable ,  TAG* filetag , char *filename )
{

    //printf("Enter Function insertMetadata\n");

    //Create a new tag to be insert in list
    TAG* newfiletag = g_new(TAG,1);
    newfiletag->num = filetag->num ;
    newfiletag->clientID = filetag->clientID ;
    newfiletag->isSecure = filetag->isSecure ;

    int err=0;

    //Check if the filename exist in metadata table
    //if(g_hash_table_contains(funmetatable , filename) == 1 )
    //{

    //Lock strtok due to is not deadlock free
    if(err=pthread_mutex_lock(&lockermetadata))
    {
        perror2("Failed to lock()",err);
    }

            //Pointer to a list to retrieve all the tags for the
            //specific filename
            GSList *pointlist=NULL;

            //Store all the metadata information by fileID
            GPtrArray *pointArray=NULL;


            //Retrieve all the tags which are associated withe the specific filename
             pointArray =(GPtrArray*) g_hash_table_lookup(funmetatable,  filename );

            if(pointArray != NULL)
            {


                g_ptr_array_add(pointArray, (gpointer) newfiletag);

                //Insert first the tag in list and after in hashtable
             //   pointlist=g_slist_prepend(pointlist , newfiletag );

                //Insert updated list in hashtable to the associated key(filename)
                g_hash_table_insert(funmetatable , filename, pointArray);

            }
                //In case where the filename does not exist in hashtable you have to insert
                //the key and also create a list that will holds the tags
            else
            {
                //Create of the list that will hold the tags for a specific filename
                GPtrArray *tagset=NULL;

                //Initialize metadata array
                tagset = g_ptr_array_sized_new(100);

                //Insert new tag in the list
                g_ptr_array_add(tagset, (gpointer) newfiletag);
                //tagset=g_slist_prepend(tagset , newfiletag );
```

```
                //Insert new list in hashtable to the associated key(filename)
                g_hash_table_insert(funmetatable , g_strdup( filename ), tagset);

            }


    //Unloack Mutex//
    if(err=pthread_mutex_unlock(&lockermetadata))
    {
        perror2("Failed to lock()",err);
    }

   // printf("Exit  Function insertMetadata\n");

    return funmetatable;

}

TAG* findMaxTag(GHashTable * funmetatable , TAG* filetag , char *filename  , int *hasError)
{

     printf("---------------------------\n");
     printf("ENTER FINDMAXTAG FUNCTION \n");


    //Initialize variable hasError
    //If error exist the variable will become 1
    *hasError=0;

    //Pointers for tags
    TAG* secureTag = NULL;
    TAG* existTag  = NULL;


    //Check if the filename exist in metadata table
    //otherwise you must fill the hasError variable
    //if(g_hash_table_contains(funmetatable , filename) == 1 )
    //{

    //Pointer to a list to retrieve all the tags for the
    //specific filename
    //GSList *pointlist=NULL;


    //Store all the metadata information by fileID
    GPtrArray *pointArray=NULL;


    //Retrieve all the tags which are associated with the specific filename
    pointArray =(GPtrArray*) g_hash_table_lookup(funmetatable,filename);


    if(pointArray != NULL)
    {
            //To point at the beginning of the list and to go through all the list
            //to detect if the tag that looking for exist. Otherwise, must return
            //the max secure tag
            GSList* iter=NULL;

            //A temp tag to go through the list tags
            TAG* curTag  = NULL;

       /*     for(iter=pointlist; iter; iter = iter->next)
            {
                //Point to each tag in list
                curTag = (TAG *) iter->data;

                //Check if the tag that looking for exist in the list
                if(curTag->num == filetag->num && curTag->clientID == filetag->clientID )
                {
                    //Tag found
                    existTag=curTag;
                    break;
                }
                    //Looking for secure tag
                else if(curTag->isSecure == 1)
                {
                    //Secure tag found
                    secureTag = curTag;
                }
```

```
            }*/

            int i=0;
            for(i=0; i < pointArray->len; i++)
            {
                //Retrieve  the data from the specific index
                curTag = (TAG *) g_ptr_array_index(pointArray ,i );

                //Check if the tag that looking for exist in the list
                if(curTag->num == filetag->num && curTag->clientID == filetag->clientID )
                {
                    //Tag found
                    existTag=curTag;
                    break;
                }
                    //Looking for secure tag
                else if(curTag->isSecure == 1)
                {
                    //Secure tag found
                    secureTag = curTag;
                }
            }
    }

    printf("EXIT FINDMAXTAG FUNCTION \n");
    printf("--------------------------\n");

    //Check if you found the tag , otherwise you must return the max secure tag
    if(existTag != NULL)
    {
        return existTag;
    }
    else if(secureTag != NULL)
    {
        return secureTag;
    }
    else
    {
        return NULL;
    }

}

GHashTable *   deleteUnsecureTags(GHashTable * funmetatable ,TAG* secureTag , struct replicaHeader
*header )
{

   // printf("Enter delete  function insertMetadata\n");

    //Check if the filename exist in metadata table
    //otherwise you must fill the hasError variable

        //Store all the metadata information by fileID
        GPtrArray *pointArray=NULL;


        //Store all the tags file that you must
        //delete
        GPtrArray *delSet = NULL;
        //Initialize metadata array
        delSet = g_ptr_array_sized_new(10);

        char filename[256];

        bzero(filename,sizeof(filename));

        sprintf(filename,"%s_%ld",header->filename,header->fileid);


        printf("Debug: %s" , filename);


        //Retrieve all the tags which are associated with the specific filename
        pointArray =(GPtrArray*) g_hash_table_lookup(funmetatable,filename);

        if(pointArray != NULL)
        {
                //To point at the beginning of the list and to go through all the list
                //to detect if the tag that looking for exist. Otherwise, must return
                //the max secure tag
```

```
                GSList *iter = NULL;

                //A temp tag to go through the list tags
                TAG *curTag = NULL;

                //Go through all the list to find which tag are smaller from
                //the secure tags
                //So, you have to find all the smaller tags and delete it
              //  for (iter = pointlist; iter; )
                //{

            int i=0;
            for(i=0; i < pointArray->len; i++)
            {
                    //Point to each tag in list
                    curTag = (TAG *) g_ptr_array_index(pointArray ,i );

                    //Find all the tags that are smaller than the secure tag
                    //In this case, you must delete all the tags that are smaller
                    if (curTag->num < secureTag->num || curTag->num == secureTag->num && curTag-
>clientID < secureTag->clientID)
                    {
                        TAG* deletfiletag = ( TAG *)malloc(sizeof( TAG));
                        deletfiletag->num = curTag->num ;
                        deletfiletag->clientID = curTag->clientID ;
                        deletfiletag->isSecure = curTag->isSecure ;

                        //Insert the tag in the delete set
                        //You have to delete this tag

                        g_ptr_array_add(delSet, (gpointer) deletfiletag);

                        g_ptr_array_remove_index(pointArray, i);

                        free(curTag);

                        i=-1;
                    }

                }


        //Reuse temporary tag
        curTag = NULL;

        unsigned    int length;
        //Go through the list to delete all the tags
       // for (iter = delSet; iter; iter=iter->next)
        //{

        for(i=0; i < delSet->len; i++)
        {

            //Store the status of remove function
            int status;

            //Point to each tag in list
            curTag = (TAG *) g_ptr_array_index(delSet ,i );

            char delfile[256];
            bzero(delfile,sizeof(delfile));
            //ggg

            //Handle the case where file does not have
            //file extension
            if(strcmp(header->filetype ,"")==0)
            {
                length=sprintf(delfile,"%s_%ld_%d_%d_%d",header->filename,header->fileid,curTag->num
,curTag->clientID,curTag->isSecure );
            }
            else
            {
                length=sprintf(delfile,"%s_%ld_%d_%d_%d.%s",header->filename,header->fileid,curTag-
>num ,curTag->clientID,curTag->isSecure,header->filetype );
            }


            //delfile[length-1]='\0';

            if( access( delfile, F_OK ) != -1 )
```

```
            {
                //remove the file with the specific tag
                status=remove(delfile);

                //Check if remove correct the file
                if(status !=0 )
                {
                    printf("Unable to delete file:%s\n",delfile);
                    perror("Remove error");
                }
            }

        }


            for(i=0; i < delSet->len; i++)
            {
                //Point to each tag in list
                curTag = (TAG *) g_ptr_array_index(delSet ,i );

                    g_ptr_array_remove_index(delSet, i);

                    free(curTag);

                    i=-1;
                }

            }




        //Insert updated list in hashtable to the associated key(filename)
        g_hash_table_insert(funmetatable , header->filename , pointArray);



    //printf("Exit delete  function insertMetadata\n");


    return funmetatable;

}

GHashTable *   addSecure(GHashTable * funmetatable ,TAG* secureTag , struct replicaHeader *header )
{

    printf("Enter Add1 secure function insertMetadata\n");


    //Check if the filename exist in metadata table
    //otherwise you must fill the hasError variable
    //if(g_hash_table_contains(funmetatable , header->filename) == 1 )
    //{
        //Pointer to a list to retrieve all the tags for the
        //specific filename
       // GSList *pointlist = NULL;


        //Store all the metadata information by fileID
        GPtrArray *pointArray=NULL;


        int err=0;

        //Store all the tags file that you must
        //delete
        GSList *delSet = NULL;


        char filename[256];

        bzero(filename,sizeof(filename));

        sprintf(filename,"%s_%ld",header->filename,header->fileid);
```

```
        //Retrieve all the tags which are associated with the specific filename
        pointArray = (GPtrArray*) g_hash_table_lookup(funmetatable, filename);

        if(pointArray != NULL)
        {

               //To point at the beginning of the list and to go through all the list
               //to detect if the tag that looking for exist. Otherwise, must return
               //the max secure tag
               GSList *iter = NULL;

               //A temp tag to go through the list tags
               TAG *curTag = NULL;

               int ret;
               char oldname[255];
               char newname[255];
               unsigned    int length;

               //Clears buffers
               bzero(oldname,sizeof(oldname));
               bzero(newname,sizeof(newname));



            //Handle the case where file does not have
            //file extension
            if(strcmp(header->filetype ,"")==0)
            {
                length=sprintf(oldname,"%s_%ld_%d_%d_0",header->filename,header->fileid,header->tag-
>num,header->tag->clientID);
                //oldname[length-1]='\0';

                length = sprintf(newname,"%s_%ld_%d_%d_1" ,header->filename,header->fileid,header-
>tag->num,header->tag->clientID);
                //newname[length-1]='\0';
            }
            else
            {
                length=sprintf(oldname,"%s_%ld_%d_%d_0.%s",header->filename,header->fileid,header-
>tag->num,header->tag->clientID,header->filetype);
                //oldname[length-1]='\0';

                length = sprintf(newname,"%s_%ld_%d_%d_1.%s" ,header->filename,header->fileid,header-
>tag->num,header->tag->clientID,header->filetype);
                //newname[length-1]='\0';
            }

               //Go through all the list to find which tag are smaller from
               //the secure tags
               //So, you have to find all the smaller tags and delete it*/
              // for (iter = pointlist; iter; iter = iter->next )

            int i=0;
            for(i=0; i < pointArray->len; i++)
            {

                   //Point to each tag in list
                    curTag = (TAG *) g_ptr_array_index(pointArray ,i );

                   //Find all the tags that are smaller than the secure tag
                   //In this case, you must delete all the tags that are smaller
                   if (curTag->num == secureTag->num &&
                       curTag->clientID == secureTag->clientID && curTag->isSecure != 1 )
                   {
                      //Define the file as secure
                        curTag->isSecure=1;

                        //Rename the file with the secure tag
                         ret = rename(oldname, newname);

                       if(ret !=0)
                       {
                           printf("Error: unable to rename the file :%s",newname);
                       }

                       //If the found the file
                       break;
                   }
               }
```

```
        }


    printf("Exit Add secure function insertMetadata\n");
    printf("----------------------------------------\n");

    return funmetatable;

}

TAG* findSecureTag(GHashTable * funmetatable , char *filename )
{
    //Pointers for tags
    TAG *secureTag = malloc(sizeof(TAG));

    //Initialization of secureTag
    secureTag->num = 0;
    secureTag ->clientID = 0;
    secureTag ->isSecure = 0;

    int err=0;

    //Check if the filename exist in metadata table
    //otherwise you must fill the hasError variable
    //  if(g_hash_table_contains(funmetatable , filename) == 1 )
    //{

    //Pointer to a list to retrieve all the tags for the
    //specific filename
    //GSList *pointlist=NULL;


    //Store all the metadata information by fileID
    GPtrArray *pointArray=NULL;

    //Lock strtok due to is not deadlock free
    if(err=pthread_mutex_lock(&lockermetadata))
    {
        perror2("Failed to lock()",err);
    }


        //Retrieve all the tags which are associated with the specific filename
        pointArray =(GPtrArray*) g_hash_table_lookup(funmetatable,filename);

        if(pointArray != NULL)
        {

                //To point at the beginning of the list and to go through all the list
                //to detect if the tag that looking for exist. Otherwise, must return
                //the max secure tag
             //   GSList* iter=NULL;



                //A temp tag to go through the list tags
                TAG* curTag  = NULL;

              //  for(iter=pointlist; iter; iter = iter->next)
            int i=0;
            for(i=0; i < pointArray->len; i++)
            {
                    //Point to each tag in list
                    curTag = (TAG *) g_ptr_array_index(pointArray ,i );

                    //Check if the tag that looking for exist in the list
                    if( (curTag->num > secureTag->num && curTag ->isSecure == 1 ) || (curTag->num ==
secureTag->num && curTag->clientID > secureTag->clientID && curTag->isSecure == 1) )
                    {
                        //Tag found
                        secureTag ->isSecure = curTag->isSecure;
                        secureTag->clientID = curTag ->clientID;
                        secureTag->num = curTag ->num;
                    }
                }
            }

    //Unloack Mutex//
    if(err=pthread_mutex_unlock(&lockermetadata))
```

```
    {
        perror2("Failed to lock()",err);
    }

        if(secureTag->num == 0)
        {
            return NULL;
        }
        else
        {
            return secureTag;
        }

}

int isTagExist(GHashTable * funmetatable , char *filename , TAG* secureTag )
{

    //Initialize with the value -1. Means that tag does not exist
    int isFound = -1;

    //Pointer to a list to retrieve all the tags for the
    //specific filename
    GSList *pointlist=NULL;

    //Retrieve all the tags which are associated with the specific filename
    pointlist =(GPtrArray*) g_hash_table_lookup(funmetatable,filename);

    if(pointlist != NULL)
    {
        //To point at the beginning of the list and to go through all the list
        //to detect if the tag that looking for exist. Otherwise, must return
        //the max secure tag
        GSList* iter=NULL;

        //A temp tag to go through the list tags
        TAG* curTag  = NULL;

        for(iter=pointlist; iter; iter = iter->next)
        {
            //Point to each tag in list
            curTag = (TAG *) iter->data;

            //Check if the tag that looking for exist in the list
            if( ( curTag->num == secureTag->num && curTag->clientID == secureTag->clientID ))
            {
                secureTag->isSecure =1;
                isFound = 1;
            }
        }

        //Find the max secure tag
        TAG *maxSecureTag = findSecureTag(funmetatable , filename);
        if(maxSecureTag != NULL )
        {
            if(maxSecureTag ->num < secureTag->num && isFound == -1 || maxSecureTag ->num == secureTag
->num && maxSecureTag ->clientID < secureTag ->clientID && isFound ==-1 )
            {
                return -1;
            }
            else
            {
                return 1;
            }
        }
        else
        {
            return isFound;

        }
    }
    else
    {
        //Return -1 in case where the tag does not exist
        return -1;
    }

}

void *bind_thread(void *port)
```

```
{
    int PORT=(intptr_t)port;

    //Socket descriptor for the Replica
    int         sock;
    //Store Client socket.
    int         newsock;
    //The size of the details for the accepted client
    int         clientlen;

    struct      sockaddr *clientPtr;

    struct      sockaddr_in serv_addr;
    //Declare&Initialize sockaddr_in pointer for accept function
    struct      sockaddr_in client_addr;
    struct      sockaddr *servPtr;

    //for setsockopt to be possible to reuse the port
    int allow=1;

    //Store pthread ID
    pthread_t tid;
    //Store threads errors
    int err;

    /*Initialize socket structure */
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
    serv_addr.sin_port=htons((intptr_t)port);

    //Point to struct serv_addr.
    servPtr=(struct sockaddr *) &serv_addr;

    /*Create a socket and check if is created correct.*/
    if((sock=socket(PF_INET,SOCK_STREAM,0)) < 0)
    {
        perror("Socket() failed"); exit(1);
    }

    //Allow to reuse the port
    if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &allow, sizeof(int)) == -1)
    {
        perror("\nsetsockopt\n");
        close(sock);
        pthread_exit((void *) 0);
    }


    //Bind socket to address //
    if(bind(sock,servPtr, sizeof(serv_addr)) < 0)
    {
        perror("Bind() failed");
        exit(1);
    }

    /*Listen for connections */
    /* MAXCLIENT. requests in queue*/
    if(listen(sock , MAXCLIENT) < 0)
    {
        perror("Listen() failed"); exit(1);
    }

    printf("\n*************Start to listen to port:%d******************\n" , PORT);

    //Accept connection from clients forever.
    while(1)
    {

        clientPtr=(struct sockaddr *) &client_addr;
        clientlen= sizeof(client_addr);

        if((newsock=accept(sock , clientPtr  , &clientlen )) < 0)
        {
            perror("accept() failed"); exit(1);
        }


        if(err=pthread_create(&tid , NULL , &accept_thread ,  (void *)(intptr_t) newsock))
        {
            perror2("pthread_create for accept_thread" , err);
```

```
            exit(1);
        }


    }//While(1)

}

void *accept_thread(void *accept_sock)
{
    //socket descriptor for each client
    int acptsock;
    //Retrieve the socket that passed from pthread_create
    acptsock= (intptr_t)(accept_sock);

    int         bytes;
    int         len;
    int         file_size;
    char        *filename=NULL;

    struct  replicaHeader *msg;
    char    buf[BUFSIZE];

    //While the client is connect to the system you have to keep the connection
    while(1)
    {
        //If connection is established then start communicating //
        //Initialize buffer with zeros //
        bzero(buf,sizeof(buf));
        //Waiting...to receive data from the client//
        if (recv(acptsock, buf, 512, 0) < 0)
        {
            perror("send() failed");
            close(acptsock);
            pthread_exit((void *) 0);
        }

        //Check if direc received a message
        if(strlen(buf) != 0)
        {
            //Show the message that received
            printf("---------------------------------\n");
            printf("accept_thread received: %s\n", buf);
        }
        else if(strlen(buf) == 0)
        {
             //printf("Received an empty message!\n");
            //close(acptsock);
             pthread_exit((void *) 0);
        }

        //Retrieve the message from the client
        msg = decode(buf);

        bzero(buf,sizeof(buf));
        //bzero(buf,sizeof(buf));
        //Received the actual data for the file

        if( strcmp(msg->type,"WRITE") == 0 )
        {
              //To store if received the data object successful or not
              int isSuccess=-1;
              int sendSize=0;


              char filename[256];

              bzero(filename,sizeof(filename));

              sprintf(filename,"%s_%ld",msg->filename,msg->fileid);

              //Receive the file
              isSuccess=ftp_recv(acptsock,msg);

              if(isSuccess == SUCCESS)
              {
                  //Store the metadata of new the file in the hashtable
                  metadatatable =  insertMetadata(metadatatable , msg->tag , filename);

                  //Send an acknowledge to the client that received the object
```

```
                bzero(buf,sizeof(buf));

                //The format of the acknowledgment
                sprintf(buf,"WRITE-OK,%d,%s", msg->msgID, msg->filename);
                printf("%s\n",buf);
                sendSize=strlen(msg->filename) + sizeof(int) + 13;
                //sent the acknowledgment that received the file
                if (send(acptsock, buf, 512, 0) < 0)
                {
                    perror("WRITE-send() failed\n");
                }
            }
        }
        else if(strcmp(msg->type,"READ") == 0 )
        {
            int error=0;
            int err=0;

            char filename[256];

            bzero(filename,sizeof(filename));

            sprintf(filename,"%s_%ld",msg->filename,msg->fileid);

            //Retrieve curTag
            TAG *curTag=NULL;

            //Check if the tag exist in the system. If exist return the tag or
            //retrun the latest tag that is secure
            //Lock strtok due to is not deadlock free
            if(err=pthread_mutex_lock(&lockermetadata))
            {
                perror2("Failed to lock()",err);
            }

               curTag = findMaxTag( metadatatable , msg->tag , filename , &error);



            if(curTag !=NULL)
            {

                msg->tag->num = curTag->num;
                msg->tag->clientID = curTag->clientID;
                msg->tag->isSecure = curTag->isSecure;

                printf("ReadOperation, FoundMax, Tag:%d,clientID:%d,isSecure:%d\n", msg->tag->num,
msg->tag->clientID,
                       msg->tag->isSecure);


                //debug
                printf("ReadDebug: %s,%s\n",msg->filename,msg->filetype);

                //Send the file to client
                send2ftp(acptsock, msg);
            }



            //Unloack Mutex//
            if(err=pthread_mutex_unlock(&lockermetadata))
            {
                perror2("Failed to unlock()",err);
            }

        }

        else if(strcmp(msg->type,"SECURE") == 0 )
        {
            //Store error of pthread
            int err;

            //Lock strtok due to is not deadlock free
            if(err=pthread_mutex_lock(&lockermetadata))
            {
                perror2("Failed to lock()",err);
            }
                  metadatatable = addSecure(metadatatable,msg->tag,msg);
```

```
                //temp
                msg->tag->isSecure=1;


                 metadatatable =  deleteUnsecureTags(metadatatable , msg->tag , msg);

            //Unloack Mutex//
            if(err=pthread_mutex_unlock(&lockermetadata))
            {
                perror2("Failed to lock()",err);
            }
        }

        //clear buffer
        bzero(buf,sizeof(buf));

        //deallocate
        free(msg->filename);
       // free(msg->filetype);
        free(msg->tag);
        free(msg);

    }//While
}

void initialization()
{
    //Initialize metadata table and also determine hash function
    metadatatable = g_hash_table_new( g_str_hash, g_str_equal);

    //Initialize metadata array
    metatable = g_ptr_array_sized_new(1000);


    /*Initialization of mutex for strtok in decode function*/
    pthread_mutex_init(&locker,NULL);

    //Initialization of mutex for lock the metadata table
    pthread_mutex_init(&lockermetadata,NULL);

    int i=0;

    //delete
   /* int i=0;
    TAG *mytag = g_new(TAG, 1);
    for(i=0; i<5; i++)
    {
        mytag->num = i;
        mytag->clientID = 0;
        mytag->isSecure = 0;

        metadatatable = insertMetadata(metadatatable, mytag, "m");
    }

    mytag->num = 3;
    mytag->clientID = 0;
    mytag->isSecure = 1;

    metadatatable = insertMetadata(metadatatable, mytag, "110");*/


    //Allocate the array to store all the sockaddr_in for directories
    direc_sockaddr=(struct sockaddr_in *)malloc(sizeof(struct sockaddr_in ) * MAX_DIRECTORIES );

    //Allocate the array to store all the sockaddr_in for Replicas
    replica_sockaddr=(struct sockaddr_in *)malloc(sizeof(struct sockaddr_in ) * MAX_REPLICAS );

    //Allocate the array to store all the sockets descriptor for each directory
    direcSocks=( int *)malloc(sizeof(int ) * MAX_DIRECTORIES );

    //Allocate the array to store all the sockets descriptor for each Replicas
    replicaSocks=(int *)malloc(sizeof(int ) * MAX_REPLICAS );


    //Fill the direcSocks for directories server which is type sockaddr_in
    for (i = 0; i < MAX_DIRECTORIES; i++)
    {

            direc_sockaddr[i].sin_family = AF_INET; /*Internet domain*/
            direc_sockaddr[i].sin_addr.s_addr = inet_addr(direcNodes[i].ip_addr);
```

```
            direc_sockaddr[i].sin_port = htons(direcNodes[i].port);


    }//For statement

    //Fill the replicaSocks for directories server which is type sockaddr_in
    for (i = 0; i < MAX_REPLICAS; i++)
    {

        if( NODEID != replicaNodes[i].port  )
        {

            replica_sockaddr[i].sin_family = AF_INET; /*Internet domain*/
            replica_sockaddr[i].sin_addr.s_addr = inet_addr(replicaNodes[i].ip_addr);
            replica_sockaddr[i].sin_port = htons(replicaNodes[i].port);
        }
    }//For statement

    /*Go through all the available directories to establish a connection.*/
    for (i = 0; i < MAX_DIRECTORIES; i++)
    {
            //Create a socket and check if is created correct
            if ((direcSocks[i] = socket(PF_INET, SOCK_STREAM, 0)) < 0)
            {
                perror("Socket() failed");
                exit(1);
            }

    }//For statement

    //Fill the replicaSocks for directories server which is type sockaddr_in
    for (i = 0; i < MAX_REPLICAS; i++)
    {
        if( NODEID != replicaNodes[i].port  )
        {
            //Create a socket and check if is created correct
            if ((replicaSocks[i] = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
                perror("Socket() failed");
                exit(1);
            }
        }

    }//For statement

}

void signal_handler()
{
    printf("Signal Handled here\n");

    //Retrieve all the key values from the hashtable
    GList *hashPoint =  g_hash_table_get_keys (metadatatable);

    GSList* iter=NULL , *iter2=NULL , *pointlist=NULL;

    TAG *tempTag=NULL;

    for(iter=hashPoint; iter; iter = iter->next)
    {
        // printf("key:%s\n" , (char *)iter->data);

        //Retrieve all the tags which are associated withe the specific filename
        pointlist =(GSList*) g_hash_table_lookup(metadatatable, iter->data );

        if(pointlist != NULL)
        {
            for(iter2=pointlist; iter2; iter2 = iter2->next)
            {
                tempTag = ( TAG*) iter2->data;

                printf("key:%s , Num:%d, ClientID:%d \n" , (char *)iter->data ,tempTag->num , tempTag-
>clientID );
                free(tempTag);
            }
            g_slist_free(pointlist);
        }

        //  tempTag = ( TAG*) iter->data;
        //  printf("Num:%d, ClientID:%d \n" , tempTag->num , tempTag->clientID );
```

```
        /*for(tagList=iter; tagList; tagList = tagList->next)
        {
            tempTag = ( TAG*) tagList->data;

            printf("Num:%d, ClientID:%d \n" , tempTag->num , tempTag->clientID );

        }*/
    }

    //Destroy HashTable
    g_hash_table_destroy(metadatatable);

    //exit the server
    exit(0);
}

/*****************************************************************************/
/*                                  MAIN                                     */
/*****************************************************************************/
//---------------------------------------------------------------------------//
int main(int argc , char  *argv[])
{

/*****************************************************************************/
/*                                LOCAL VARIABLES                            */
/*****************************************************************************/

    //Store pthread ID
    pthread_t tid;
    //Store threads errors
    int         err;
    int         port;


    //Check of input arguments
    if(argc !=3)
    {
        printf("\nUsage: argv[0] -p [port]\n");
        exit(-1);
    }

    printf("\nPID: %d\n", getpid());

    //Initialization functions
    initialization();

    //Retrieve input parameters
    port=atoi(argv[2]);
    NODEID= port;

    printf("--------------------------------------\n");
    printf("\nStarting Replica on port:%d ....\n" , port);
    printf("--------------------------------------\n");



    // SIGINT is signal name create  when Ctrl+c will pressed
    signal(SIGINT,signal_handler);
    //Handle segmentation corrupt
    signal(SIGSEGV, signal_handler);

    //Create a thread for the bind.
    if(err=pthread_create(&tid , NULL ,(void *) &bind_thread , (void *)(intptr_t)port))
    {
        perror2("pthread_create", err);
        exit(1);
    }

    //Wait until all threads to finish. Normally Bind thread
    //is always running
    pthread_join(tid , (void *) 0);

    return 0;
}//Main Function
```

Replica.h

```
/*****************************************************************************/
/*                                                                           */
/*Name: Elias Spanos                                                         */
/*Date: 09/10/2015                                                           */
/*Filename:Replica.h                                                         */
/*                                                                           */
/*****************************************************************************/
#ifndef DISTRIBUTEDALGORITHMS_REPLICA_H
#define DISTRIBUTEDALGORITHMS_REPLICA_H

#include <glib.h>

/*****************************************************************************/
/*                              FUNCTION PROTOTYPES                          */
/*****************************************************************************/
void *bind_thread(void *port);
void *accept_thread(void *accept_sock);
void *gossip(void *header );

typedef struct
{
    unsigned  int num;
    unsigned int clientID;
    unsigned int isSecure;

}TAG;

typedef struct
{
    char *filename;
    int fileid;
    TAG *tag;

}METADATA;


struct replicaHeader
{
    char *type;
    char  *filename;
    char *filetype;
    long fileid;
    TAG *tag;
    int msgID;
    int fileSize;
    char *checksum;
};

struct serinfo
{
    char ip_addr[16];
    int port;
};

/*****************************************************************************/
/*                            PROTOTYPES                                     */
/*****************************************************************************/

void readConfig(char *filename , int port);

char *checksum_get(char *filename);

void conn2direc();

int connect2Replicas();

struct replicaHeader* decode(char *buf );

int send2ftp(int newsock , struct replicaHeader *msg );

int ftp_recv(int sock, struct replicaHeader *msg );

GHashTable *  insertMetadata(GHashTable * funmetatable ,  TAG* filetag , char *filename );

TAG* findMaxTag(GHashTable * funmetatable , TAG* filetag , char *filename  , int *hasError);

GHashTable *   deleteUnsecureTags(GHashTable * funmetatable ,TAG* secureTag , struct replicaHeader
*header);
```

```
GHashTable *   addSecure(GHashTable * funmetatable ,TAG* secureTag , struct replicaHeader *header );

TAG* findSecureTag(GHashTable * funmetatable , char *filename );

int isTagExist(GHashTable * funmetatable , char *filename , TAG* secureTag );

void *bind_thread(void *port);

void *accept_thread(void *accept_sock);

void initialization();

void signal_handler();


#endif //DISTRIBUTEDALGORITHMS_REPLICA_H
```

Filemanager.c

```
/*****************************************************************************/
/*                                                                           */
/*Name: Elias Spanos                                                         */
/*Date: 16/02/2016                                                           */
/*Filename: filemanager.c                                                    */
/*                                                                           */
/*****************************************************************************/
/*****************************************************************************/
/*                                  LIBRARIES                                */
/*****************************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>

#include "filemanager.h"

/*****************************************************************************/
/*                                  MACROS                                   */
/*****************************************************************************/
//---------------------------------------------------------------------------//
#define MAXCLIENT 10000

//Format for error for the threads
#define perror2(s,e) fprintf(stderr, "%s:%s\n" , s, strerror(e))

/*****************************************************************************/
/*                              DATA STRUCTURES                              */
/*****************************************************************************/
//---------------------------------------------------------------------------//
//Store all usernames for the clients
GPtrArray *clidata=NULL;

//Store all metadata for the clients
GPtrArray *metadata=NULL;

/*****************************************************************************/
/*                                  LOCKERS                                  */
/*****************************************************************************/
//---------------------------------------------------------------------------//
//Locker clientCounter
pthread_mutex_t lockercliCoun;
pthread_mutex_t lockerFileids;


//locker for the decode function
pthread_mutex_t locker;

/*****************************************************************************/
/*                                FUNCTIONS                                  */
/*****************************************************************************/
//---------------------------------------------------------------------------//
```

```
/*****************************************************************************/
/*                                  DECODE                                   */
/*****************************************************************************/
int decode(char *buf , FILEHEADER *header )
{
    //Store the thread error if exist
    int err;

    //Lock strtok due to is not deadlock free//
    if(err=pthread_mutex_lock(&locker))
    {
        perror2("Failed to lock()",err);
    }
         //Use comma delimiter to find the type of
         //send message and retrieve the apropriate
         // fields//
         header->type=strdup(strtok(buf, ","));

        if( strcmp(header->type , "REQCLIENTID" )== 0)
        {
            header->username=strdup(strtok(NULL,","));
            header->MSGID = atol( strtok(NULL,",") );
        }
        else if( strcmp(header->type , "REQCREATE" )== 0)
        {
            header->filename = strdup(strtok(NULL,","));
            header->owner = atol( strtok(NULL,","));
            header->MSGID  = atol(strtok(NULL,","));
        }
        else if( strcmp(header->type , "REQID" )== 0)
        {
            header->filename = strdup(strtok(NULL, ","));
        }
        else if( strcmp(header->type , "REQFILEID" )== 0)
        {
            header->filename = strdup(strtok(NULL,","));
            header->owner = atol( strtok(NULL,","));
            header->MSGID = atol( strtok(NULL,","));
        }
        else if( strcmp(header->type , "REQFILESLIST" )== 0)
        {
            header->MSGID = atol( strtok(NULL,","));
        }

    //Unloack Mutex//
    if(err=pthread_mutex_unlock(&locker))
    {
        perror2("Failed to lock()",err);
    }

}


/*****************************************************************************/
/*                              REQUEST FILE LIST                            */
/*****************************************************************************/
GString *getfilelist(GString* strset)
{
    int i;

    strset = g_string_new(NULL);

    //Point to metadata array
    METADATA *point2meta = NULL;

    char strtemp[4096];

    //Initialize the set
    bzero(strtemp,sizeof(strtemp));

    //Set the total size of the set
    sprintf(strtemp,"%d",metadata->len);
    g_string_append(strset , strtemp);

    //Deallocate all the memory
    for(i=0; i < metadata->len; i++)
    {
        //Retrieve  the data from the specific index
        point2meta = (METADATA *) g_ptr_array_index(metadata , i );
```

```
        //Create file list
        sprintf(strtemp,",%s,%lu,%ld",point2meta->filename->str , point2meta->fileid , point2meta->owner);

        g_string_append(strset , strtemp);
    }

    return strset;

}


/*****************************************************************************/
/*                                BIND_THREAD                                */
/*****************************************************************************/
void *bind_thread(void *port)
{
    int PORT=(intptr_t)port;

    //Socket descriptor for the Replica
    int         sock;
    //Store Client socket.
    int         newsock;
    //The size of the details for the accepted client
    int         clientlen;

    struct      sockaddr *clientPtr;

    struct      sockaddr_in serv_addr;
    //Declare&Initialize sockaddr_in pointer for accept function
    struct      sockaddr_in client_addr;
    struct      sockaddr *servPtr;

    //for setsockopt to be possible to reuse the port
    int allow=1;

    //Store pthread ID
    pthread_t tid;
    //Store threads errors
    int err;

    /*Initialize socket structure */
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
    serv_addr.sin_port=htons((intptr_t)port);

    //Point to struct serv_addr.
    servPtr=(struct sockaddr *) &serv_addr;

    /*Create a socket and check if is created correct.*/
    if((sock=socket(PF_INET,SOCK_STREAM,0)) < 0)
    {
        perror("Socket() failed"); exit(1);
    }

    //Allow to reuse the port
    if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &allow, sizeof(int)) == -1)
    {
        perror("\nsetsockopt\n");
        close(sock);
        pthread_exit((void *) 0);
    }


    //Bind socket to address //
    if(bind(sock,servPtr, sizeof(serv_addr)) < 0)
    {
        perror("Bind() failed");
        exit(1);
    }

    //Listen for connections //
    // MAXCLIENT. requests in queue//
    if(listen(sock , MAXCLIENT) < 0)
    {
        perror("Listen() failed"); exit(1);
    }

    printf("\n**************Start to listen to port:%d******************\n" , PORT);
```

```
    //Accept connection from clients forever.
    while(1)
    {
        clientPtr=(struct sockaddr *) &client_addr;
        clientlen= sizeof(client_addr);

        if((newsock=accept(sock , clientPtr  , &clientlen )) < 0)
        {
            perror("accept() failed"); exit(1);
        }

        if(err=pthread_create(&tid , NULL , &accept_thread ,  (void *) (intptr_t)newsock))
        {
            perror2("pthread_create for accept_thread" , err);
            exit(1);
        }

    }//While(1)

}

/*****************************************************************************/
/*                                ACCEPT_THREAD                              */
/*****************************************************************************/
void *accept_thread(void *accept_sock)
{
    //socket descriptor for each client
    int acptsock;
    //Retrieve the socket that passed from pthread_create
    acptsock= (intptr_t)accept_sock;

    //Buffer to send&receive data//
    char buf[256];

    //Received bytes
    int bytes;

    //Declare the structure of the sending message in order
    //the filemanager to communicate with the client and vice versa
    FILEHEADER *msg = (FILEHEADER *)malloc(sizeof(FILEHEADER));


    //While the client is connect to the system you have to keep the connection
    while(1)
    {
        //If connection is established then start communicating //
        //Initialize buffer with zeros //
        bzero(buf, sizeof(buf));
        //Waiting...to receive data from the client//
        if (recv(acptsock, buf, sizeof(buf), 0) < 0)
        {
            perror("Received() failed!");
            close(acptsock);
            pthread_exit((void *) 0);
        }

        //Check if direc received a message
        if(strlen(buf) != 0)
        {
            //Show the message that received//
            printf("---------------------------------\n");
            printf("accept_thread received: %s\n", buf);
        }
        else if (strlen(buf) == 0)
        {
            //printf("Unable to received messsage!\n");
            close(acptsock);
            pthread_exit((void *) 0);
        }

        //Decode the message that receive from the client
        //in order to identify the type of the message//
        decode(buf,msg);

        if( strcmp(msg->type , "REQCLIENTID" )== 0)
        {
            bzero(buf,sizeof(buf));
            //encode the clientID
            sprintf(buf,"REQCLIENTID,%ld,%ld" , registerClient(msg->username ), msg->MSGID );
            if (send(acptsock, buf, sizeof(buf) , 0) < 0 )
```

```
    {
        perror("Send:Unable to send clientID");
    }

    //Deallocations
    free(msg->username);
}


else if( strcmp(msg->type , "REQCREATE" )== 0 )
{
    printf("Received Create: %s , owner:%ld\n", msg->filename, msg->owner);

    //Store fileid
    unsigned long fileid = lookUpFileID(msg->filename , msg->owner );

    if(fileid == -1)
    {
        //Store the new file in the metadata
        //Also , retrieve the fileID for the file
        fileid = registerFile(msg->filename, msg->owner);
    }

    printf("REQCREATE Function return: %ld \n", fileid);

    bzero(buf, sizeof(buf));
    //encode the clientID
    sprintf(buf, "REQCREATE,%ld,%ld", fileid, msg->MSGID);
    if (send(acptsock, buf, sizeof(buf), 0) < 0)
    {
        perror("Send:Unable to send clientID");
    }
    //Deallocations
    free(msg->filename);

}
else if( strcmp(msg->type , "REQFILEID" )== 0 )
{
    printf("Received REQFILEID: %s , owner:%ld\n" , msg->filename ,msg->owner);

    //Store the new file in the metadata
    //Also , retrieve the fileID for the file
    unsigned long fileid = lookUpFileID(msg->filename , msg->owner );

    if(fileid == -1)
    {
        //Store the new file in the metadata
        //Also , retrieve the fileID for the file
        fileid = registerFile(msg->filename, msg->owner);
    }

    printf("REQFILEID Function return: %ld \n" , fileid);

    bzero(buf,sizeof(buf));
    //encode the clientID
    sprintf(buf,"REQFILEID,%ld,%ld" , fileid , msg->MSGID );
    if (send(acptsock, buf, sizeof(buf) , 0) < 0 )
    {
        perror("Send:Unable to send clientID");
    }

    //Deallocations
    free(msg->filename);
}
else if( strcmp(msg->type , "REQFILESLIST" )== 0 )
{
    bzero(buf,sizeof(buf));

    GString *list=NULL;

    list=getfilelist(list);
    printf("List\n");

    if (send(acptsock, list->str, strlen(list->str) , 0) < 0 )
    {
        perror("Send:Unable to send clientID");
    }

    //Deallocate memory
    g_string_free(list,TRUE);
```

```
        }


    }//While
}


/*****************************************************************************/
/*                                REGISTER CLIENT                            */
/*****************************************************************************/
long registerClient(char *username)
{
    //Counter for the index
    int i=0;

    //Store the error of mutex
    int err;

    //Check if the username exist in the data
    int isFound=0;

    //Pointer to the metadata
     CLIENT *point2client = NULL;

    //Check if the username exist in the array
    for(i=0; i < clidata->len; i++)
    {
        //Retrieve  the data from the specific index
        point2client = ( CLIENT *) g_ptr_array_index(clidata ,i );

        if(strcmp(username , point2client->username->str)==0)
        {
            //Indicate that username exist in the data
            isFound=1;

            //Stop the loop
            break;
        }
    }



    if(isFound==0)
    {
        //Allocate memory for the new entry
        CLIENT *entry = (CLIENT *)malloc(sizeof(CLIENT));

        //Initialize gstring
        entry->username = g_string_new(NULL);

        //Insert the username the metadata table
        g_string_assign( entry->username , g_strdup(username));

        //Lock strtok due to is not deadlock free
        if(err=pthread_mutex_lock(&lockercliCoun))
        {
            perror2("Failed to lock()",err);
        }
            //Increase client ID
            countClientIds +=1;

            //Store new client ID in the struct
            entry->client_id = countClientIds;

        //Unloack Mutex
        if (err = pthread_mutex_unlock(&lockercliCoun))
        {
            perror2("Failed to lock()", err);
        }

        //insert new entry in client data table
        g_ptr_array_add(clidata, (gpointer) entry );

        return entry->client_id;
    }

    //If it found the client Id return it
    return  point2client->client_id;
}
```

```
/*****************************************************************************/
/*                                REGISTER FILE                              */
/*****************************************************************************/
long registerFile(char *filename , long owner)
{
    //Counter for the index
    int i=0;

    //Store the error of mutex
    int err;

    //Store the fileid
    long tmpfileid=0;

    //Allocate memory for the new entry
    METADATA *entry = (METADATA *)malloc(sizeof(METADATA));

    //Initialize gstring
    entry->filename = g_string_new(NULL);

    //Insert the filename the metadata table
    g_string_assign( entry->filename , g_strdup(filename));

    //Insert create of the file in the metadata
    entry->owner = owner;

    //Lock strtok due to is not deadlock free
    if(err=pthread_mutex_lock(&lockerFileids))
    {
        perror2("Failed to lock()",err);
    }
        //Increase counter for fileIds
        countFileIds +=1;

        //Copy fileid for consistence purpose
        tmpfileid = countFileIds;

        printf("Function registerFile: %ld\n" , countFileIds);

    //Store new client ID in the struct
    entry->fileid = tmpfileid;

    //Unloack Mutex
    if (err = pthread_mutex_unlock(&lockerFileids))
    {
        perror2("Failed to lock()", err);
    }

    //insert new entry in client data table
    g_ptr_array_add(metadata, (gpointer) entry );

    printf("entry fileid: %ld\n" , entry->fileid);

    //Return the new fileID for the file
    return  entry->fileid;
}


/*****************************************************************************/
/*                                LOOKUPFILEID                               */
/*****************************************************************************/
long lookUpFileID(char *filename , long owner)
{
    //Counter for the index
    int i=0;

    //Store the error of mutex
    int err;

    //Check if the username exist in the data
    int isFound=0;

    //Look up for the fileid for the parameter filename
    long tmpfileid=-1;

    //Pointer to the metadata
    METADATA *point2meta = NULL;

    //Check if the username exist in the array
```

```
    for(i=0; i < metadata->len; i++)
    {
        //Retrieve  the data from the specific index
        point2meta = ( METADATA *) g_ptr_array_index(metadata ,i );

        //Check if it found the filename that it looking for
        if(strcmp(filename , point2meta->filename->str)==0)
        {
            tmpfileid = point2meta->fileid;

            //Stop the loop
            break;
        }
    }

    return tmpfileid;
}

/*****************************************************************************/
/*                                INITIALIZATION                             */
/*****************************************************************************/
void initialization()
{
    //Initialize metadata array
    clidata = g_ptr_array_sized_new(10);

    //Store metadata for all the files
    metadata = g_ptr_array_sized_new(100);

    //initialize clientsIDS
    countClientIds=0;

    //Initialize fileIDS
    countFileIds=0;

    //Initialization of mutex for strtok in decode function//
    pthread_mutex_init(&locker,NULL);

    //Initialization of mutex for lock the metadata table
    pthread_mutex_init(&lockercliCoun,NULL);

    //Initialization of mutex for lock the metadata table
    pthread_mutex_init(&lockerFileids,NULL);

}

/*****************************************************************************/
/*                                SIGNAL_HANDLER                             */
/*****************************************************************************/
void signal_handler()
{
    printf("Signal Handled here\n");

    int i=0;

    printf("\nd*************************************\n");
    printf("CLIENTS CREDENTIALS\n");
    printf("*************************************\n");

    //Pointer to the metadata
    CLIENT *point2cli = NULL;

    //Deallocate all the memory
    for(i=0; i < clidata->len; i++)
    {
        //Retrieve  the data from the specific index
        point2cli = (CLIENT *) g_ptr_array_index(clidata , i );

        //deallocations
        printf("Username: %s , ClientID: %ld \n" , point2cli->username->str , point2cli->client_id );

        //deallocate filename
        g_string_free (point2cli->username, TRUE);

        //deallocate struct
        free(point2cli);
    }

    //Point to metadata array
    METADATA *point2meta = NULL;
```

```
    printf("\n**************************************\n");
    printf("FILES METADATA CREDENTIALS\n");
    printf("**************************************\n");

    //Deallocate all the memory
    for(i=0; i < metadata->len; i++)
    {
        //Retrieve  the data from the specific index
        point2meta = (METADATA *) g_ptr_array_index(metadata , i );

        //deallocations
        printf("Filename: %s , Fileid: %lu , owner: %ld \n" , point2meta->filename->str , point2meta-
>fileid , point2meta->owner );

        //deallocate filename
        g_string_free (point2meta->filename , TRUE);

        //deallocate struct
        free(point2meta);
    }

    //exit the server
    exit(0);
}
//-------------------------------------------------------------------------------------//

int main(int argc , char  *argv[])
{

/*****************************************************************************/
/*                                LOCAL VARIABLES                            */
/*****************************************************************************/

    //Store pthread ID
    pthread_t tid;
    //Store threads errors
    int         err;
    int         port;

    //Check of input arguments
    if(argc !=3)
    {
        printf("\nUsage: argv[0] -p [port]\n");
        exit(-1);
    }

    //Initialize
    initialization();

    //Retrieve input parameters
    port=atoi(argv[2]);

    printf("--------------------------------------\n");
    printf("\nStarting File manager on port:%d ....\n" , port);
    printf("--------------------------------------\n");

    // SIGINT is signal name create  when Ctrl+c will pressed
    signal(SIGINT,signal_handler);
    //Handle segmentation corrupt
    signal(SIGSEGV, signal_handler);

    //Create a thread for the bind.
    if(err=pthread_create(&tid , NULL ,(void *) &bind_thread , (void *)(intptr_t)port))
    {
        perror2("pthread_create", err);
        exit(1);
    }

    //Wait until all threads to finish. Normally Bind thread
    //is always running
    pthread_join(tid , (void *) 0);

    return 0;
}//Main Function
```

Filemanager.h

```
//
// Created by elias on 2/17/16.
//

#ifndef DISTRIBUTEDALGORITHMS_FILEMANAGER_H
#define DISTRIBUTEDALGORITHMS_FILEMANAGER_H

#include <glib.h>

typedef struct
{
    long client_id;
    GString *username;

}CLIENT;

typedef struct
{
    char *type;
    long MSGID;
    char *filename;
    char *filetype;
    char *username;
    long owner;

}FILEHEADER;

typedef struct
{
    GString *filename;
    long fileid;
    long owner;
    GString *username;
}METADATA;


/*****************************************************************************/
/*                                VARIABLES                                  */
/*****************************************************************************/
//Counter for client IDs
long countClientIds;

//Counter for File IDs
long countFileIds;


/*****************************************************************************/
/*                              PROTOTYPES                                   */
/*****************************************************************************/

int decode(char *buf , FILEHEADER *header );

GString *getfilelist(GString* strset);

void *bind_thread(void *port);

void *accept_thread(void *accept_sock);

long registerClient(char *username);

long registerFile(char *filename , long owner);

long lookUpFileID(char *filename , long owner);

void initialization();

void signal_handler();


#endif //DISTRIBUTEDALGORITHMS_FILEMANAGER_H
```

Client.h

```
//
// Created by elias on 9/29/15.
//

#ifndef DISTRIBUTEDALGORITHM_CLIENT_H
#define DISTRIBUTEDALGORITHM_CLIENT_H

#include <glib.h>

/***********************************/
/*           STRUCTS               */
/***********************************/

struct TAG
{
    long num;
    long clientID;
};

struct message
{
    char *type;
    long  msg_id;
    struct TAG tag;
    GSList* replicaSet;
    long  fileID;
    char *filename;
    char *filetype;
    long clientID;
    unsigned int fileSize;
    char *checksum;
    char *permission;
};

struct cmd
{
    char oper[9];
    char *filename;
    char *fileType;
    long fileid;
};

struct serinfo
{
    char ip_addr[16];
    int port;
};

int reqClientID(char *username);

int get_file(int sock, struct message *msg );

int send2ftp(struct cmd *msgCmd, int newsock , struct TAG *tagIn , long msgIDIn );

/*****************************************************************************/
/*                              FUNCTION PROTOTYPES                          */
/*****************************************************************************/

long reqCreate(char *filename);
long reqFileID(struct cmd *cmdmsg , long clientID);

#endif //DISTRIBUTEDALGORITHM_CLIENT_H
```

Client.c

```
/*****************************************************************************/
/*                                                                           */
/*Name: Elias Spanos                                                         */
/*Date: 30/09/2015                                                           */
/*Filename: client.c                                                         */
/*                                                                           */
/*****************************************************************************/
/*****************************************************************************/
/*                                  LIBRARIES                                */
/*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/sendfile.h>
#include <arpa/inet.h>
#include <unistd.h>
#include "client.h"
#include <time.h>
#include <error.h>
#include <gtk/gtk.h>

/*****************************************************************************/
/*                                  MACROS                                   */
/*****************************************************************************/
#define SUCCESS 1
#define FAILURE -1

#define MAXBUF 99999
#define BUF_SIZE 256 //Size of the buffer


#define GINT_TO_POINTER(i) ((gpointer) (glong) (i))
#define GPOINTER_TO_INT(p) ((gint)  (glong) (p))

/*****************************************************************************/
/*                            GLOBAL VARIABLES                               */
/*****************************************************************************/
FILE *LOG_FILE = NULL;                      //File to store the log information
char msg_info[256];                         //Buffer to store message for the log file
fd_set direcs_fds;                          //Set of socket descriptor for select for directories
fd_set replicas_fd;                         //Set of socket descriptor for select for replicas
fd_set crash_direc_fds;                     //To store all the crash directories that exist in the
system
fd_set crash_repli_fds;                     //To store all the crash replicas that exist in the
system
int max_direc_fd=-1;                        //To store the max directory socket descriptor for the
SELECT
int max_replica_fd=-1;                      //To store the max replica socket descriptor for the
SELECT

long message_id;                             //Store the message counter

int    *direcSocks;                         //Store all the sockets for directories
int    *replicaSocks;                       //Store all the sockets for replicas
int    *filemanagerSocks;                   //Store all the sockets for filemanagers
struct sockaddr_in *direc_sockaddr;         //Store all the ips and ports for directories
struct sockaddr_in *replica_sockaddr;       //Store all the ips and ports for Replicas
struct sockaddr_in *filemanager_sockaddr;   //Store all the ips and ports for filemanagers
struct serinfo *direcNodes;                 //Store IP and port for each directory server
struct serinfo *replicaNodes;               //Store IP and port for each replica server
struct serinfo *filemanagerNodes;           //Store IP and port for each filemanager
int    MAX_DIRECTORIES=0;                   //Store the number of directories that exist in the
system
int    MAX_REPLICAS=0;                      //Store the number of replicas that exist in the system
int    MAX_FILEMANAGERS;                    //Store the number of filemanagers that exist in the
system
int    *repliVals;

char   permission[40];                      //Store the permission for each client

long   clientID=0;                          //Store clientID

double    failrate=0.1;                      //store the rate for the fail
```

```
GHashTable * hashFiletags;                  //Metadata table that holds all tag for the data


/* Shuffle the replica set in order to have
 * uniform distribution. */
void shuffle(int *array, size_t n)
{
    srand(time(NULL));

    if (n > 1)
    {
        size_t i;
        for (i = 0; i < n - 1; i++)
        {
            size_t j = i + rand() / (RAND_MAX / (n - i) + 1);
            int t = array[j];
            array[j] = array[i];
            array[i] = t;
        }
    }
}

int isMaxTag(struct TAG *tag1 , struct TAG tag2 )
{

    if ((tag1->num < tag2.num) || (tag1->num == tag2.num  && tag1->clientID <  tag2.clientID)  )
    {
        return 1;
    }

    return 0;
}

char *checksum_get(char *filename)
{

    int MAX_SIZE=900000;

    GChecksum   *cs;
    guchar       data[MAX_SIZE];
    gsize        size = 0;
    FILE        *input;

    const char *sum;

    cs = g_checksum_new( G_CHECKSUM_MD5 );
    input = fopen( filename, "rb" );
    do
    {
        size = fread( (void *)data, sizeof( guchar ), MAX_SIZE, input );
        g_checksum_update( cs, data, size );
    }
    while( size == MAX_SIZE );
    fclose( input );

    sum = g_checksum_get_string( cs );
    // g_checksum_free( cs );

    return (char*)sum;

}

/*****************************************************************************/
/*                              decode message                               */
/*****************************************************************************/
GSList* decode(struct message *msg , char *buf)
{
    //Use comma delimiter to find the type of
    //send message and retrieve the apropriate
    //fields//
    msg->type=strdup(strtok(buf, ","));

    //Check if the type of the message is RREAD-OK//
    if( (strcmp(msg->type , "RREAD-OK" )== 0) )
    {
        int i=0 , len= 0 , repliValue;
        gpointer val;

        msg->tag.num = atoi(strtok(NULL, ","));
```

```
    msg->tag.clientID = atoi(strtok(NULL, ","));
    len = atoi(strtok(NULL, ","));
    //Read all replica that hold the data
    for(i=0; i<len; i++)
    {
       repliValue = atoi(strtok(NULL, ","));
       val=GINT_TO_POINTER(repliValue);
       msg->replicaSet=g_slist_prepend(msg->replicaSet , val );
    }
    msg->msg_id = atoi(strtok(NULL, ","));
    msg->fileID = atoi(strtok(NULL, ","));
    return msg->replicaSet;
}

else if((strcmp(msg->type , "WREAD-OK" )== 0))
{

    int i=0 , len= 0 , repliValue;
    gpointer val;

    msg->tag.num = atoi(strtok(NULL, ","));
    msg->tag.clientID = atoi(strtok(NULL, ","));
    len = atoi(strtok(NULL, ","));
    //Read all replica that hold the data
    for(i=0; i<len; i++)
    {
        repliValue = atoi(strtok(NULL, ","));
        val=GINT_TO_POINTER(repliValue);
        msg->replicaSet=g_slist_prepend(msg->replicaSet , val );
    }
    msg->msg_id = atoi(strtok(NULL, ","));
    msg->fileID = atoi(strtok(NULL, ","));

    return msg->replicaSet;
}

//Check if the type of the message is RWRITE
else if((strcmp(msg->type , "RWRITE-OK" )== 0))
{
    //msg->tag.num =atoi(strtok(NULL, ","));
   // msg->tag.id = atoi(strtok(NULL, ","));
    msg->msg_id = atoi(strtok(NULL, ","));
    //msg->fileID = atoi(strtok(NULL, ","));
}

else if((strcmp(msg->type , "WWRITE-OK" )== 0))
{
    msg->msg_id = atoi(strtok(NULL, ","));
}
else if((strcmp(msg->type , "READ-OK" )== 0))
{
    msg->tag.num = atoi(strtok(NULL, ","));
    msg->tag.clientID = atoi(strtok(NULL, ","));
    msg->msg_id = atoi(strtok(NULL, ","));
    msg->fileSize = atoi(strtok(NULL, ","));
    msg->checksum = strdup(strtok(NULL, ","));
}
else if((strcmp(msg->type , "WRITE-OK" )== 0))
{
    msg->msg_id = atoi(strtok(NULL, ","));
    msg->filename = strdup(strtok(NULL, ","));
}
else if((strcmp(msg->type , "REQCLIENTID" )== 0))
{
    msg->clientID = atol(strtok(NULL, ","));
    msg->msg_id =  atol(strtok(NULL, ",")) ;
}
else if((strcmp(msg->type , "REQCREATE" )== 0))
{
    msg->fileID = atol(strtok(NULL, ","));
    msg->msg_id =  atol(strtok(NULL, ","));
}
else if((strcmp(msg->type , "REQFILEID" )== 0))
{
    msg->fileID = atol(strtok(NULL, ","));
    msg->msg_id =  atol(strtok(NULL, ","));
}
else if((strcmp(msg->type , "ACCESSDENIED" )== 0))
{
    msg->msg_id =  atol(strtok(NULL, ","));
```

```
        msg->fileID = atol(strtok(NULL, ","));
    }

    return NULL;

}//Function decode message

/*****************************************************************************/
/*                              encode message                               */
/*****************************************************************************/
void encode(struct message *msg , char *buf , char *type)
{

    bzero(buf,sizeof(buf));

    //Handle the case where file does not have an extension type
    char filename[256];
    //Initialize filename in order to know what data exist inside the variable
    bzero(filename,sizeof(filename));
    //Check if the file has type extension or not.
    if(strcmp(msg->filetype,"")==0)
    {
        sprintf(filename,"%s",msg->filename);
    }
    else
    {
        sprintf(filename,"%s.%s",msg->filename,msg->filetype);
    }

    //Check if the message is secure
    if((strcmp(type , "SECURE" )== 0) )
    {
        sprintf(buf,"%s,%ld,%ld,%ld,%ld,%s" ,type , msg->tag.num , msg->tag.clientID, message_id, msg-
>fileID , filename );
    }

    //Check if the type of the message is RREAD
    if((strcmp(type , "RREAD" )== 0) )
    {
        sprintf(buf,"%s,%ld,%s,%ld,%s" ,type , message_id , msg->filename , msg->fileID , permission);
    }
    //Check if the type of the message is RWRITE
    else if( (strcmp(type , "RWRITE" )== 0 ) || (strcmp(type , "WWRITE" )== 0 ) )
        {
            int i=0 , len , repliValue ;
            len=g_slist_length(msg->replicaSet);
            //Read all replica that hold the data
            if( len != 0 )
            {
                //To point on replica set & go through all the list
                GSList* iterator=NULL;
                //To serialize replicaSet to string
                char strSet[256];
                char str[40];

                //Initialize str
                bzero(strSet,sizeof(strSet));

                for (iterator = msg->replicaSet; iterator; iterator = iterator->next)
                {
                    bzero(str,sizeof(str));
                    sprintf(str,",%d" , GPOINTER_TO_INT(iterator->data) );
                    strcat(strSet,str);
                }
                sprintf(buf, "%s,%d%s,%ld,%ld,%ld,%s", type ,len, strSet ,msg->tag.num, msg-
>tag.clientID, message_id , filename);

            }
            else
            {
                sprintf(buf, "%s,0,%ld,%ld,%ld,%s", type , msg->tag.num, msg->tag.clientID, message_id
, filename);
            }

        }//If RWRITE statment

    else if((strcmp(type , "WWRITE" )== 0 ))
    {
        int i=0 , len , repliValue ;
        len=g_slist_length(msg->replicaSet);
```

```
        //Read all replica that hold the data
        if( len != 0 )
        {
            //To point on replica set & go through all the list
            GSList* iterator=NULL;
            //To serialize replicaSet to string
            char strSet[256];
            char str[40];

            //Initialize str
            bzero(strSet,sizeof(strSet));

            for (iterator = msg->replicaSet; iterator; iterator = iterator->next)
            {
                bzero(str,sizeof(str));
                sprintf(str,",%d" , GPOINTER_TO_INT(iterator->data));
                strcat(strSet,str);
            }
            sprintf(buf, "%s,%d%s,%ld,%ld,%ld,%s", type ,len, strSet ,msg->tag.num, msg->tag.clientID,
message_id , filename );

        }
        else
        {
            sprintf(buf, "%s,0,%ld,%ld,%ld,%s", type , msg->tag.num, msg->tag.clientID, message_id ,
filename );
        }
    }

    else if( (strcmp(type , "WREAD" )== 0))
    {
        sprintf(buf,"%s,%ld,%ld,%s,%s" ,type , message_id , msg->fileID , filename , permission  ) ;
    }


}//Function decode message

/*****************************************************************************/
/*                              RECEIVE MAJORITY                             */
/*****************************************************************************/
//The isCheck parameter is a set with domain {0,1,2}
//In case where you are interest to know the replica you have to
//specify the variable is check as 1.If your are interested only
//for tag you have to specify just only 2
GSList* receive_quorum(struct TAG *maxTag, GSList *replicaSet,int ischeck , int *ismajor)
{
    int bytes=0;            //To validate the size of the received msg.
    struct timeval tv;      //To set a time out for the select
    int i;                  //For loop statment
    char buffer[512];
    int readsocks=0;
    int majority=0;         //Count the acks from the directories.
    int IsComplete=0;
    (*ismajor) = -1;

    //To store the message that receive from each server//
    struct message *msg;

    //Allocation memory
    msg=(struct message *)malloc(sizeof(struct message));

    //initialization
    msg->replicaSet = NULL;

    /*Initialize tag*/
    //msg.tag.num=0;
    //msg.tag.id=0;

    //Sleep x microsecond
    //usleep(10000);

    //set the end time to the current time plus 5 seconds
    time_t endTime = time(NULL) + 5;

    usleep(5000);
    while(time(NULL) < endTime)
    {
        // if(result < 0){exit(-1);}
        for(i=0; i <MAX_DIRECTORIES; i++)
        {
```

```
            bzero(buffer, sizeof(buffer));
            if ((bytes = recv(direcSocks[i], buffer, sizeof(buffer) , MSG_DONTWAIT)) < 0)
            {
                //perror("Failed client received the message");
                continue;
            }
            if(strlen(buffer) != 0 )
            {
                //printf("MessageDirecMajor: %s\n", buffer);

                //Decode the message that you received from server//
                msg->replicaSet=decode(msg, buffer);


                //Check if the client have the permission to read the file, otherwise return
                //access denied to the client
                if(strcmp( msg->type,"ACCESSDENIED") == 0)
                {
                    //In case where the client does not have permission to read the file
                    (*ismajor) = -2;

                    return NULL;
                }

                //Check if you received the message that you wait..otherwise drop the msg//
                if (msg->msg_id == message_id)
                {
                    ++IsComplete;
                    /*In case you wait the tag&value with the response message
                     * otherwise it's not nessesary to wait something*/
                    if (ischeck ==1)
                    {
                       if( isMaxTag(maxTag , msg->tag) == 1)
                       {
                            (maxTag->num) = msg->tag.num;
                            (maxTag->clientID) = msg->tag.clientID;
                            GSList* iterator = NULL;

                            //Store all replica that hold the data
                            for (iterator = msg->replicaSet; iterator; iterator = iterator->next)
                            {
                                //Check if the replica exist in replicaSet if does not exist
                                //insert it
                                if(g_slist_find(replicaSet,iterator->data) == NULL)
                                {
                                    replicaSet=g_slist_append(replicaSet, iterator->data);
                                }
                            }
                       }
                    }
                    else if(ischeck==2)
                    {
                        if( isMaxTag(maxTag , msg->tag) == 1)
                        {
                            (maxTag->num) = msg->tag.num;
                            (maxTag->clientID) = msg->tag.clientID;
                        }
                    }
                    /*condition ? valueIfTrue : valueIfFalse*/
                }
                /*  #ifdef DEVMODE
                  else
                  {
                      printf("Old message received, msg_id:%d",msg.msg_id);
                  }
                  #endif

                  printf("Before going to is_major:%d\n" , is_major );
                 */
            }//For statment
        }//For statment

        if(IsComplete  >=  ( floor((MAX_DIRECTORIES/2)) +1))
        {
            break;
        }
        usleep(500000);
    }

    if(IsComplete >= (floor(MAX_DIRECTORIES/2)+1))
```

```
    {
      //  printf("Received Directory Quorum\n");
        (*ismajor) = 1;
        return replicaSet;
    }

    return NULL;
}//Function recv_majority

GSList* recvrepliquorum(struct message *msg , GSList *replicaSet)
{
    int bytes=0;            //To validate the size of the received msg.
    struct timeval tv;      //To set a time out for the select
    int i;                  //For loop statment
    char buffer[512];
    int readsocks=0;
    int majority=0;         //Count the acks from the directories.
    int IsComplete=0;
    int fd=0;


    // set the end time to the current time plus 5 seconds
    time_t endTime = time(NULL) + 5;

    /*To store the message that receive from each server*/
    //struct message *msg;
    //Allocation memory
    //msg=(struct message *)malloc(sizeof(struct message));

    usleep(5000);
    while(time(NULL) < endTime)
    {
        for(i=0; i <MAX_REPLICAS; i++)
        {
               bzero(buffer, sizeof(buffer));
               if ((bytes = recv(replicaSocks[i], buffer, sizeof(buffer), MSG_DONTWAIT)) < 0)
               {
                   // perror("Failed client received the message");
                    continue;
               }
               if(strlen(buffer) != 0 )
               {
                   // printf("ReplicaQuorumReceived:%s\n",buffer);

                    //Decode the message that you received from server*/
                    decode(msg, buffer);

                    //Check if you received the message that you wait..otherwise drop the msg*/
                    if (msg->msg_id == message_id)
                    {
                        //Count how many acknowledge received so far
                        ++IsComplete;

                        //Store all replica that hold the data
                        replicaSet=g_slist_append(replicaSet, GINT_TO_POINTER(i));
                    }
               }//For
        }//For
        //Check if you receive quorum of replicas
        if(IsComplete >=  ( ceil((MAX_REPLICAS *failrate) ) ))
        {
            break;
        }
        //sleep x second for avoid congestion in cpu utilization
        usleep(500000);
    }

    if( IsComplete >= (ceil( ( MAX_REPLICAS * failrate ) ) ) )
    {
      //printf("Received Replica Quorum\n");
        return replicaSet;
    }

    return NULL;
}

GHashTable *storefiletag(char *filename, struct TAG *filetag )
{
    //Create a new tag to be insert in list
    struct TAG* newfiletag = g_new(struct TAG,1);
```

```
    newfiletag->num = filetag->num ;
    newfiletag->clientID = filetag->clientID ;

    //Point to the tag for the specific filename
    struct TAG *pointTag=NULL;

    //Retrieve all the tags which are associated withe the specific filename
    pointTag =(struct TAG*) g_hash_table_lookup(hashFiletags , filename);

    if(pointTag != NULL)
    {
        //Delete the old one tag and replace it with the new one
        free(pointTag);
    }

    //Insert updated list in hashtable to the associated key(filename)
    g_hash_table_insert(hashFiletags , g_strdup( filename ) , newfiletag);

    return hashFiletags;
}

/*****************************************************************************/
/*                            READ_QUERY FUNCTION                            */
/*****************************************************************************/
GSList  *read_Query(struct cmd *cmdmsgIn  ,  struct TAG *tag , struct message *msg )
{
    int IsSuccess=1;

    char buf[512];
    int i;

    int isReceiveMajor;

    //The replicas that hold the file
    GSList  *setOfReplica=NULL;

    //delete
    tag->num=0;
    tag->clientID=0;

    //Retrieve information about the file
    msg->fileID= cmdmsgIn->fileid;
    msg->filename=strdup(cmdmsgIn->filename);
    msg->filetype=strdup(cmdmsgIn->fileType);

    //Initialize buffer
    bzero(buf, sizeof(buf));
    encode(msg, buf , "RREAD");

    //Broadcast to all directories to request the data with the
    //specific ID
    for(i=0; i < MAX_DIRECTORIES; i++)
    {
        if (send(direcSocks[i], buf, sizeof(buf), 0) < 0)
        {
            perror("send() failed\n");
            //exit(EXIT_FAILURE);
        }
    }//For statment

    //Wait until to receive a quorum the last tag with a set of replica that
    //hold the data
    setOfReplica=receive_quorum(tag , setOfReplica , 1 , &isReceiveMajor );
    if( isReceiveMajor ==  -1 )
    {
        printf("---------------------------------------\n");
        printf("\nUNABLE TO RECEIVE DIRECTORY QUORUM\n");
        printf("---------------------------------------\n");
        return NULL;
    }
    else  if(isReceiveMajor ==  -2)
    {
        printf("\n*************\n");
        printf("ACCESS DENIED\n");
        printf("*************\n");
        return NULL;
    }

    //Return the uptodate replicas that stored the file
    return setOfReplica;
```

```
}

/*****************************************************************************/
/*                            READ_INFORM FUNCTION                           */
/*****************************************************************************/
int read_Inform( struct cmd *cmdmsgIn  ,  struct TAG *tag , struct message *msg , GSList
*setOfReplica  )
{

    //Buffer to store the message that it will sent
    char buf[512];

    //index for the for
    int i;

    //Variable that indicate if it receive majority
    int isReceiveMajor;

    //Variable that indicate if it receive correctly the file
    int IsSuccess=1;

    //Second phase of the Directory
    // Announce the maxtag & value that you found in order
    // guaranties strong consistent //
    msg->tag.num=tag->num;
    msg->tag.clientID=tag->clientID;
    msg->replicaSet = setOfReplica;
    msg->msg_id= (++message_id);

    bzero(buf, sizeof(buf));
    encode(msg , buf , "RWRITE" );
    for(i=0; i < MAX_DIRECTORIES; i++)
    {
        if (send(direcSocks[i], buf, sizeof(buf), 0) < 0)
        {
            perror("send() failed\n");
        }
    }//For statment

    receive_quorum(tag,setOfReplica,0 , &isReceiveMajor);
    if( (isReceiveMajor) != 1)
    {
        printf("-------------------------------------------------------\n");
        printf("UNABLE TO READ QUORUM IN THE SECOND PHASE OF READER\n");
        printf("-------------------------------------------------------\n");
    }

    //Store file tag in the hashtable
    storefiletag(msg->filename , tag);

    printf("-------------------------------------------------------------------------------------\n");
    printf("Read operation completed: Value: , tagNum:%ld , tagID:%ld\n",tag->num,tag->clientID );
    printf("-------------------------------------------------------------------------------------\n");


    //Go through the list that holds the data.
    //In case when one replica is fault you can
    //request the data from the next available replica
    GSList   *iter=NULL;


    printf("\n-------------------------------------------------\n");
    printf("Replica Set\n");
    printf("{ ");
    for (iter = setOfReplica; iter; iter = iter->next)
    {
        if(iter->next !=NULL)
        {
            printf("%d," , GPOINTER_TO_INT(iter->data));
        }
        else
        {
            printf("%d }" , GPOINTER_TO_INT(iter->data));
        }

    }
    printf("\n-------------------------------------------------\n");


    //Go through available replicas
```

```
    for (iter = setOfReplica; iter; iter = iter->next)
    {
        IsSuccess=get_file(replicaSocks[(GPOINTER_TO_INT(iter->data))] , msg );
        //Check if the file have transfered  successful
        if(IsSuccess == SUCCESS )
        {
            break;
        }
    }

    //deallocate
    free(msg);
    free(tag);

    return IsSuccess;

}

/*****************************************************************************/
/*                              READER FUNCTION                              */
/*****************************************************************************/
int reader_oper(int msg_id , struct cmd *cmdfile )
{
   //Store if the action are
   int isSuccess=FAILURE;

   //Allocation memory
   struct message *msg=(struct message *)malloc(sizeof(struct message));
   struct TAG *tag=(struct TAG *)malloc(sizeof(struct TAG));

   //Store the file that hold the UpToDate replicas
   GSList  *setOfReplica=NULL;

   //First phase of ABD algorithm
   setOfReplica = read_Query(cmdfile , tag , msg);

   //Check if it receive a replica set
   if(setOfReplica == NULL)
   {
       return FAILURE;
   }

   //Retrive the file from the replica
   isSuccess = read_Inform(cmdfile, tag , msg , setOfReplica);

   return isSuccess;

}//Function reader

/*****************************************************************************/
/*                            WRITER FUNCTION                                */
/*****************************************************************************/
int writer_oper(int msg_id , struct cmd *cmdfile  )
{

    int isReceiveMajor;

    char buf[512];
    int i;
    //The replicas that hold the file
    struct message *msg;
    struct TAG *tag;

    //The replicas that hold the file
    GSList  *setOfReplica=NULL;

    //Allocation memory
    msg=(struct message *)malloc(sizeof(struct message));
    tag=(struct TAG *)malloc(sizeof(struct TAG));

    //delete
    tag->num=0;
    tag->clientID=0;

    //Retrieve information about the file
    msg->fileID = cmdfile->fileid;
    msg->filename=strdup(cmdfile->filename);
    msg->filetype=strdup(cmdfile->fileType);

    //Initialize buffer
```

```
    bzero(buf, sizeof(buf));
    encode(msg, buf , "WREAD");

    //Broadcast to all directories to request the tag for a
    //specific file ID
    for(i=0; i < MAX_DIRECTORIES; i++)
    {
        if (send(direcSocks[i], buf, sizeof(buf), 0) < 0)
        {
            perror("send() failed\n");
            //exit(EXIT_FAILURE);
        }
    }//For statment

    //Wait until to receive a quorum with the last tag
    setOfReplica = receive_quorum(tag,setOfReplica,1 , &isReceiveMajor);

    //Check if you received the latest tag
    if( isReceiveMajor == -1)
    {
        printf("-----------------------------------------------\n");
        printf("UNABLE TO RECEIVE QUORUM OF DIRECTORY IN WRITER\n");
        printf("-----------------------------------------------\n");
        return FAILURE;
    }
    else  if(isReceiveMajor ==  -2)
    {
        printf("\n**************\n");
        printf("ACCESS DENIED\n");
        printf("**************\n");
        return FAILURE;
    }

    printf("-------------------------------------------------------------------------------------------------
-------------\n");
    printf("READ TAG FROM DIRECTORIES OPERATION COMPLETED , TAG_NUM:%ld , TAG_CLIENTID:%ld\n",tag->num
, tag->clientID );
    printf("-------------------------------------------------------------------------------------------------
-------------\n");

    //Retrieve the tag for the file
    struct TAG *filetag = NULL;
    filetag = g_hash_table_lookup(hashFiletags , msg->filename );
    if(filetag == NULL )
    {
        filetag =(struct TAG*) malloc(sizeof(tag));
        filetag->num = 2 ;
        filetag->clientID = clientID;
    }

    if(filetag->num < tag->num || (filetag->num == tag->num && filetag->clientID < tag->clientID))
    {
        printf("---------------------\n");
        printf("FOUND CONFLICT ON TAG\n");
        printf("---------------------\n");

        char tmpfilename[256];
        bzero(tmpfilename,sizeof(tmpfilename));

        //Replace the old one version of the file with the new one
        unsigned int length=sprintf(tmpfilename,"%s.%s", msg->filename , msg->filetype);
        //tmpfilename[length-1]='\0';

        //Remove the old file
        //remove(tmpfilename);

        //Read the new version of the file
        read_Inform(cmdfile , tag , msg , setOfReplica);

       //reader_oper(msg_id , cmdmsgIn);

        return FAILURE;
    }

    //Increase the msgID before to broadcast the message to all replicas
    ++message_id;
    tag->num +=1;
    tag->clientID = clientID;

    //Clear the old replica set
```

```
    g_slist_free(setOfReplica);
    setOfReplica= NULL;

    shuffle(repliVals,MAX_REPLICAS);

    //Broadcast to all replicas the object and wait from f(fail number) replicas to response
    //with an acknowledgment
    for(i=0; i < ceil(( MAX_REPLICAS * failrate )); i++)
    {
        int repl= repliVals[i];

        if( send2ftp(cmdfile,replicaSocks[repl] , tag ,(message_id) ) == FAILURE)
        {
            printf("Error:Send2ftp\n");
            return  FAILURE;
        }
    }


    setOfReplica = recvrepliquorum(msg,setOfReplica);
    if(setOfReplica  == NULL)
    {
        printf("--------------------------------------\n");
        printf("UNABLE TO RECEIVE RATEFAIL OF REPLICAS\n");
        printf("--------------------------------------\n");
        return FAILURE;
    }

    printf("-----------------------------------------------------------------------------------------------
------\n");
    printf("WRITE TO REPLICAS OPERATION COMPLETED , TAG_NUM:%ld , TAG_CLIENTID:%ld\n",tag->num , tag-
>clientID );
    printf("-----------------------------------------------------------------------------------------------
------\n");

    //Debug Purpose
    GSList *iter=NULL;

    printf("-------------------------------------------\n");
    printf("Replica Set\n");
    printf("{ ");
    for (iter = setOfReplica; iter; iter = iter->next)
    {
        if(iter->next !=NULL)
        {
            printf("%d," , GPOINTER_TO_INT(iter->data));
        }
        else
        {
            printf("%d }" , GPOINTER_TO_INT(iter->data));
        }

    }
    printf("\n-------------------------------------------\n");


    //Announce to all directories that f replicas
    //had store the object
    (++message_id);
    msg->replicaSet = setOfReplica;
    msg->msg_id= message_id ;
    msg->tag.num =tag->num;
    msg->tag.clientID = tag->clientID;

    bzero(buf, sizeof(buf));
    encode(msg , buf , "WWRITE" );

    //Inform all directories which replicas hold the data
    for(i=0; i < MAX_DIRECTORIES; i++)
    {
        if (send(direcSocks[i], buf, sizeof(buf), 0) < 0)
        {
            perror("send() failed\n");
        }
    }//For statment

    //Phase wdw.Wait a quorum Directories
    receive_quorum(tag , setOfReplica , 0 , &isReceiveMajor);
    if( isReceiveMajor != 1)
    {
```

```
        printf("wdw..Unable to receive a majority\n");
    }

    //Update the hashtable with the new tag for the file
    storefiletag(msg->filename , tag );

    printf("-------------------------------------------------------------------------------------------------
----------\n");
    printf("WRITE TO DIRECTORIES OPERATION COMPLETED , LAST_TAG_NUM:%ld , LAST_TAG_IS:%ld\n",tag->num
, tag->clientID);
    printf("-------------------------------------------------------------------------------------------------
----------\n");

    //delete
    //sleep(2);

    //Initialize buffer
    bzero(buf, sizeof(buf));

    //Encode the message that it will send
    encode(msg , buf , "SECURE" );

    //GSList *iter=NULL;

    for (iter = setOfReplica; iter; iter = iter->next)
    {
        if (send(replicaSocks[(GPOINTER_TO_INT(iter->data))], buf , sizeof(buf) , 0) < 0)
        {
            perror("send() secure failed\n");
        }
    }


    printf("-------------------------------------------------------------------------------------------------
----\n");
    printf("SEND SECURE MESSAGE TO REPLICAS , LAST_TAG_NUM:%ld , LAST_TAG_IS:%ld\n",tag->num , tag-
>clientID );
    printf("-------------------------------------------------------------------------------------------------
----\n");


    //deallocate
    free(msg->filename);
    free(msg->filetype);
    g_slist_free(msg->replicaSet);
   // g_slist_free(setOfReplica);
    free(msg);
    free(tag);

}//Function writer



/*****************************************************************************/
/*                            READ COMMAND                                   */
/*****************************************************************************/
int read_cmd(char *cmd_str , struct cmd *cmdmsg )
{
    char *cmd;
    char *temp;
    /* get the first token */
    cmd = strtok(cmd_str, " ");
    strcpy(cmdmsg->oper,cmd);

    //Check the command if it exist
    if(strcmp(cmd,"exit") ==0)
    {

        int i=0;

        //Fill the direcSocks for directories server which is type sockaddr_in
        for (i = 0; i < MAX_DIRECTORIES; i++)
        {
           close(replicaSocks[i]);
        }//For statement

        //Fill the replicaSocks for directories server which is type sockaddr_in
        for (i = 0; i < MAX_REPLICAS; i++)
        {
           close(direcSocks[i]);
```

```
    }//For statement


    //Fill the replicaSocks for directories server which is type sockaddr_in
    for (i = 0; i < MAX_FILEMANAGERS; i++)
    {
        close(filemanagerSocks[i]);
    }//For statement

    return 1;
}
else if(strcmp(cmd , "loggin" ) == 0)
{
    //Store the username of the client
    char *username;

    username = strdup(strtok(NULL," "));

    //Request clientID
    reqClientID(username);

    //Deallocate
    free(username);

    return 1;
}
else if(strcmp(cmd , "create" ) == 0)
{
    //Store the filename
    char *filename;

    filename = strdup(strdup(strtok(NULL," ")));

    long fileid = reqCreate(filename);

    printf("Create executed! FileID: %ld" , fileid);

    //Deallocate
    free(filename);

    return 1;
}

else if(strcmp(cmd , "list" ) == 0)
{

    get_filelist();

    return 1;

}

//Retieve information of the file
temp=strtok(NULL," ");
if(strchr(temp, '.') != NULL)
{
    cmdmsg->filename=strdup(strtok(temp,"."));
    cmdmsg->fileType=strdup(strtok(NULL," "));
}
else
{
    cmdmsg->filename=strdup(temp);
    cmdmsg->fileType="";
}

if(strcmp(cmd , "read" ) == 0 )
{
    cmdmsg->fileid = reqFileID(cmdmsg , clientID);

    //In case where it receive correct the fileid
    if(cmdmsg->fileid != FAILURE)
    {
        //get_file(replicaSocks[1] , msg);
        if( reader_oper(message_id , cmdmsg ) == FAILURE )
        {
            printf("----------------------------");
            printf("\nUNABLE TO READ THE FILE\n");
            printf("----------------------------");
        }
    }
```

```
        else
        {
            printf("--------------------------------------");
            printf("\nUNABLE TO READ THE FILEID CORRECTLY\n");
            printf("--------------------------------------");
        }

    }

    //In case when client interested to write a file to replicasc
    else if(strcmp(cmd , "write" ) == 0)
    {
        cmdmsg->fileid = reqFileID(cmdmsg , clientID);

        //In case where it receive correct the fileid
        if(cmdmsg->fileid != FAILURE)
        {

            writer_oper(message_id, cmdmsg);
        }
        else
        {
            printf("Unable to receive the fileID correct\n");
        }
    }

    //deallocate
    free(cmdmsg->filename);
    //free(cmdmsg->fileType);

    return 1;
}

/*****************************************************************************/
/*                              Read config                                  */
/*****************************************************************************/
void readConfig(char *filename)
{

    FILE * fp;
    char * line = NULL;
    size_t len = 0;
    ssize_t read;

    int i=0,j=0,d=0;
    fp = fopen(filename, "r");
    if (fp == NULL)
    {
        printf("Unable to open config fiLe\n");
        exit(EXIT_FAILURE);
    }

    while ((read = getline(&line, &len, fp)) != -1)
    {

        if(strcmp(line,"#DIRECTORIES\n") == 0)
        {
            while ((read = getline(&line, &len, fp)) != -1)
            {
                if(strcmp(line,"#REPLICAS\n") == 0)
                {
                    break;
                }
                if(MAX_DIRECTORIES==0)
                {
                    MAX_DIRECTORIES=atoi(line);
                    direcNodes=(struct serinfo *)malloc(sizeof(struct serinfo ) * MAX_DIRECTORIES );
                }
                else
                {
                    strcpy(direcNodes[i].ip_addr, strtok(line, " ") );
                    direcNodes[i].port = atoi(strtok(NULL, " "));
                    i +=1;
                }
            }//While
        }

        if(strcmp(line,"#REPLICAS\n") == 0)
        {
```

```
            while ((read = getline(&line, &len, fp)) != -1)
            {
                if(strcmp(line,"#FILEMANAGERS\n") == 0)
                {
                    break;
                }
                if (MAX_REPLICAS == 0)
                {
                    MAX_REPLICAS = atoi(line);
                    replicaNodes=(struct serinfo *)malloc(sizeof(struct serinfo ) * MAX_REPLICAS );
                }
                else
                {
                    strcpy(replicaNodes[j].ip_addr, strtok(line, " "));
                    replicaNodes[j].port = atoi(strtok(NULL, ","));
                    j += 1;
                }
            }//While
        }
        if(strcmp(line,"#FILEMANAGERS\n") == 0)
        {
            while ((read = getline(&line, &len, fp)) != -1)
            {
                if(strcmp(line,"#PERMISSION\n") == 0)
                {
                    break;
                }
                if (MAX_FILEMANAGERS == 0)
                {
                    MAX_FILEMANAGERS = atoi(line);
                    filemanagerNodes=(struct serinfo *)malloc(sizeof(struct serinfo ) *
MAX_FILEMANAGERS );
                }
                else
                {
                    strcpy(filemanagerNodes[d].ip_addr , strtok(line, " ") );
                    filemanagerNodes[d].port = atoi(strtok(NULL, " "));
                    d += 1;
                }
            }//While
        }
        if(strcmp(line,"#PERMISSION\n") == 0)
        {
            read = getline(&line, &len, fp);

            //Store the permission for the client
            strcpy(permission, line);
        }
    }

    fclose(fp);
    if (line)
        free(line);
}

int reqClientID(char *username)
{
    //Buffer message
    char buf[256];

    //Initialize the buffer
    bzero(buf,sizeof(buf));

    //Store the bytes that sent
    int bytes;

    //Message ID to validate that it receive the correct message
    long messageID = (++message_id);

    //Allocation memory
    struct message *msg=(struct message *)malloc(sizeof(struct message));

    //Request from the filemanager to set up a client ID
    sprintf(buf,"REQCLIENTID,%s,%ld",username , messageID );

    if (bytes = send(filemanagerSocks[0] , buf, sizeof(buf) , 0) < 0)
    {
        perror("Send:Unable to request clientID");
    }
```

```
    //Initialize the buffer
    bzero(buf,sizeof(buf));

    //Wait to receive the message
    if (recv(filemanagerSocks[0], buf, sizeof(buf), 0) < 0)
    {
        perror("Received() Unable to receive clientID");
    }

    //Decode the message that it receive
    decode(msg,buf);

    //Deallocate type
    free(msg->type);

    //Check if it received the correct message
    if(msg->msg_id == messageID)
    {
        //Deallocate struct
        free(msg);

        //Assign client ID that it received
        clientID = msg->clientID;
    }
    else
    {
        //Deallocate struct
        free(msg);

        return FAILURE;
    }
}

long reqCreate(char *filename)
{
    //Buffer message
    char buf[256];

    //Message ID to validate that it receive the correct message
    long messageID = (++message_id);

    //Store the number of bytes that sent via socket
    int bytes;

    //Store the fileid for the file that it create
    long fileid;

    //Request from the filemanager to set up a client ID
    sprintf(buf,"REQCREATE,%s,%ld,%ld" , filename , clientID, messageID );

    if (bytes = send(filemanagerSocks[0] , buf, sizeof(buf) , 0) < 0)
    {
        perror("Send:Unable to request clientID");
    }

    bzero(buf,sizeof(buf));
    if (recv(filemanagerSocks[0], buf, sizeof(buf) , 0) < 0)
    {
        perror("Received() Unable to receive clientID");
    }

    printf("ReqCreate Received: %s\n" , buf );

    //Allocation memory
    struct message *msg=(struct message *)malloc(sizeof(struct message));

    //Decode the message that it receive
    decode(msg,buf);

    //Deallocate type
    free(msg->type);

    //Check if it received the correct message
    if(msg->msg_id == messageID)
    {
        //Assign fileid
        int fileid = msg->fileID;

        //Deallocate struct
        free(msg);
```

```
        return fileid;
    }

    else
    {
        //Deallocate struct
        free(msg);

        return FAILURE;
    }
}

long reqFileID(struct cmd *cmdmsg , long clientID)
{
    //Buffer message
    char buf[256];

    //Store the number of bytes that sent via socket
    int bytes;

    //Message ID to validate that it receive the correct message
    long messageID = (++message_id);

    bzero(buf,sizeof(buf));


    //Handle the case where file does not have
    //file extension
    if(strcmp(cmdmsg->fileType  ,"")==0)
    {
        //Request from the filemanager to set up a client ID
        sprintf(buf,"REQFILEID,%s,%ld,%ld" , cmdmsg->filename , clientID , messageID );
    }
    else
    {
        //Request from the filemanager to set up a client ID
        sprintf(buf,"REQFILEID,%s.%s,%ld,%ld" , cmdmsg->filename , cmdmsg->fileType , clientID ,
messageID );

    }


    if (bytes = send(filemanagerSocks[0] , buf, sizeof(buf) , 0) < 0)
    {
        perror("Send:Unable to request clientID");
    }

    bzero(buf,sizeof(buf));
    if (recv(filemanagerSocks[0], buf, sizeof(buf), 0) < 0)
    {
        perror("Received() Unable to receive clientID");
    }

    printf("REQFILEID Received: %s\n" , buf );

    //Allocation memory
    struct message *msg=(struct message *)malloc(sizeof(struct message));

    //Decode the message that it receive
    decode(msg,buf);

    //Deallocate type
    free(msg->type);

    //Check if it received the correct message
    if(msg->msg_id == messageID)
    {
        //Assign fileid
        int fileid = msg->fileID;

        //Deallocate struct
        free(msg);

        return fileid;
    }

    else
    {
        //Deallocate struct
```

```
        free(msg);

        return FAILURE;
    }


}

/*****************************************************************************/
/*                            Establish connections                          */
/*****************************************************************************/
int connect2Replicas()
{
    int i=0;

    for(i=0; i < MAX_REPLICAS; i++)
    {
        /*Establish a connection withe the IP address from the struct server*/
        if (connect(replicaSocks[i] , (struct sockaddr *) &replica_sockaddr[i],
sizeof(replica_sockaddr[i])) < 0)
        {
            /*Use htohs to convert network port to host port.*/
            printf("Unable to Connect to : %s ,  PORT: %d\n", inet_ntoa(replica_sockaddr[i].sin_addr),
ntohs(replica_sockaddr[i].sin_port) );
        }
        else
        {
            printf("Connection established to IP: %s ,  PORT: %d\n",
inet_ntoa(replica_sockaddr[i].sin_addr),ntohs(replica_sockaddr[i].sin_port));
        }
    }
}

void conn2direc()
{
    int i=0;
    char buf[50];
    bzero(buf,sizeof(buf));
    int bytes;

    for(i=0; i < MAX_DIRECTORIES; i++)
    {
        /*Establish a connection withe the IP address from the struct server*/
        if (connect(direcSocks[i] , (struct sockaddr *) &direc_sockaddr[i], sizeof(direc_sockaddr[i]))
< 0)
        {
            /*Use htohs to convert network port to host port.*/
            printf("Unable to Connect to : %s ,  PORT: %d\n", inet_ntoa(direc_sockaddr[i].sin_addr),
ntohs(direc_sockaddr[i].sin_port) );
        }
        else
        {

            printf("Connection established to IP: %s ,  PORT: %d\n",
inet_ntoa(direc_sockaddr[i].sin_addr),ntohs(direc_sockaddr[i].sin_port) );

        }
        //Add socket to set
        FD_SET(direcSocks[i] , &direcs_fds);
        if (direcSocks[i] > max_direc_fd)
        {
            max_direc_fd = direcSocks[i];
        }
    }//For statment

}//Function establish connections

int conn2filemanager()
{
    int i=0;

    //Establish Connection with all the filemanager servers
    for(i=0; i < MAX_FILEMANAGERS; i++)
    {
        /*Establish a connection withe the IP address from the struct server*/
        if (connect(filemanagerSocks[i] , (struct sockaddr *) &filemanager_sockaddr[i],
sizeof(filemanager_sockaddr[i])) < 0)
        {
            /*Use htohs to convert network port to host port.*/
```

```
            printf("Unable to Connect to : %s ,  PORT: %d\n",
inet_ntoa(filemanager_sockaddr[i].sin_addr), ntohs(filemanager_sockaddr[i].sin_port) );
        }
        else
        {
            printf("Connection established to IP: %s ,  PORT: %d\n",
inet_ntoa(filemanager_sockaddr[i].sin_addr),ntohs(filemanager_sockaddr[i].sin_port));
        }
    }

}

int get_file(int sock, struct message *msg )
{
    FILE     *received_file;
    int      remain_data;
    int      bytes;
    int      bufsize=2048;
    char     buf[bufsize];
    char     filename[256];
    unsigned    int length;

    //
    struct message *recvmsg;

    //Allocation memory
    recvmsg=(struct message *)malloc(sizeof(struct message));

    sprintf(buf ,"READ,%ld,%ld,%ld,%ld,%s.%s" , msg->tag.num , msg->tag.clientID , (++msg->msg_id) ,
msg->fileID, msg->filename , msg->filetype );

    //printf("Filename in get_file: %s , length: %d\n",buf, file_size);
    if ((bytes=send(sock, buf, 512 , 0)) < 0)
    {
        perror("send() getfile failed\n");
        return FAILURE;
    }

    printf("Bytes sends: %d\n" ,bytes );

    //Wait until to receive an acknowldge with the
    //same messageID
    bzero(buf,sizeof(buf));
    if(recv(sock, buf, 512 , 0) < 0)
    {
        perror("Received");
        exit(sock);
        return FAILURE;
    }

    //Unable to receive details for the file
    if(strlen(buf) ==0)
    {
        return FAILURE;
    }

    printf("Received in getFile: %s\n" , buf);

    decode(recvmsg,buf);

    length=sprintf(filename , "%s.%s" , msg->filename , msg->filetype);
    //filename[length-1]='\0';

    received_file = fopen(filename, "w");
    if (received_file == NULL)
    {
        printf("Failed to open file foo --> %s" , filename);
        return FAILURE;
    }

    //Retrieve the size of the file
    remain_data = recvmsg->fileSize;

    bzero(buf, bufsize);
    while (((bytes = recv(sock, buf, bufsize, 0)) > 0) && (remain_data > 0))
    {
        fwrite(buf, sizeof(char), bytes, received_file);
        remain_data -= bytes;
     // fprintf(stdout, "Receive %d bytes and we hope :- %d bytes\n", bytes, remain_data);
```

```
      //  printf("Received: %d , remain : %d\n",bytes , remain_data);
        if(remain_data == 0 )
        {
            break;
        }

    }//While


    //close file descriptor
    fclose(received_file);


    //Retrieve the checksum from the file that the
    //client receive in order to check if receive
    //the whole file
    char *filechecksum = checksum_get(filename);


    //check if it received the whole file
    //otherwise return an error
    if(remain_data > 0)
    {
        return FAILURE;
    }
    else if(strcmp(filechecksum , recvmsg->checksum) != 0)
    {
        printf("Invalid checksum\n");
        //Remove the file
        remove(filename);

        return FAILURE;
    }


    return SUCCESS;
}

int send2ftp(struct cmd *msgCmd, int newsock , struct TAG *tagIn , long msgIDIn )
{
    int         fd;
    off_t       offset = 0;
    int         remain_data;
    int         sent_bytes;
    struct      stat file_stat; /*to retrieve information for the file*/
    int         len;
    int         file_size;
    char        buf[512];
    char        filename[256];

    bzero(filename,sizeof(filename));

    //Handle the case where file does not have
    //file extension
    if(strcmp(msgCmd->fileType,"")==0)
    {
        unsigned int length=sprintf(filename,"%s", msgCmd->filename);
    }
    else
    {
        unsigned int length=sprintf(filename,"%s.%s", msgCmd->filename,msgCmd->fileType);
    }

    //filename[length-1]='\0';

    fd = open(filename,  O_RDONLY);
    if (fd < 0 )
    {
        printf("Error opening file --> %s" , filename);
        return FAILURE;
    }

    // Get file stats
    if (fstat(fd, &file_stat) < 0)
    {
        printf("Error fstat");
        close(fd);
        return FAILURE;
    }
```

```
    // Sending file size //
    file_size=file_stat.st_size;

    //Check if the file is empty
    if(file_size <= 0)
    {
        printf("UNABLE TO SEND AN EMPTY FILE!!\n");
        return  FAILURE;
    }

    //Retrieve checksum for the file
    char *filechecksum = checksum_get(filename);

    bzero(buf,sizeof(buf));
    sprintf(buf,"WRITE,%ld,%ld,%ld,%ld,%d,%s,%s" , tagIn->num,tagIn->clientID , msgCmd->fileid ,
msgIDIn , file_size, filechecksum,filename );

    // If connection is established then start communicating //
    len = send(newsock, buf, 512 , 0);
    if (len < 0)
    {
        perror("send");
        close(fd);
        return FAILURE;
    }

    //Wait to received acknowledge that received the metadata for the file
    remain_data = file_stat.st_size;
    // Sending file data //
    while (((sent_bytes = sendfile(newsock, fd, &offset, MAXBUF)) > 0) && (remain_data > 0))
    {
        remain_data -= sent_bytes;
    //  printf("Server sent %d bytes from file's data, offset is now : %d and remaining data = %d\n",
sent_bytes, offset, remain_data);

    }

    //Close the file descriptor
    close(fd);

    //check if it received the whole file
    //otherwise return an error
    if(remain_data > 0)
    {
        return FAILURE;
    }
    else
    {
        printf("File: %s send...\n" , msgCmd->filename);

    }

    return SUCCESS;
}

int get_filelist()
{
    //Buffer message
    char buf[99999];

    char tempbuf[256];

    bzero(tempbuf,sizeof(tempbuf));
    bzero(buf,sizeof(buf));

    long msg_id = ++message_id;

    //Request from the filemanager to set up a client ID
    sprintf(tempbuf,"REQFILESLIST,%ld" , msg_id  );

    if (send(filemanagerSocks[0] , tempbuf, strlen(tempbuf) , 0) < 0)
    {
        perror("Send:Unable to request clientID");
    }

    bzero(buf,sizeof(buf));
    if (recv(filemanagerSocks[0], buf, sizeof(buf) , 0) < 0)
    {
        perror("Received() Unable to receive clientID");
    }
```

```
    int i=0;

    //Retrieve the size of the list
    int size = atoi(strtok(buf, ","));

    //Display format
    printf("\n\n-------------------------------------------------\n");
    printf("%-20s%-20s%-20s\n", "Filename","Fileid","owner");
    printf("-------------------------------------------------\n");

    for(i=0; i<size; i++)
    {
        printf("%-20s", strtok(NULL, ",") );
        printf("%-20s", strtok(NULL, ",") );
        printf("%-20s", strtok(NULL, ",") );
        printf("\n");
    }

    printf("-------------------------------------------------\n");

}

void inisialization()
{

    int i;

    //Allocate the array to store all the replica nodes
    repliVals = (int *)malloc(sizeof(int) * MAX_REPLICAS);

    //Allocate the array to store all the sockaddr_in for directories
    direc_sockaddr=(struct sockaddr_in *)malloc(sizeof(struct sockaddr_in ) * MAX_DIRECTORIES );

    //Allocate the array to store all the sockaddr_in for Replicas
    replica_sockaddr=(struct sockaddr_in *)malloc(sizeof(struct sockaddr_in ) * MAX_REPLICAS );

    //Allocate the array to store all the sockaddr_in for file managers
    filemanager_sockaddr=(struct sockaddr_in *)malloc(sizeof(struct sockaddr_in ) * MAX_FILEMANAGERS
);

    //Allocate the array to store all the sockets descriptor for each directory
    direcSocks=( int *)malloc(sizeof(int ) * MAX_DIRECTORIES );

    //Allocate the array to store all the sockets descriptor for each Replicas
    replicaSocks=(int *)malloc(sizeof(int ) * MAX_REPLICAS );

    //Allocate the array to store all the sockets descriptor for each Replicas
    filemanagerSocks=(int *)malloc(sizeof(int ) * MAX_FILEMANAGERS );

    //Fill the direcSocks for directories server which is type sockaddr_in
    for (i = 0; i < MAX_DIRECTORIES; i++)
    {
        direc_sockaddr[i].sin_family = AF_INET; /*Internet domain*/
        direc_sockaddr[i].sin_addr.s_addr = inet_addr(direcNodes[i].ip_addr);
        direc_sockaddr[i].sin_port = htons(direcNodes[i].port);
    }//For statement

    //Fill the replicaSocks for directories server which is type sockaddr_in
    for (i = 0; i < MAX_REPLICAS; i++)
    {
        replica_sockaddr[i].sin_family = AF_INET; /*Internet domain*/
        replica_sockaddr[i].sin_addr.s_addr = inet_addr(replicaNodes[i].ip_addr);
        replica_sockaddr[i].sin_port = htons(replicaNodes[i].port);
    }//For statement


    //Fill the replicaSocks for directories server which is type sockaddr_in
    for (i = 0; i < MAX_FILEMANAGERS; i++)
    {
        filemanager_sockaddr[i].sin_family = AF_INET; /*Internet domain*/
        filemanager_sockaddr[i].sin_addr.s_addr = inet_addr(filemanagerNodes[i].ip_addr);
        filemanager_sockaddr[i].sin_port = htons(filemanagerNodes[i].port);
       // filemanager_sockaddr[i].sin_addr.s_addr = inet_addr("127.0.0.1");
        //filemanager_sockaddr[i].sin_port = htons(40001);
    }//For statement

    //initialize hash table that stores the tags for the files
    hashFiletags = g_hash_table_new( g_str_hash, g_str_equal);
```

```
    //Inisialization  of the set of sockets descriptors
    FD_ZERO(&direcs_fds);
    FD_ZERO(&replicas_fd);
    FD_ZERO(&crash_direc_fds);

    //Go through all the available directories to establish a connection.*/
    for (i = 0; i < MAX_DIRECTORIES; i++)
    {
        //Create a socket and check if is created correct
        if ((direcSocks[i] = socket(PF_INET, SOCK_STREAM, 0)) < 0)
        {
            perror("Socket() failed");
            exit(1);
        }
    }//For statement

    //Fill the replicaSocks for directories server which is type sockaddr_in
    for (i = 0; i < MAX_REPLICAS; i++)
    {
        //Fill with all the replica
        repliVals[i]=i;

        //Create a socket and check if is created correct
        if ((replicaSocks[i] = socket(PF_INET, SOCK_STREAM, 0)) < 0)
        {
            perror("Socket() failed");
            exit(1);
        }
        //Add socket to set
        FD_SET(replicaSocks[i] , &replicas_fd);
        if (replicaSocks[i] > max_replica_fd)
        {
            max_replica_fd = replicaSocks[i];
        }
    }//For statement

    //Fill the replicaSocks for filemanager servers
    for (i = 0; i < MAX_FILEMANAGERS; i++)
    {
        //Create a socket and check if is created correct
        if ((filemanagerSocks[i] = socket(PF_INET, SOCK_STREAM, 0)) < 0)
        {
            perror("Socket() failed");
            exit(1);
        }
    }//For statement


    /*Is an option to set a timeout value for input operations.
      It accepts a struct timeval parameter with the number of seconds
      and microseconds used to limit waits for input operations to complete.*/

    //Configure sockets
    struct timeval timeout;
    timeout.tv_sec = 3;
    timeout.tv_usec = 0;

    //Set the socket to block only for some second
    for (i = 0; i < MAX_REPLICAS; i++)
    {
          setsockopt(replicaSocks[i], SOL_SOCKET, SO_RCVTIMEO, (char *) &timeout , sizeof(timeout)) ;

    }

    //Set the socket to block only for some second
    for (i = 0; i < MAX_FILEMANAGERS; i++)
    {
        setsockopt(filemanagerSocks[i], SOL_SOCKET, SO_RCVTIMEO, (char *) &timeout , sizeof(timeout));
    }


}//Function inisialization

void unitTest(char *filename , char *filetype , char *username)
{
    //Check the system if is working correct
    struct cmd *cmdmsg;

    cmdmsg=(struct cmd*)malloc(sizeof(struct cmd));
    cmdmsg->filename=strdup(filename);
```

```
cmdmsg->fileType=strdup(filetype);

//Unitest for the function reqClientID
long clientID = reqClientID(username);

//Counter to calculate the times that the unitest run
int i=0;

    //newline
    printf("\n");
    if (clientID != FAILURE)
    {
        printf("-----------------------------------\n");
        printf("TEST1 reqClientID function PASSED!!\n");
        printf("Received clientID: %ld\n", clientID);
        printf("-----------------------------------\n");

    }
    else
    {
        printf("-----------------------------------\n");
        printf("TEST1 reqClientID function FAILED!!\n");
        printf("Received clientID: %ld\n", clientID);
        printf("-----------------------------------\n");
    }
    printf("\n--------------------------------------------------------------------------------\n");

for(i=0; i<5; i++)
{
    //Unit test for the function reqCreate
    //Store the filename with the corresponding type
    char newfilename[256];
    bzero(newfilename, sizeof(newfilename));

    sprintf(newfilename, "%s.%s", filename, filetype);

    //Call reqCreate function
    long fileid = reqCreate(newfilename);

    //newline
   /* if (fileid != FAILURE)
    {
        //Store the file id that received
        cmdmsg->fileid = fileid;

        printf("-----------------------------------\n");
        printf("TEST2 reqCreate function PASSED!!\n");
        printf("Received fileID: %ld\n", fileid);
        printf("-----------------------------------\n");

    }
    else
    {
        printf("-----------------------------------\n");
        printf("TEST2 reqCreate function FAILED!!\n");
        printf("Received fileID: %ld\n", fileid);
        printf("-----------------------------------\n");
    }
    printf("\n--------------------------------------------------------------------------------\n");
    */

    //message id
    long messageid=0;

    //Call reqCreate function
    int isSuccess =  writer_oper(messageid,cmdmsg);

    //newline
    if (isSuccess != FAILURE)
    {
        printf("\n*********************************\n");
        printf("TEST3 WRITER_OPER FUNCTION PASSED!!\n");
        printf("*********************************\n");
    }
    else
    {
        printf("*********************************\n");
        printf("TEST3 WRITER_OPER FUNCTION FAILED!!\n");
        printf("*********************************\n");
    }
```

```
        printf("\n-----------------------------------------------------------------------\n");


    }//For statment



    //deallocate
    free(cmdmsg->filename);
    free(cmdmsg->fileType);

}

/* The file selection widget and the string to store the chosen filename */
GtkWidget *file_selector;
gchar *selected_filename;

GtkWidget     *view;
//The entry for the username
GtkWidget    *entry;
GtkWidget    *fileentry;


GtkWidget     *loggin;
GtkWidget     *list;
GtkWidget     *upload;
GtkWidget      *btnread;
GtkWidget *chkVisiblebtn;

/* The Text Buffer as a global variabel*/
GtkTextBuffer *buffer;
/* File descriptors for pipe. */
int fds[2];

/* Create a Button Box with the specified parameters */
GtkWidget *create_bbox (gint  horizontal,
                        char* title,
                        gint  spacing,
                        gint  child_w,
                        gint  child_h,
                        gint  layout)
{

    GtkWidget *frame;
    GtkWidget *bbox;
    GtkWidget *bbox2;

    frame = gtk_frame_new (title);

    if (horizontal)
        bbox = gtk_hbutton_box_new ();
    else
        bbox = gtk_vbutton_box_new ();


    bbox2 = gtk_vbutton_box_new ();


    //Read file entry
    fileentry = gtk_entry_new_with_max_length(160);


    gtk_container_set_border_width (GTK_CONTAINER (bbox), 5);
    gtk_container_add (GTK_CONTAINER (frame), bbox);

    gtk_container_set_border_width (GTK_CONTAINER (bbox2), 5);
    gtk_container_add (GTK_CONTAINER (bbox), bbox2);

    /* Set the appearance of the Button Box */
    gtk_button_box_set_layout (GTK_BUTTON_BOX (bbox), layout);
    gtk_button_box_set_spacing (GTK_BUTTON_BOX (bbox), spacing);
    gtk_button_box_set_child_size (GTK_BUTTON_BOX (bbox), child_w, child_h);


    btnread = gtk_button_new_with_label ("Read");
    gtk_container_add (GTK_CONTAINER (bbox2), btnread);

    gtk_container_add (GTK_CONTAINER (bbox2), fileentry);
```

```
    upload = gtk_button_new_with_label ("Upload");

    list = gtk_button_new_with_label ("List");
    gtk_container_add (GTK_CONTAINER (bbox), list);

    gtk_container_add (GTK_CONTAINER (bbox), upload);


        //disable buttons
    gtk_widget_set_sensitive (btnread, FALSE);
        gtk_widget_set_sensitive (upload, FALSE);
        gtk_widget_set_sensitive (list, FALSE);



    return(frame);

}

/* Create a Button Box with the specified parameters */
GtkWidget *create_Login (gint  horizontal,
                         char* title,
                         gint  spacing,
                         gint  child_w,
                         gint  child_h,
                         gint  layout)
{


    GtkWidget *frame;
    GtkWidget *bbox;
    GtkWidget *bbox2;
    GtkWidget *button;

    frame = gtk_frame_new (title);


    bbox = gtk_vbutton_box_new ();


    gtk_container_set_border_width (GTK_CONTAINER (bbox), 0);
    gtk_container_add (GTK_CONTAINER (frame), bbox);

    bbox2 = gtk_vbutton_box_new ();
    gtk_container_add (GTK_CONTAINER (bbox), bbox2);

    /* Set the appearance of the Button Box */
    gtk_button_box_set_layout (GTK_BUTTON_BOX (bbox), layout);
    gtk_button_box_set_spacing (GTK_BUTTON_BOX (bbox), spacing);
    gtk_button_box_set_child_size (GTK_BUTTON_BOX (bbox), child_w, child_h);


    /* create a text entry. */
    entry = gtk_entry_new_with_max_length(15);

    /* insert the following text into the text entry, as its initial value. */
    gtk_entry_set_text(GTK_ENTRY(entry), "");

    //Align the text in the entry
   // gtk_entry_set_alignment(entry , 0.5);

    loggin = gtk_button_new_with_label("loggin");


    gtk_container_add (GTK_CONTAINER (bbox), loggin);
    gtk_container_add (GTK_CONTAINER (bbox), entry);

    return(frame);

}

/* Create a Button Box with the specified parameters */
GtkWidget *create_label (gint  horizontal,
                         char* title,
                         gint  spacing,
                         gint  child_w,
                         gint  child_h,
                         gint  layout)
{
```

```
    GtkWidget *frame;
    GtkWidget *bbox;
    GtkWidget *bbox2;
    GtkWidget *button;

    frame = gtk_frame_new (title);


    bbox = gtk_vbutton_box_new ();


    //Check button for visibility
    chkVisiblebtn = gtk_check_button_new_with_label("Log Visibility");

    gtk_container_set_border_width (GTK_CONTAINER (bbox), 0);
    gtk_container_add (GTK_CONTAINER (frame), bbox);

    bbox2 = gtk_vbutton_box_new ();
    gtk_container_add (GTK_CONTAINER (bbox), bbox2);

    /* Set the appearance of the Button Box */
    gtk_button_box_set_layout (GTK_BUTTON_BOX (bbox), layout);
    gtk_button_box_set_spacing (GTK_BUTTON_BOX (bbox), spacing);
    gtk_button_box_set_child_size (GTK_BUTTON_BOX (bbox), child_w, child_h);


    gtk_container_add (GTK_CONTAINER (bbox), chkVisiblebtn);

    return(frame);

}


/* This is a callback function to watch data in standard output and write to a view text widget. */
void input_callback( gpointer          data,
                     gint              source,
                     GdkInputCondition condition )
{

    //Clear buffer
    buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW (view));
    gtk_text_buffer_set_text (buffer, "", -1);

    gchar buf[1024];
    gint chars_read;
    GtkTextIter iter;
    chars_read = 1024;
    gtk_text_buffer_get_end_iter(buffer, &iter);
    while (chars_read == 1024)
    {
        chars_read = read(fds[0], buf, 1024);
        // fprintf(stderr, "%i chars: %s\n", chars_read, buf);
        gtk_text_buffer_insert (buffer, &iter, buf, chars_read);
    }
}


/* This is a callback function. The data arguments are ignored
 * in this example. More on callbacks below. */
static void showfileselector(GtkWidget *widget,
                             gpointer data)
{
    gtk_widget_show (file_selector);
}

static void logVisibility( GtkWidget *widget,
                   gpointer   data )
{

    gtk_widget_show(view);


}

static void invisibleLog( GtkWidget *widget,
                          gpointer   data )
{

    gtk_widget_hide(view);
```

```
}

/* This is a callback function. The data arguments are ignored
 * in this example. More on callbacks below. */
static void listOutput( GtkWidget *widget,
                   gpointer   data )
{
    get_filelist();
}

void hide_file_selector(GtkFileSelection *selector, gpointer user_data)
{
    gtk_widget_hide(file_selector);
}

void store_filename(GtkFileSelection *selector, gpointer user_data)
{
    gtk_widget_hide(file_selector);

    char cwd[1024];
    char command[2048];

    bzero(cwd,sizeof(cwd));
    bzero(command,sizeof(command));

    getcwd(cwd, sizeof(cwd));

    // g_print("Current working dir: %s\n", cwd);
    selected_filename = gtk_file_selection_get_filename (GTK_FILE_SELECTION(file_selector));

    //g_print ("%s\n" ,selected_filename);

    g_print ("%s\n" , g_path_get_basename(selected_filename));

    // g_print("%s\n", cwd);

    sprintf(command,"cp %s %s", selected_filename, cwd);

    //g_print("%s\n", command);

    system(command);

    char* filename =g_strdup(g_path_get_basename(selected_filename) );

    struct cmd *cmdmsg = (struct cmd *) malloc((sizeof(struct cmd)));

    cmdmsg->filename=strdup(strtok(g_path_get_basename(selected_filename),"."));
    cmdmsg->fileType=strdup(strtok(NULL," "));


    cmdmsg->fileid = reqFileID(cmdmsg , clientID);

   // g_print("%s , %s\n" , cmdmsg->filename , cmdmsg->fileType);


    writer_oper(message_id, cmdmsg);

    g_free(filename);
    free(cmdmsg->filename);
    free(cmdmsg->fileType);

}

//xxx
void
on_button1_clicked(GtkButton* button, gpointer data)
{
    /* cast the data back to a GtkEntry*  */
    GtkEntry* entry = (GtkEntry*)data;

    //Request ClientID
    reqClientID(gtk_entry_get_text(entry));

    gtk_widget_set_sensitive(GTK_WIDGET(entry),FALSE);


        //Enable
        gtk_widget_set_sensitive (btnread, TRUE);
        gtk_widget_set_sensitive (upload, TRUE);
        gtk_widget_set_sensitive (list, TRUE);
```

```
   // gtk_widget_show(view);
   // gtk_widget_set_sensitive(,FALSE);

}

void
readfile_clicked(GtkButton* button, gpointer data)
{

    //Cast the data back to a GtkEntry
    GtkEntry* entry = (GtkEntry*)data;


    char* filename =g_strdup(gtk_entry_get_text(entry) );

    struct cmd *cmdmsg = (struct cmd *) malloc((sizeof(struct cmd)));

    cmdmsg->filename=strdup(strtok(filename,"."));
    cmdmsg->fileType=strdup(strtok(NULL," "));


    cmdmsg->fileid = reqFileID(cmdmsg , clientID);

    g_print("%s , %s\n" , cmdmsg->filename , cmdmsg->fileType);

    reader_oper(++message_id , cmdmsg );

    g_free(filename);
    free(cmdmsg->filename);
    free(cmdmsg->fileType);

}


int main( int   argc,
          char *argv[] )
{

    //Tha name of the file that holds all the information for the servers
    char *filename="config.txt";

    //Retrieve information for the servers
    readConfig(filename);

    inisialization();

    //Establish  connection with all available diretories & replicas  servers.
    conn2direc();
    connect2Replicas();

    //Establish connection with the filemanager
    conn2filemanager();
    //Gt// GtkWidget is the storage type for widgets//

    static GtkWidget* window = NULL;

    GtkWidget *scrolled_window;

    GtkWidget *main_vbox;

    GtkWidget *frame_horz;

    GtkWidget *frame_horz2;

    GtkWidget *vbox;

    GtkWidget *toolbar;
    GtkToolItem *saveTb;
    GtkToolItem *appearlog;
    GtkToolItem *exitTb;


    // Create a pipe. File descriptors for the two ends of the pipe are placed in fds.
    pipe (fds);
    // Redirect fds[1] to be writed with the standard output.
    dup2 (fds[1], STDOUT_FILENO);

    /* Initialize GTK */
    gtk_init( &argc, &argv );
```

```
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window), "Distributed File System");

//Set the position the main window in the Center
gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);

//Set the size the main window
gtk_window_set_default_size(GTK_WINDOW(window), 800, 800);

gtk_container_set_border_width (GTK_CONTAINER (window), 4);

main_vbox = gtk_vbox_new (FALSE, 0);
gtk_container_add (GTK_CONTAINER (window), main_vbox);

toolbar = gtk_toolbar_new();
gtk_toolbar_set_style(GTK_TOOLBAR(toolbar), GTK_TOOLBAR_ICONS);

/* Create the selector */
 file_selector = gtk_file_selection_new("Please select a file for editing.");

saveTb = gtk_tool_button_new_from_stock(GTK_STOCK_CLEAR );
gtk_toolbar_insert(GTK_TOOLBAR(toolbar), saveTb, -1);

appearlog = gtk_tool_button_new_from_stock(GTK_STOCK_DND );
gtk_toolbar_insert(GTK_TOOLBAR(toolbar), appearlog, -1);

exitTb = gtk_tool_button_new_from_stock(GTK_STOCK_QUIT);
gtk_toolbar_insert(GTK_TOOLBAR(toolbar), exitTb, -1);


g_signal_connect(G_OBJECT(appearlog), "clicked",
                 G_CALLBACK(logVisibility), NULL);

g_signal_connect(G_OBJECT(saveTb), "clicked",
                 G_CALLBACK(invisibleLog), NULL);


g_signal_connect(G_OBJECT(exitTb), "clicked",
                 G_CALLBACK(gtk_main_quit), NULL);


g_signal_connect(G_OBJECT(window), "destroy",
                 G_CALLBACK(gtk_main_quit), NULL);

gtk_box_pack_start(GTK_BOX(main_vbox), toolbar, FALSE, FALSE, 5);

frame_horz = gtk_frame_new ("Layout");
gtk_box_pack_start (GTK_BOX (main_vbox), frame_horz, TRUE, TRUE, 10);

frame_horz2 = gtk_frame_new ("Log file");

gtk_box_pack_start (GTK_BOX (main_vbox), frame_horz2, TRUE, TRUE, 10);

//Creates a new text view //
view = gtk_text_view_new ();

scrolled_window = gtk_scrolled_window_new(NULL, NULL);

 gtk_container_add (GTK_CONTAINER (frame_horz2), scrolled_window);

 gtk_container_add(GTK_CONTAINER(scrolled_window), view);

vbox = gtk_vbox_new (FALSE, 0);
gtk_container_set_border_width (GTK_CONTAINER (vbox), 10);
gtk_container_add (GTK_CONTAINER (frame_horz), vbox);

gtk_box_pack_start (GTK_BOX (vbox),
                    create_Login (FALSE, "Login",0, 0, 0, GTK_BUTTONBOX_EDGE),
                    FALSE, FALSE, 0);

gtk_box_pack_start (GTK_BOX (vbox),
                    create_bbox (TRUE, "Operations", 40, 85, 20, GTK_BUTTONBOX_START),
                    TRUE, TRUE, 0);

gtk_signal_connect(GTK_OBJECT(loggin), "clicked",
                   GTK_SIGNAL_FUNC(on_button1_clicked),
                   (gpointer)entry);
```

```
    gtk_signal_connect(GTK_OBJECT(btnread), "clicked",
                       GTK_SIGNAL_FUNC(readfile_clicked),
                       (gpointer) fileentry);


    g_signal_connect (G_OBJECT (upload), "clicked",
                      GTK_SIGNAL_FUNC(showfileselector) , NULL);

    g_signal_connect (G_OBJECT (list), "clicked",
                      GTK_SIGNAL_FUNC(listOutput) , NULL);

    gtk_signal_connect (GTK_OBJECT (GTK_FILE_SELECTION(file_selector)->ok_button),
                        "clicked", GTK_SIGNAL_FUNC (store_filename), NULL);

    gtk_signal_connect (GTK_OBJECT (GTK_FILE_SELECTION(file_selector)->cancel_button),
                        "clicked", GTK_SIGNAL_FUNC (hide_file_selector), NULL);


    /* This is the singnal connection to call input_callback when we have data in standard output read
end pipe. */
    gdk_input_add(fds[0], GDK_INPUT_READ, input_callback, NULL);

    /* always display the window as the last step so it all splashes on
     * the screen at once. */
    gtk_widget_show_all (window);

    gtk_main ();

    return(0);
}
```