Ατομική Διπλωματική Εργασία

ΕΠΕΚΤΑΣΗ ΚΑΤΑΝΕΜΗΜΕΝΟΥ ΣΥΣΤΗΜΑΤΟΣ ΑΡΧΕΙΩΝ ΚΑΙ ΔΗΜΙΟΥΡΓΙΑ ΔΙΕΠΑΦΗΣ ΧΡΗΣΤΗ

Σωτήρης Σωτηρίου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ



ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2017

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΠΕΚΤΑΣΗ ΚΑΤΑΝΕΜΗΜΕΝΟΥ ΣΥΣΤΗΜΑΤΟΣ ΑΡΧΕΙΩΝ ΚΑΙ ΔΗΜΙΟΥΡΓΙΑ ΔΙΕΠΑΦΗΣ ΧΡΗΣΤΗ

Σωτήρης Σωτηρίου

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 2017

Ευχαριστίες

Επιθυμώ να ευχαριστήσω τον επιβλέποντα καθηγητή της Ατομικής Διπλωματική Εργασίας μου Δρ. Χρύση Γεωργίου για την συνεχή υποστήριξη και καθοδήγηση που μου πρόσφερε καθ' όλη την διάρκεια της διπλωματικής μου εργασίας.

Επίσης, επιθυμώ να ευχαριστήσω την οικογένεια μου για την στήριξη που μου πρόσφεραν.

Περίληψη

Τα Κατανεμημένα Συστήματα Αρχείων (ΚΣΑ) παρέχουν την δυνατότητα στους χρήστες να διαχειρίζονται τα αρχεία τους, μέσω Διαδικτύου, απ' όπου και αν βρίσκονται καθώς και να τα μοιράζονται με άλλους χρήστες. Παράδειγμα ενός ΚΣΑ αποτελεί το Dropbox [6] όπου επιτρέπει στους χρήστες να ανεβάζουν τα αρχεία τους στους εξυπηρετητές (servers) του συστήματος και αν επιθυμούν μπορούν να τα μοιράζονται με άλλους χρήστες. Όμως, μέχρι τώρα δεν υπάρχει υλοποίηση ενός ΚΣΑ που να υπόσχεται ισχυρή συνέπεια δεδομένων στην παρουσία ταυτοχρονισμού. Για παράδειγμα, στο Dropbox αν υπάρχουν πολλοί χρήστες που μοιράζονται το ίδιο αρχείο την ίδια στιγμή τότε παρατηρούνται προβλήματα ασυνέπειας, δηλαδή διαφορετικοί χρήστες να έχουν διαφορετική εκδοχή του ίδιου αρχείου. Έτσι σκοπός της Ατομικής Διπλωματικής Εργασίας μου ήταν η ανάπτυξη ενός ολοκληρωμένου ΚΣΑ που να υπόσχεται ισχυρή συνέπεια στην παρουσία ταυτοχρονισμού και η δημιουργία μιας εύχρηστης διεπαφής για τον χρήστη στο λειτουργικό σύστημα Windows. Ένα μέρος του ΚΣΑ που υλοποιήσαμε βασίστηκε σε μια πρώιμη εκδοχή ΚΣΑ που δημιουργήθηκε σε προηγούμενη ΑΔΕ [7]. Επίσης, οι χρήστες του συστήματος που έχουμε υλοποιήσει έχουν την δυνατότητα να μοιράζονται τα αρχεία τους με άλλους χρήστες, να μετονομάζουν και να διαγράφουν τα αρχεία τους, να βλέπουν το ιστορικό του συστήματος, να καθορίσουν τι δικαιώματα θα έχουν οι άλλοι χρήστες για τα αρχεία που έχουν δημιουργήσει (είναι "owner" δηλαδή). Επιπλέον, ο κάθε χρήστης έχει τον προσωπικό του φάκελο σε περιβάλλον Windows όπου εκεί βρίσκονται τα αρχεία που ανέβασε στο σύστημα και τα αρχεία που κατέβασε από το σύστημα. Τέλος, το ΚΣΑ που έχει υλοποιηθεί είναι κατάλληλο για τοπική δίκτυα (LAN) αλλά και για δίκτυα ευρείας εμβέλειας (WAN).

Περιεχόμενα

Κεφάλαιο 1 Εισαγωγή **1**
1.1 Κίνητρο και σκοπός εργασίας 1
1.2 Μεθοδολογία που ακολουθήθηκε 2
1.3 Οργάνωση διπλωματικής εργασίας 2
Κεφάλαιο 2 Υπόβαθρο και προηγούμενη εργασία **4**
2.1 Κατανεμημένα συστήματα αρχείων 4
2.2 Συνέπεια μνήμης 5
2.3 Συστήματα απαρτίας 7
2.4 Προηγούμενη διπλωματική 8
Κεφάλαιο 3 Περιγραφή αλγορίθμων **10**
3.1 Αλγόριθμος ABD 10
3.1.1 Λειτουργία γραφής 11
3.1.2 Λειτουργία ανάγνωσης 11
3.2 Αλγόριθμος LDR 11
3.2.1 Πελάτης 12
3.2.2 Εξυπηρετητής ευρετηρίου 16
3.2.3 Εξυπηρετητής αρχείων 18
3.3 Τροποποιημένος αλγόριθμος LDR 19
Κεφάλαιο 4 Περιγραφή συστήματος **21**
4.1 Επικοινωνία 22
4.1.1 Μοντέλο Πελάτη-Εξυπηρετητή 22
4.1.2 Νήματα 23
4.1.3 Πρωτόκολλο TCP 23
4.1.4 Υποδοχές ροής 24
4.2 Διεπαφή 27
4.3 Πελάτης 28
4.4 Εξυπηρετητής ευρετηρίου 31
4.5 Εξυπηρετητής αρχείων 33
4.6 Διαχειριστής αρχείων και διαχειριστής πιστοποίησης 34
Κεφάλαιο 5 Περιγραφή διεπαφής **37**
5.1 Επεξήγηση οθονών 38
5.2 Εγγραφή 40
5.3 Είσοδος 41

5.4 Ανέβασμα αρχείου 42
5.5 Ανανέωση αρχείου 43
5.6 Κατέβασμα αρχείου 45
5.7 Προεπισκόπηση αρχείου 46
5.8 Διαγραφή αρχείου 46
5.9 Μετονομασία αρχείου 48
5.10 Ιστορικό 48
Κεφάλαιο 6 Πειραματική αξιολόγηση **49**
6.1 Προετοιμασία 49
6.2 Πλατφόρμα υλοποίησης 49
6.3 Σενάρια 50
6.4 Αποτελέσματα 51
Κεφάλαιο 7 Συμπεράσματα **55**
7.1 Γενικά συμπεράσματα 55
7.2 Προβλήματα και παραδοχές 56
7.3 Μελλοντική εργασία 56
Βιβλιογραφία **58**

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο και σκοπός εργασίας	1
1.2 Μεθοδολογία που ακολουθήθηκε	2
1.3 Οργάνωση διπλωματικής εργασίας	2

1.1 Κίνητρο και σκοπός εργασίας

Οι χρήστες ενός Κατανεμημένου Συστήματος Αρχείων (ΚΣΑ) [2,4] μπορούν να ανταλλάζουν αρχεία μεταξύ τους ακόμα και όταν βρίσκονται σε διαφορετικές γεωγραφικές τοποθεσίες. Επίσης, τα αρχεία που ανεβάζουν είναι κοινά για όλους και μπορούν να τα κατεβάσουν όλοι οι χρήστες. Ακόμη έχουν την δυνατότητα να τροποποιήσουν τα αρχεία στον τοπικό τους χώρο και μετά αν ανεβάσουν το τροποποιημένο αρχείο τότε αυτό θα γραφτεί στην θέση του υπάρχοντος αρχείου. Ένα παράδειγμα ΚΣΑ αποτελεί το Dropbox[6], το οποίο υποστηρίζει ένα ασθενή τύπο συνέπειας, που ονομάζεται Eventual Consistency [2]. Συγκεκριμένα, το σύστημα δεν μας εγγυάται ότι το αρχείο που θα διαβάσει ένας χρήστης θα είναι πάντα το πιο πρόσφατο. Δηλαδή, αν υπάρχουν πολλαπλοί χρήστες που δουλεύουν στο ίδιο αρχείο την ίδια χρονική στιγμή, τότε παρατηρείται ασυνέπεια. Αν όμως σταματήσουν να υπάρχουν νέες ενημερώσεις στο αρχείο, τότε το σύστημα μας εγγυάται ότι τελικά το αρχείο που θα διαβάσει ο χρήστης θα είναι το πιο πρόσφατο. Ακόμη ένα παράδειγμα ΚΣΑ αποτελεί το Google File System (GFS) όπου και αυτό υποστηρίζει ένα ασθενή τύπο συνέπειας, το Relaxed Consistency [2], όπου δεν παρέχει οποιεσδήποτε εγγυήσεις σε θέματα συνέπειας στην παρουσία ταυτοχρονισμού. Να αναφέρουμε ότι δεν είναι μόνο το Dropbox και το GFS που δεν εγγυόνται συνέπεια δεδομένων στην παρουσία ταυτοχρονισμού, αλλά, εξ όσων γνωρίζουμε, κανένα από τα ΚΣΑ που έχουν υλοποιηθεί μέχρι τώρα δεν την εγγυάται.

Έτσι κύριος σκοπός της διπλωματικής εργασίας ήταν η δημιουργία ενός ολοκληρωμένου ΚΣΑ το οποίο να υπόσχεται ισχυρή συνέπεια δεδομένων στην παρουσία ταυτοχρονισμού. Το σύστημα για να θεωρείται ολοκληρωμένο θα πρέπει να παρέχει όλες τις βασικές λειτουργίες που αναμένεται να έχει ένα σύστημα αρχείων. Δηλαδή, οι χρήστες του συστήματος να μπορούν

να δημιουργήσουν ένα νέο αρχείο, να το ανοίξουν, να το επεξεργαστούν/διαφοροποιήσουν, να το κλείσουν, να το διαγράψουν και να το μετονομάσουν. Τέλος, σκοπός της ΑΔΕ ήταν και η δημιουργία μιας εύχρηστης διεπαφής για περιβάλλον Windows μέσω της οποίας οι χρήστες να μπορούν να εκτελούν τις λειτουργίες που προσφέρει το σύστημα, χωρίς να αντιλαμβάνονται την κατανεμημένη φύση του συστήματος.

1.2 Μεθοδολογία που ακολουθήθηκε

Η μεθοδολογία που ακολουθήθηκε για την υλοποίηση της ΑΔΕ είναι η εξής:
Αρχικά, έγινε μελέτη άρθρων για κατανεμημένα συστήματα αρχείων και ειδικότερα για τους δύο βασικούς αλγόριθμους ABD και LDR όπου βασίστηκε το σύστημα μας. Ακολούθως έγινε μελέτη της ΑΔΕ του Ηλία Σπανού [7] για περαιτέρω κατανόηση του θεωρητικού υπόβαθρου. Συγκεκριμένα, το ΚΣΑ που υλοποίησε ο Ηλίας είναι ένα *πρώιμο* κατανεμημένο σύστημα αρχείων που λειτουργεί σε περιβάλλον Unix και ήταν βασισμένο στους αλγόριθμους ABD και LDR . Επίσης, έχει περιορισμούς στις λειτουργείς αρχείων που παρέχει το σύστημα (π.χ. δεν μπορείς να διαγράψεις ένα αρχείο) και ο διαχειριστής αρχείων που είχε ο Ηλίας ήταν κεντρικοποιημένος. Ακόμη, η διεπαφή για τους χρηστές του συστήματος ήταν αποκλειστικά για Unix. Έπειτα, δημιουργήθηκε από την αρχή ο κώδικας του πελάτη (client) έτσι ώστε να μπορεί να λειτουργήσει σε Windows. Ακολούθως, δημιουργήθηκε η διεπαφή του συστήματος μέσω του εργαλείου Qt [9] και προστέθηκαν νέες λειτουργίες όπως η δυνατότητα εγγραφής χρηστών, διαγραφή αρχείων κ.α. στις οποίες θα αναφερθούμε αναλυτικά στα επόμενα κεφάλαια. Ακόμη, κάναμε τον διαχειριστή αρχείων κατανεμημένο. Επιπλέον, δημιουργήθηκε ένας διαχειριστής πιστοποίησης όπου εγγράφει τους χρήστες και ελέγχει τα στοιχεία που δίνει ο χρήστης για είσοδο στο σύστημα. Τέλος, υπήρξε μια πειραματική αξιολόγηση του συστήματος όπου ταυτόχρονα με αυτήν κάναμε και απασφαλμάτωση της υλοποίηση μας.

1.3 Οργάνωση διπλωματικής εργασίας

Το έγγραφο της ΑΔΕ αποτελείται από εφτά (7) Κεφάλαια. Στο Κεφάλαιο 1 αναφέρονται τα κίνητρα, ο σκοπός και η μεθοδολογία της διπλωματικής εργασίας. Στο Κεφάλαιο 2 αναφέρονται κάποιες βασικές έννοιες και δίδεται το αναγκαίο υπόβαθρο, ενώ στο Κεφάλαιο 3 γίνεται περιγραφή των αλγορίθμων ABD [7] και LDR [1,7,10] όπου στηρίζεται το ΚΣΑ που υλοποιήσαμε . Ακολούθως, στο Κεφάλαιο 4, γίνεται αναλυτική περιγραφή του συστήματος που υλοποιήθηκε. Στην συνέχεια, στο Κεφάλαιο 5 γίνεται μια αναλυτική περιγραφή της διεπαφής του συστήματος και ο τρόπος χρήσης της. Στο Κεφάλαιο 6 περιγράφεται η πειραματική αξιολόγηση του συστήματος. Τέλος, στο Κεφάλαιο 7 δίνονται τα γενικά

συμπεράσματα, προβλήματα, παραδοχές και κάποιες μελλοντικές εργασίες που θα μπορούσαν να πραγματοποιηθούν.

Κεφάλαιο 2

Υπόβαθρο και προηγούμενη εργασία

2.1 Κατανεμημένα συστήματα αρχείων	4
2.2 Συνέπεια μνήμης	5
2.3 Συστήματα απαρτίας	7
2.4 Προηγούμενη διπλωματική	8

2.1 Κατανεμημένα συστήματα αρχείων

Κατανεμημένο σύστημα αρχείων (ΚΣΑ) είναι ένα σύστημα αρχείων, όπου κατανέμεται μεταξύ πολλαπλών μηχανών και εμφανίζεται στον χρήστη σαν ένα κεντρικοποιημένο σύστημα αρχείων [2,4]. Το ΚΣΑ αποτελεί μια ειδική περίπτωση των κατανεμημένων συστημάτων, και επιτρέπει σε πολλούς χρήστες να μοιράζονται μεταξύ τους αρχεία απ' όπου και αν βρίσκονται. Επίσης, κάθε μηχανή έχει τον δικό της προσωπικό χώρο μνήμης και μπορεί να ανταλλάξει δεδομένα με μια άλλη απομακρυσμένη μηχανή. Τα ΚΣΑ παρέχουν την δυνατότητα στους χρήστες να δημιουργούν αρχεία, να τα διαβάζουν, να τα επεξεργάζονται, να τα διαγράφουν κ.τ.λ., δηλαδή προσφέρει τις ίδιες δυνατότητες με αυτές που προσφέρει το σύστημα αρχείων που έχει ένας υπολογιστής. Επίσης η επικοινωνία μεταξύ των μηχανών μπορεί να πραγματοποιηθεί μέσω ενός δικτύου όπου μπορεί να είναι τοπικό (LAN) ή ευρείας εμβέλειας (WAN). Ένα πλεονέκτημα των κατανεμημένων συστημάτων είναι η υψηλή απόδοση αφού οι υπολογισμοί κατανέμονται μεταξύ αρκετών μηχανών. Και άρα η επέκταση ενός κατανεμημένου συστήματος θα έχει ως αποτέλεσμα την αύξηση της απόδοσης του. Ακόμη ένα πλεονέκτημα είναι η καλή ανοχή σφαλμάτων, καθ' ότι υπάρχουν πολλαπλά αντίγραφα των δεδομένων έτσι αν καταρρεύσουν μέχρι ένα συγκεκριμένο αριθμό μηχανών, το σύστημα θα μπορεί να συνεχίσει κανονικά την λειτουργία του.

Ένα ΚΣΑ μπορεί να μοντελοποιηθεί πάνω σε κατανεμημένη κοινόχρηστη μνήμη [5] όπου αποτελεί μια μορφή αρχιτεκτονικής μνήμης όπου φυσικά χωρισμένες μνήμες μπορούν να αντιμετωπιστούν ως ένας λογικά κοινόχρηστος χώρος διευθύνσεων. Κάθε μνήμη προσφέρει ένα μέρος της για την κοινόχρηστη μνήμη με αποτέλεσμα να δημιουργείται ένας μεγάλος

εικονικός χώρος διευθύνσεων. Σε αυτή την κοινόχρηστη μνήμη μπορούν να έχουν πρόσβαση πολλαπλές μηχανές είτε για ανάγνωση της μνήμης είτε για γράψιμο στην μνήμη. Ακόμη, η πρόσβαση στην μνήμη μπορεί να είναι σειριακή ή ταυτόχρονη.

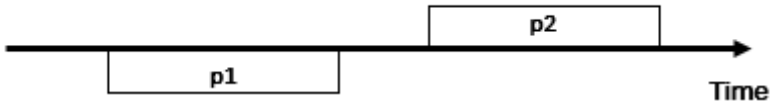
2.2 Συνέπεια μνήμης

Με τη χρήση κατανεμημένων συστημάτων αρχείων υπάρχει η ανάγκη για σωστή διαχείριση της μνήμης έτσι ώστε με κάθε ενέργεια που κάνουμε να είμαστε σίγουροι πως θα έχουμε τα σωστά αποτελέσματα. Με τη σωστή διαχείριση της μνήμης έχουμε συνέπεια μνήμης και γνωρίζουμε πως κάθε φορά που θα διαβάζει ο χρήστης την τιμή μιας μεταβλητής για παράδειγμα, θα ξέρει σίγουρα πως είναι η πιο πρόσφατη τιμή. Για να υπάρχει όμως συνέπεια μνήμης πρέπει να ορίσουμε κάποιο κανόνα όπου σε κάθε ανάγνωση πρέπει να ισχύει. Ένας κανόνας θα μπορούσε να είναι ο εξής: κάθε ανάγνωση επιστρέφει την τιμή που αποθηκεύτηκε από την πιο πρόσφατη εγγραφή. Όμως, για να μπορέσουμε να εντοπίσουμε ποια είναι η πιο πρόσφατη γραφή πρέπει να αποθηκεύουμε μαζί με την τιμή και κάποια άλλα δεδομένα. Υπάρχουν διάφορα μοντέλα συνέπειας μνήμης, όμως στη παρούσα διπλωματική εργασία θα ασχοληθούμε μόνο με το μοντέλο της ατομικής/γραμμικής μνήμης (Linearizability/atomic consistency) το οποίο μας εγγυάται συνέπεια μνήμης ακόμη και στην παρουσία υψηλού βαθμού ταυτοχρονισμού. Τώρα θα εξηγήσουμε αυτό το μοντέλο.

Ορισμός ατομικότητας:

Έστω ότι έχουμε ένα αντικείμενο (π.χ. μεταβλητή) το οποίο μπορεί να αποθηκεύει μια τιμή από ένα σύνολο πιθανών τιμών. Επίσης, για να διαβάσει κάποιος την τιμή του αντικειμένου πρέπει να εκτελέσει την λειτουργία ανάγνωσης (read) στο συγκεκριμένο αντικείμενο. Αν κάποιος επιθυμεί να γράψει μια νέα τιμή στο αντικείμενο αντικαθιστώντας την υπάρχουσα πρέπει να εκτελέσει την λειτουργία γραφής (write) στο συγκεκριμένο αντικείμενο. Με την παρουσία όμως ταυτόχρονων αναγνώσεων και εγγραφών πρέπει να ορίσουμε την ατομικότητα έτσι ώστε να έχουμε συνέπεια μνήμης στο αντικείμενο.

Για να ορίσουμε την ατομικότητα θα πρέπει πρώτα να ορίσουμε την έννοια της προτεραιότητας μεταξύ λειτουργειών. Έστω ότι έχουμε δύο λειτουργίες την p1 και p2 όπως φαίνεται και στο Σχήμα 2.1, όπου μπορούν να είναι λειτουργίες ανάγνωσης ή εγγραφής. Τότε θα λέμε πως μια λειτουργία προηγείται της άλλης αν η απάντηση της πρώτης συμβαίνει χρονικά πριν από την απάντηση της άλλης. Στο παράδειγμα που φαίνεται στο Σχήμα 2.2 η λειτουργία p1 προηγείται της p2.



Σχήμα 2.1

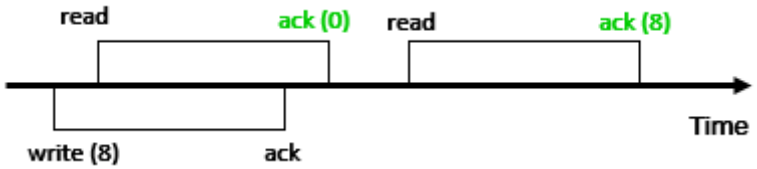
Τώρα θα ορίσουμε τις συνθήκες που πρέπει να ισχύουν για να ικανοποιείται η ατομικότητα και είναι οι εξής :

1. 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.
2. Εάν μια λειτουργία ανάγνωσης 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.
3. Αν έχουμε δύο διαδοχικές αναγνώσεις (μη επικαλυπτόμενες), τότε η δεύτερη ανάγνωση επιστρέφει την ίδια ή μια πιο πρόσφατη τιμή από αυτή της πρώτης.

Τέλος θα παρουσιάσουμε κάποια παραδείγματα για καλύτερη κατανόηση της έννοιας της ατομικότητας και των συνθηκών της:

Παράδειγμα 1

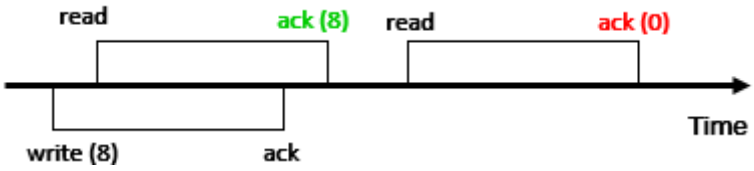
Στο πρώτο μας παράδειγμα έχουμε αρχικά μια λειτουργία ανάγνωσης η οποία επιστρέφει την τιμή 0 και εκτελείται ταυτόχρονα με μια λειτουργία γραφής η οποία γράφει την τιμή 8 και πιο μετά έχουμε μια λειτουργία ανάγνωσης η οποία επιστρέφει την τιμή 8. Το συγκεκριμένο παράδειγμα φαίνεται διαγραμματικά στο Σχήμα 2.2. Βάση της 2ης συνθήκης που ορίσαμε πιο πάνω η πρώτη λειτουργεία ανάγνωσης θα μπορούσε να επιστρέψει την τιμή 0 ή 8. Όπως βλέπουμε και στο σχήμα, επιστρέφει ορθά την τιμή 0. Ακολούθως, η δεύτερη ανάγνωση βάση της 1ης συνθήκης πρέπει να επιστρέψει την τιμή 8, ενώ βάση με την 3η συνθήκη θα μπορούσε να επιστρέψει την τιμή 0 ή 8. Επομένως, ορθά επιστρέφει την τιμή 8 ικανοποιώντας και τις 2 συνθήκες. Άρα στο συγκεκριμένο παράδειγμα δεν παραβιάζεται καμιά συνθήκη της ατομικότητας και έτσι έχουμε συνέπεια μνήμης.



Σχήμα 2.2

Παράδειγμα 2

Στο δεύτερο μας παράδειγμα έχουμε αρχικά μια λειτουργία γραφής η οποία γράφει την τιμή 8 και εκτελείται ταυτόχρονα με μια λειτουργία ανάγνωσης η οποία επιστρέφει την τιμή 8 και πιο μετά έχουμε μια λειτουργία ανάγνωσης η οποία επιστρέφει την τιμή 0. Το συγκεκριμένο παράδειγμα φαίνεται διαγραμματικά στο Σχήμα 2.3. Βάση της 2ης συνθήκη που ορίσαμε πιο πάνω η πρώτη λειτουργεία ανάγνωσης θα μπορούσε να επιστρέψει την τιμή 0 ή 8. Όπως βλέπουμε και στο σχήμα, επιστρέφει ορθά την τιμή 8. Ακολούθως, η δεύτερη ανάγνωση βάση με την 1η συνθήκη πρέπει να επιστρέψει την τιμή 8 και βάση με την 3η συνθήκη θα πρέπει να επιστρέψει την τιμή 8. Όμως, λανθασμένα επιστρέφει την τιμή 0 αφού δεν ικανοποιείται η 1η αλλά ούτε και η 3η συνθήκη. Άρα, στο συγκεκριμένο παράδειγμα δεν εφαρμόζεται η έννοια της ατομικότητας και έτσι δεν έχουμε συνέπεια μνήμης.



Σχήμα 2.3

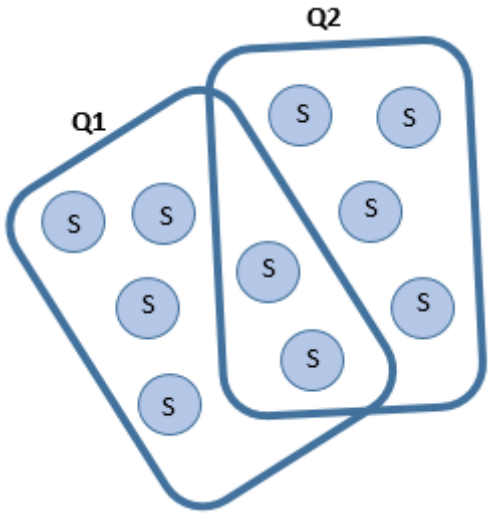
2.3 Συστήματα απαρτίας

Αν χρησιμοποιούσαμε μόνο ένα χώρο για αποθήκευση των τιμών των αντικειμένων τότε θα είχαμε το πρόβλημα του bottleneck και single point of failure. Αφού όλες οι αιτήσεις για διάβασμα και γραφή θα πηγαίνανε στον μοναδικό χώρο αποθήκευσης με αποτέλεσμα να παρατηρείται συμφόρηση, αυτό είναι το bottleneck. Επίσης, σε περίπτωση κατάρρευσης του μοναδικού χώρου τότε το σύστημα δεν θα μπορούσε να συνεχίσει να λειτουργεί ωσότου επανέλθει ο χώρος αποθήκευσης, αυτό είναι το single point of failure. Αυτά τα δύο προβλήματα καθιστούν το σύστημα μας μη-αποδοτικό και δεν υπάρχει καλή ανοχή σφαλμάτων.

Χρησιμοποιώντας σύστημα απαρτίας μπορούμε να επιλύσουμε τα πιο πάνω προβλήματα. Στην παρούσα διπλωματική εργασία ασχοληθήκαμε μόνο με το σύστημα απαρτίας της πλειοψηφίας. Συγκεκριμένα, αν έχουμε N εξυπηρετητές (servers) όπου θα αποθηκεύουμε τα δεδομένα μας σε αυτούς, τότε χρησιμοποιώντας το σύστημα απαρτίας της πλειοψηφίας θα πρέπει κάθε φορά να γράφουμε τα δεδομένα μας σε ⌈(N+1)/2⌉ εξυπηρετητές. Όμως, χρησιμοποιώντας το

σύστημα απαρτίας της πλειοψηφίας παρουσιάζεται το εξής πρόβλημα: μπορεί κάποιοι εξυπηρετητές να μην λάβουν όλα τα δεδομένα και ακολούθως αν ζητήσουμε δεδομένα από αυτούς τους εξυπηρετητές θα λάβουμε λανθασμένα δεδομένα. Για την επίλυση αυτού του προβλήματος εφαρμόσαμε την εξής μέθοδο: όταν θα λαμβάνουμε δεδομένα από τους εξυπηρετητές θα περιμένουμε απαντήσεις από ⌈(N+1)/2⌉ εξυπηρετητές. Με αυτό τον τρόπο θα ξέρουμε πως τουλάχιστον μία από τις απαντήσεις περιλαμβάνει τα σωστά δεδομένα έτσι εξασφαλίζουμε συνέπεια των δεδομένα. Ακόμη, έχουμε το πλεονέκτημα ότι αν καταρρεύσουν μέχρι και ⌈N/2⌉ – 1 εξυπηρετητές το σύστημα θα μπορεί να συνεχίσει να λειτουργεί κανονικά.

Το σύστημα απαρτίας αποτελείται από μια συλλογή από σύνολα $Q=\{Q_1,\ldots Q_n\}$, γνωστά ως quorums, από τα οποία το κάθε ένα περιέχει ένα σύνολο εξυπηρετητών $S= \{S_1,\ldots,S_j\}$ για κάθε σύνολο Q_i. Το σύστημα απαρτίας είναι ένα υποσύνολο του S, έτσι ώστε κάθε δύο υποσύνολα του Q έχουν μια κοινή τομή. Στο Σχήμα 2.4 βλέπουμε ένα παράδειγμα συστήματος απαρτίας όπου αποτελείται από δύο quorums και το καθ' ένα έχει 6 εξυπηρετητές. Επίσης, παρατηρούμε πως τα δύο αυτά quorums έχουν 2 κοινούς εξυπηρετητές.



Σχήμα 2.4

2.4 Προηγούμενη διπλωματική

Η διπλωματική μου εργασία αποτελεί επέκταση της διπλωματικής εργασίας του Ηλία Σπανού [7]. Ο Ηλίας υλοποίησε ένα πρώιμο κατανεμημένο σύστημα αρχείων σε Unix, όπου υπήρχαν εξυπηρετητές ευρετηρίου (directories), εξυπηρετητές αρχείων (replicas), πελάτες (clients) και ένας διαχειριστής αρχείων (filemanager). Να σημειώσουμε ότι ο διαχειριστής αρχείων ήταν κεντρικοποιημένος. Επίσης, το σύστημα που υλοποίησε ήταν βασισμένο στους αλγόριθμους

ABD και LDR. Επιπλέον, μαζί με το σύστημα δημιούργησε και μια απλή διεπαφή όπου και αυτή ήτανε για περιβάλλον Unix.

Στην διπλωματική μου εργασία κρατήθηκαν οι δομές δεδομένων και ο τρόπος αποστολής μηνυμάτων μεταξύ πελάτη-εξυπηρετητών που χρησιμοποίησε ο Ηλίας. Όπως ανάφερα και πιο πάνω η υλοποίηση του Ηλία ήταν για το λειτουργικό σύστημα Unix, γι' αυτό αλλάξαμε ολόκληρο τον πελάτη έτσι ώστε να λειτουργεί σε Windows. Ακόμη, ο διαχειριστής αρχείων που είχε ο Ηλίας ήταν κεντρικοποιημένος, έτσι το κάναμε κατανεμημένο. Ο λόγος που το κάναμε αυτό ήταν επειδή σε περίπτωση κατάρρευσης του διαχειριστή αρχείων το σύστημα δεν θα μπορούσε να λειτουργήσει, ως επακόλουθο το σύστημα δεν θα είχε καλή ανοχή στα σφάλματα. Ακόμη, η υλοποίηση του Ηλία δεν παρέχει την δυνατότητα εγγραφής χρηστών γι' αυτό δημιουργήθηκε ένας διαχειριστής πιστοποίησης όπου εγγράφει τους χρήστες και ελέγχει αν τα στοιχεία που δίνουν οι χρήστες για να εισέλθουν στο σύστημα είναι σωστά. Επίσης, προστέθηκε η δυνατότητα διαγραφής και μετονομασίας ενός αρχείου. Ακόμη, υπάρχει ιστορικό με τις τελευταίες ενέργειες που έχουν πραγματοποιηθεί από τους χρήστες. Ως επίσης, κάθε χρήστης μετά την επιτυχή εγγραφή του στο σύστημα δημιουργείται ένας προσωπικός φάκελος στο περιβάλλον Windows όπου εκεί τοποθετούνται τα αρχεία που ανέβασε στο σύστημα και τα αρχεία που κατέβασε από το σύστημα ο χρήστης. Επιπλέον, ο κάθε χρήστης μπορεί να καθορίσει τα δικαιώματα που θα έχουν οι άλλοι χρήστες για τα αρχεία που έχει δημιουργήσει (δηλαδή είναι ο "owner" των αρχείων αυτών). Επιπλέον, το σύστημα αποθηκεύει μια λίστα αρχείων την οποία βλέπει συνεχώς ο χρήστης και εκεί περιλαμβάνονται όλα τα αρχεία του συστήματος. Η λίστα αυτή υπήρχε και στον Ηλία αλλά τώρα έχουν προστεθεί νέα στοιχεία για το κάθε αρχείο όπως ποιος είναι ο owner, τα δικαιώματα του αρχείου κ.τ.λ. Ακόμη, προστέθηκε η δυνατότητα στον χρήστη να έχει προεπισκόπηση ενός αρχείου και αν το επιθυμεί θα μπορεί μετά να το κατεβάσει στον προσωπικό του φάκελο. Επίσης, υπάρχουν εφεδρικά (backup) δεδομένα των εξυπηρετητών και των πελατών, έτσι σε περίπτωση κατάρρευσής τους όταν θα επανέλθουν μετά θα μπορούν να συνεχίσουν από εκεί που έμειναν. Τέλος, κάναμε αποσφαλμάτωση (debug) τον κώδικα του εξυπηρετητή αρχείων που είχε υλοποιήσει ο Ηλίας για να εντοπίσουμε τυχόν προβλήματα. Και εντοπίσαμε το εξής πρόβλημα: αν υπήρχαν ταυτόχρονες εγγραφές τότε ο εξυπηρετητής αρχείων εξυπηρετούσε τις εγγραφές ταυτόχρονα πράγμα το οποίο είναι λάθος αφού αν οι ταυτόχρονες εγγραφές ήτανε για το ίδιο αρχείο τότε το τελικό αρχείο αλλοιώνεται αφού όλοι οι χρήστες γράφουν τα δεδομένα τους πάνω στο ίδιο αρχείο. Γι' αυτό αλλάξαμε τον εξυπηρετητή αρχείων έτσι ώστε να εξυπηρετεί τις εγγραφές σειριακά, δηλαδή κάθε εγγραφή να είναι ατομική.

Κεφάλαιο 3

Περιγραφή αλγορίθμων

3.1 Αλγόριθμος ABD 10
3.1.1 Λειτουργία γραφής 11
3.1.2 Λειτουργία ανάγνωσης 11
3.2 Αλγόριθμος LDR 11
3.2.1 Πελάτης 12
3.2.2 Εξυπηρετητής ευρετηρίου 16
3.2.3 Εξυπηρετητής αρχείων 18
3.3 Τροποποιημένος αλγόριθμος LDR 19

3.1 Αλγόριθμος ABD

Ο αλγόριθμος ABD προτάθηκε από τους Attiya, Bar-Noy και Dolev και προσομοίωνε μια single-writer/multi-reader κοινόχρηστη μνήμη μέσα σε ένα δίκτυο όπου ανταλλάζονται μηνύματα [7,10]. Δηλαδή, μόνο ένας χρήστης μπορεί να γράφει ενώ όλοι οι χρήστες μπορούν να διαβάσουν. Ακολούθως, ο αλγόριθμος αυτός επεκτάθηκε και προσομοίωνε multi-writer/multi-reader ενώ μπορούσε να διαχειριστεί και δυναμικά σετ από χρήστες. Δηλαδή, τώρα μπορούν όλοι οι χρήστες να γράφουν και να διαβάζουν. Στην παρούσα διπλωματική χρησιμοποιήσαμε την επέκταση του αλγορίθμου ABD και άρα στο εξής όταν θα αναφέρουμε τον αλγόριθμο ABD θα μιλάμε για την επέκταση του ABD. Ο αλγόριθμος ABD μπορεί να χρησιμοποιηθεί σαν αλγόριθμος αναπαραγωγής, δηλαδή να υπάρχει αποθήκευση των ίδιων δεδομένων σε διαφορετικά μέρη και μπορεί να χρησιμοποιηθεί σε συστήματα απαρτίας. Στην συγκεκριμένη περίπτωση θα χρησιμοποιήσουμε το σύστημα απαρτίας της πλειοψηφίας όπως αναφέραμε στο Κεφάλαιο 2.

Ας υποθέσουμε πως έχουμε ένα σύνολο από εξυπηρετητές αρχείων (replicas) όπου ο καθένας μπορεί να αποθηκεύσει μια τιμή. Επιπλέον, έχουμε κάποιους χρήστες που μπορούν να διαβάζουν την τιμή που έχει αποθηκευμένη ο κάθε εξυπηρετητής αρχείων. Ακόμη, οι χρήστες μπορούν και να γράψουν μια νέα τιμή στους εξυπηρετητές αρχείων αντικαθιστώντας την ήδη

υπάρχουσα. Ο κάθε εξυπηρετητής αρχείων έχει αποθηκευμένο το εξής ζεύγος: <value, tag>. Όπου η μεταβλητή value έχει την τελευταία τιμή που έχουν δώσει οι χρήστες για να γραφτεί και το tag είναι η χρονοσφραγίδα της τιμής. Το tag είναι ένας φυσικός αριθμός όπου αυξάνεται κάθε φορά που γίνεται μια νέα γραφή. Άρα, το tag δείχνει πόσο πρόσφατη είναι η τιμή και συγκεκριμένα όσο πιο μεγάλο είναι το tag τόσο πιο πρόσφατη είναι η τιμή.

3.1.1 Λειτουργία γραφής

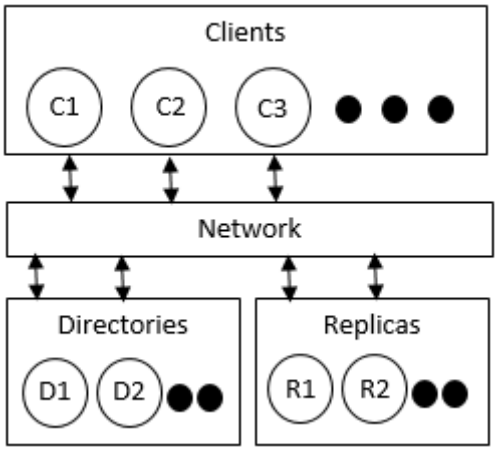
Για να γράψει μια τιμή ο χρήστης πρέπει πρώτα να διαβάσει από μια πλειοψηφία εξυπηρετητών αρχείων το tag που έχει αποθηκευμένο ο κάθε εξυπηρετητής. Ακολούθως, ο χρήστης επιλέγει το μεγαλύτερο tag από αυτά που έχει διαβάσει και το αυξάνει κατά ένα. Τέλος, στέλνει σε μια πλειοψηφία από εξυπηρετητές την τιμή που επιθυμεί να γράψει μαζί με το tag που υπολόγισε.

3.1.2 Λειτουργία ανάγνωσης

Για να διαβάσει ο χρήστης μια τιμή πρώτα διαβάζει από μια πλειοψηφία εξυπηρετητών αρχείων την τιμή που έχει αποθηκευμένη ο κάθε εξυπηρετητής μαζί με το αντίστοιχο tag της. Ακολούθως, ο χρήστης επιλέγει την τιμή που είχε το μεγαλύτερο tag από αυτές που έχει διαβάσει. Τέλος, γράφει σε μια πλειοψηφία εξυπηρετητών αρχείων την τιμή και το tag που επέλεξε. Η φάση που ο χρήστης διαβάζει από μια πλειοψηφία εξυπηρετητών αρχείων ονομάζεται query phase, ενώ η φάση που ο χρήστης γράφει σε μια πλειοψηφία εξυπηρετητών ονομάζεται propagation phase.

3.2 Αλγόριθμος LDR

Ο αλγόριθμος Layer Data Replication LDR [7,10] εργάζεται σε ένα αξιόπιστο και ασύγχρονο δίκτυο ανταλλαγής μηνυμάτων όπως και ο ABD. Επίσης, σε αυτό το δίκτυο έχουμε κάποιους πελάτες (clients) και κάποιους εξυπηρετητές (servers). Έχουμε μόνο δύο ειδών εξυπηρετητών, τους εξυπηρετητές ευρετηρίου (directories) και τους εξυπηρετητές αρχείων (replicas). Οι εξυπηρετητές αρχείων αποθηκεύουν την τιμή μιας μεταβλητής, έστω x. Και οι εξυπηρετητές ευρετηρίου αποθηκεύουν ποιοι εξυπηρετητές αρχείων έχουν την πιο πρόσφατη τιμή της x. Τέλος, το γράψιμο μιας τιμής μπορεί να πραγματοποιηθεί σε όλους τους εξυπηρετητές αρχείων, όμως το διάβασμα μιας τιμής πρέπει να πραγματοποιηθεί μόνο στους εξυπηρετητές αρχείων που έχουν την πιο πρόσφατή τιμή. Ένα σχεδιάγραμμα της αρχιτεκτονικής του αλγόριθμου LDR φαίνεται στο Σχήμα 3.1 όπου έχουμε πολλαπλούς πελάτες (C1,C2…), εξυπηρετητές ευρετηρίου (D1,D2…) και εξυπηρετητές αρχείων (R1,R2…).



Σχήμα 3.1

Η ιδέα του αλγόριθμου LDR είναι να αποθηκεύονται τα δεδομένα σε τυχαίες τοποθεσίες (δηλ. στους εξυπηρετητές αρχείων) και μετά χρησιμοποιώντας τους εξυπηρετητές ευρετηρίου (directories) να μπορεί να εντοπίσει τα πιο πρόσφατα δεδομένα. Έχουμε όμως την εξής δυσκολία: οι πληροφορίες που έχουν αποθηκευμένες οι εξυπηρετητές ευρετηρίου πρέπει να συμβαδίζουν με τα δεδομένα που έχουν οι εξυπηρετητές αρχείων. Έτσι αν ο χρήστης ζητήσει μια τιμή, τότε μέσω των εξυπηρετητών ευρετηρίου θα πρέπει να εντοπιστεί όντως η πιο πρόσφατη τιμή. Ο αλγόριθμος LDR χρησιμοποιεί την ιδέα του αλγόριθμου ABD όμως ο αλγόριθμος LDR είναι πιο αποδοτικός γιατί οι διεργασίες διενεργούνται στα μεταδομένα που βρίσκονται στους εξυπηρετητές ευρετηρίου αντί στα πραγματικά δεδομένα που βρίσκονται στους εξυπηρετητές αρχείων. Και αφού το μέγεθος των μεταδεδομένων είναι πιο μικρό από αυτό των πραγματικών δεδομένων οι λειτουργίες χρειάζονται λιγότερο χρόνο. Όμως, στην παρούσα διπλωματική εργασία χρησιμοποιήσαμε τον τροποποιημένο αλγόριθμο LDR [1]. Περισσότερες λεπτομέρειες για αυτό τον αλγόριθμο θα δώσουμε στην συνέχεια στο Υποκεφάλαιο 3.3.

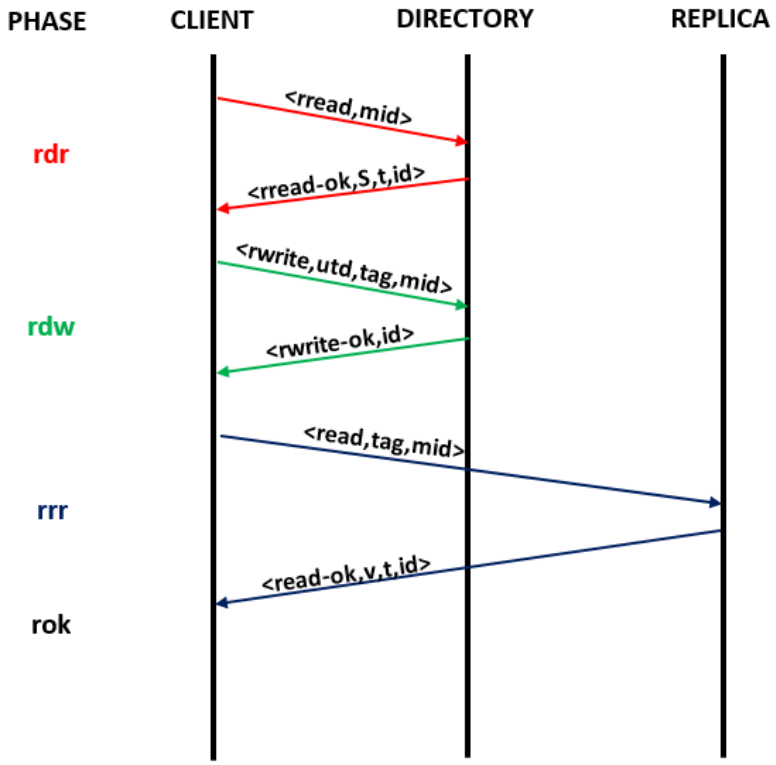
3.2.1 Πελάτης

Ο πελάτης λαμβάνει εντολές από τον χρήστη για να γράψει μια τιμή στην μεταβλητή x ή να διαβάσει την τιμή της μεταβλητής x. Όταν ένας πελάτης στέλνει ένα μήνυμα τότε συμπεριλαμβάνει και ένα ακέραιο αριθμό το mid όπου ουσιαστικά είναι το id του μηνύματος και είναι μοναδικό. Ακόμη, όταν απαντούν οι εξυπηρετητές στον πελάτη τότε στο μήνυμα συμπεριλαμβάνουν το mid που είχε το μήνυμα που παρέλαβαν. Έτσι, όταν θα λαμβάνει την απάντηση ο πελάτης θα μπορεί να υπολογίζει πόσο πρόσφατη είναι η απάντηση. Αν ο πελάτης

λάβει απάντηση και το mid της απάντησης είναι μικρότερο από το mid που έχει αποθηκευμένο τότε αυτό σημαίνει πως η απάντηση είναι παλιά και άρα την αγνοεί.

Λειτουργία ανάγνωσης

Αν ο πελάτης Ci λάβει το μήνυμα readi από τον χρήστη τότε ξεκινάει να εκτελεί τις ενέργειες της λειτουργίας ανάγνωσης. Όπως φαίνεται και στο Σχήμα 3.2 η λειτουργία ανάγνωσης χωρίζεται σε 4 φάσεις και έχουν την εξής ακολουθία: read-directories-read (rdr), read-directories-write (rdw), read-replicas-read (rrr) και read-ok (rok).



Σχήμα 3.2 Λειτουργία ανάγνωσης

- read-directories-read (rdr)

  Ο πελάτης Ci στέλνει ένα μήνυμα της μορφής <rread,mid> σε όλους τους εξυπηρετητές ευρετηρίου για να εντοπίσει ένα σετ από εξυπηρετητές αρχείων που έχουν την πιο πρόσφατη τιμή του x. Ακολούθως, περιμένει μια πλειοψηφία απαντήσεων από τους εξυπηρετητές ευρετηρίου με τις απαντήσεις να έχουν την εξής μορφή: <rread-ok,S,t,id> όπου S είναι ένα σετ από εξυπηρετητές αρχείων, t είναι το tag και το id είναι ίσο με το mid του μηνύματος που έστειλε ο πελάτης. Μετά, ο Ci επιλέγει ένα ζευγάρι (S,t) όπου t είναι το μεγαλύτερο tag που εντοπίστηκε από τις απαντήσεις και S είναι οι εξυπηρετητές

αρχείων που έχουν αυτό το tag, δηλαδή έχουν την πιο πρόσφατη τιμή του x. Μετέπειτα, ο Ci θέτει το (utd,tag) να είναι το ίδιο με το (S,t). Και εδώ τερματίζεται η φάση rdr.

- read-directories-write (rdw)

  Ο Ci στέλνει ένα μήνυμα της μορφής <rwrite,utd,tag,mid> σε όλους τους εξυπηρετητές ευρετηρίου. Ο λόγος που στέλνεται αυτό το μήνυμα είναι για να διασφαλίσουμε ότι οποιαδήποτε ανάγνωση μετά από αυτή που βρισκόμαστε τώρα, θα εντοπίσει μια τιμή με tag ίσο με το tag του μηνύματος ή ακόμη και μεγαλύτερο. Ακολούθως, ο Ci περιμένει μια πλειοψηφία απαντήσεων από τους εξυπηρετητές ευρετηρίου και οι απαντήσεις έχουν την εξής μορφή: <rwrite-ok,mid>. Όταν λάβει την πλειοψηφία τότε η φάση rdw τερματίζεται.

- read-replicas-read (rrr)

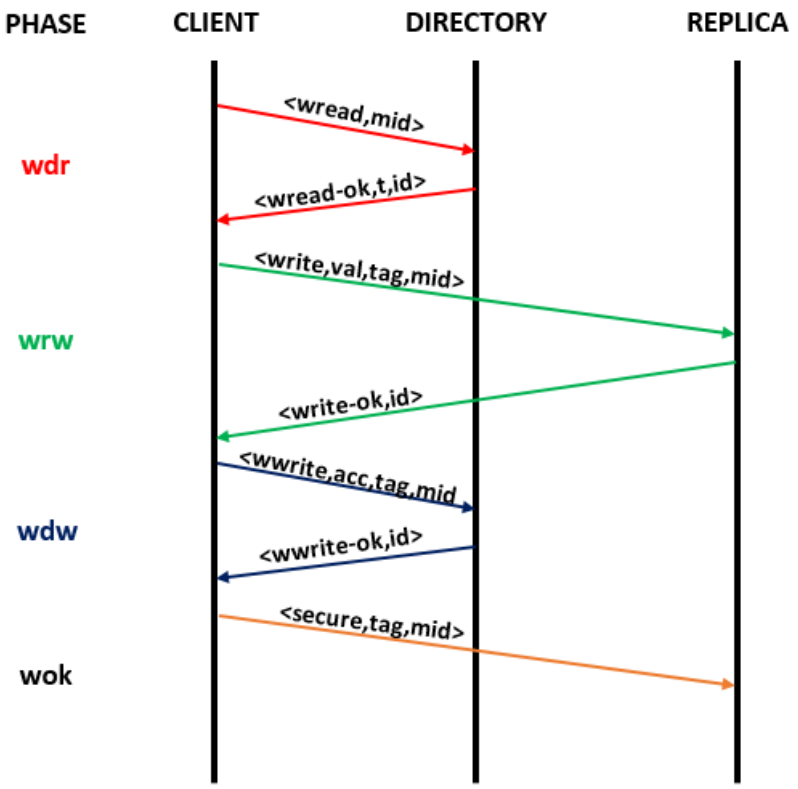
  Ο Ci στέλνει μήνυμα στους εξυπηρετητές αρχείων που βρίσκονται στο σύνολο utd με την εξής μορφή: <read,tag,mid>. Ο λόγος που στέλνεται και το tag είναι επειδή κάθε εξυπηρετητής αρχείων έχει πολλαπλές τιμές του x και γι' αυτό ζητάει την τιμή που έχει ως tag το tag του μηνύματος. Οι εξυπηρετητές αρχείων απαντούν με το εξής μήνυμα: <read-ok,v,t,id> όπου v είναι η τιμή της x που έχει ως tag το t. Όταν λάβει έστω και μια απάντηση από τους εξυπηρετητές αρχείων τότε η φάση rrr τερματίζεται.

- read-ok (rok)

  Ο Ci στέλνει στον χρήστη το μήνυμα read-ok(v)i, όπου το v περιέχει την πιο πρόσφατη τιμή της μεταβλητής x.

Λειτουργία γραφής

Αν ο πελάτης Ci λάβει το μήνυμα write(v)i από τον χρήστη τότε ξεκινάει να εκτελεί τις ενέργειες της λειτουργίας γραφής. Δηλαδή, ο χρήστης ζητάει από τον πελάτη Ci να γράψει την τιμή v στην x. Όπως φαίνεται και στο Σχήμα 3.3 η λειτουργία γραφής χωρίζεται σε 4 φάσεις και έχουν την εξής ακολουθία: write-directories-read (wdr), write-replicas-write (wrw), write-directories-write (wdw) και write-ok (wok).



Σχήμα 3.3 Λειτουργία γραφής

- write-directories-read (wdr)

  Ο Ci αρχικά θα προσπαθήσει να εντοπίσει το μεγαλύτερο tag και γι' αυτό στέλνει ένα μήνυμα της μορφής <wread,mid> σε όλους τους εξυπηρετητές ευρετηρίου. Ακολούθως, ο πελάτης περιμένει μια πλειοψηφία απαντήσεων από τους εξυπηρετητές ευρετηρίου με τις απαντήσεις να έχουν την εξής μορφή: <wread-ok,t,id> όπου το t είναι το tag της τιμής και το id είναι ίσο με το mid του μηνύματος που έστειλε ο Ci. Μετά, ο πελάτης εντοπίζει το μεγαλύτερο tag από τις απαντήσεις που έλαβε και θέτει t = (n,i΄) όπου n είναι το μεγαλύτερο tag και i΄ είναι το ID του πελάτη που έγραψε την πιο πρόσφατη τιμή. Μετέπειτα, ο Ci θέτει tag = (n +1,i) και val = v, όπου i είναι το ID του πελάτη Ci. Εδώ τερματίζει η φάση wdr.

- write-replicas-write (wrw)

  Ο Ci στέλνει το εξής μήνυμα σε όλους τους εξυπηρετητές αρχείων: <write,val,tag,mid>. Με αυτό το μήνυμα ζητάει από τους εξυπηρετητές αρχείων να γράψουν την τιμή val στην μεταβλητή x μαζί με το tag που υπάρχει στο μήνυμα. Ακολούθως, ο Ci περιμένει απαντήσεις από f+1 εξυπηρετητές αρχείων. Έτσι, αν καταρρεύσουν μέχρι και f εξυπηρετητές αρχείων το σύστημα θα συνεχίσει να διατηρεί την τιμή της x. Συγκεκριμένα,

η τιμή f είναι μια εκτίμηση για τον αριθμό των εξυπηρετητών που ενδέχεται να καταρρεύσουν. Όταν λάβει απαντήσεις από f+1 εξυπηρετητές αρχείων τότε τους προσθέτει στο σετ acc και τερματίζεται η φάση wrw.

- write-directories-write (wdw)

  Ο Ci στέλνει ένα μήνυμα με την εξής μορφή σε όλους τους εξυπηρετητές ευρετηρίου: <wwrite,acc,tag,mid>. Συγκεκριμένα ο Ci ενημερώνει τους εξυπηρετητές ευρετηρίου πως οι εξυπηρετητές αρχείων που βρίσκονται στο σετ acc έχουν την πιο πρόσφατη τιμή του x και τιμή αυτή έχει ως tag ίσο με αυτό του μηνύματος. Ακολούθως, ο Ci περιμένει μια πλειοψηφία απαντήσεων από τους εξυπηρετητές ευρετηρίου με την απάντηση να έχει την μορφή <wwrite-ok,id>. Όταν λάβει πλειοψηφία απαντήσεων τότε τερματίζει η φάση wdw.

- write-ok (wok)

  Ο Ci στέλνει ένα μήνυμα σε όλους του εξυπηρετητές αρχείων που βρίσκονται στο σετ acc με την εξής μορφή: <secure,tag,mid>. Και ακολούθως ο Ci στέλνει στον χρήστη το μήνυμα write-oki. Ο λόγος που στέλνεται το secure μήνυμα είναι για να ενημερώσουμε τους εξυπηρετητές αρχείων πως μια πλειοψηφία εξυπηρετητών ευρετηρίου γνωρίζει για την εγγραφή με το συγκεκριμένο tag και άρα στο μέλλον δεν θα ξανά χρειαστούμε τιμές που έχουν ως tag μικρότερο από το tag του secure μηνύματος. Έτσι, επιτρέπει στους εξυπηρετητές αρχείων να διαγράψουν όλες τις τιμές που έχουν ως tag μικρότερο από το tag του secure μηνύματος.

3.2.2 Εξυπηρετητής ευρετηρίου

Ο εξυπηρετητής ευρετηρίου αποθηκεύει στο utd ένα σετ εξυπηρετητών αρχείων που πιστεύει πως έχουν την πιο πρόσφατη τιμή του x μαζί με το tag της το οποίο είναι αποθηκευμένο στην μεταβλητή tag. Υπάρχει η περίπτωση όπου ένας εξυπηρετητής ευρετηρίου να έχει αποθηκευμένα λανθασμένα στοιχεία, συγκεκριμένα οι εξυπηρετητές αρχείων που υπάρχουν στο utd να μην περιλαμβάνουν την τιμή με το συγκεκριμένο tag. Αυτό μπορεί να συμβεί όταν ο εξυπηρετητής ευρετηρίου δεν λάβει ενημέρωση για την νέα τιμή της x και το tag της, έτσι θα παραμείνουν τα δεδομένα τους όπως έχουν, όμως η παλιά τιμή στους εξυπηρετητές αρχείων θα διαγραφεί. Τέλος, ο εξυπηρετητής ευρετηρίου λαμβάνει μηνύματα μόνο από τους πελάτες και τους επιτρέπει να διαβάσουν και να γράψουν στο utd και στο tag.

Οι εξυπηρετητές ευρετηρίου μπορούν να λάβουν τα εξής μηνύματα από τους πελάτες:

- <rread, mid>

  Αν λάβει αυτό το μήνυμα σημαίνει πως ο πελάτης βρίσκεται στην λειτουργία ανάγνωσης. Στο συγκεκριμένο μήνυμα επιστρέφει στον πελάτη το utd και το tag που έχει αποθηκευμένα. Η μορφή του μηνύματος που στέλνει είναι η εξής: <rread-ok,utd,tag,mid>.

- <rwrite, S, t, mid>

  Αν λάβει αυτό το μήνυμα σημαίνει πως ο πελάτης βρίσκεται στην λειτουργία ανάγνωσης. Το S είναι σετ από εξυπηρετητές αρχείων που έχουν ως tag ίσο με το t του μηνύματος. Πρώτα, ο εξυπηρετητής ευρετηρίου ελέγχει αν το t είναι μικρότερο από το tag που έχει αποθηκευμένο, αν ισχύει αυτό τότε η γραφή είναι out-of-date και στέλνει στον πελάτη το μήνυμα <rwrite-ok,id> και τερματίζει. Αλλιώς, αν το t είναι ίσο με το tag τότε το S περιέχει σετ από εξυπηρετητές αρχείων που έχουν την πιο πρόσφατη τιμή και τους προσθέτει στο utd. Ακολούθως στέλνει στον πελάτη το μήνυμα <rwrite-ok,id> και τερματίζει. Αλλιώς, αν το t είναι μεγαλύτερο από το tag τότε το S περιέχει σετ από εξυπηρετητές αρχείων που έχουν μια πιο πρόσφατη τιμή από αυτούς που βρίσκονται στο utd. Ακολούθως, ελέγχει αν το μέγεθος του S είναι μεγαλύτερο από το f, αν είναι μεγαλύτερο τότε διαγράφει τα δεδομένα του utd και το tag και τα αντικαθιστά με το S και t αντίστοιχα. Αν όμως το μέγεθος του S είναι μικρότερο ή ίσο από το f τότε το μήνυμα αγνοείται. Και στις δύο περιπτώσεις απαντάει με το μήνυμα <rwrite-ok,id> και μετά τερματίζει.

- <wread, mid>

  Αν λάβει αυτό το μήνυμα σημαίνει πως ο πελάτης βρίσκεται στην λειτουργία γραφής. Στο συγκεκριμένο μήνυμα επιστρέφει στον πελάτη το utd και το tag που έχει αποθηκευμένα. Η μορφή του μηνύματος που στέλνει είναι η εξής: <wread-ok,utd,tag,mid>.

- <wwrite, S, t, mid>

  Αν λάβει αυτό το μήνυμα σημαίνει πως ο πελάτης βρίσκεται στην λειτουργία γραφής. Η διαδικασία που ακολουθεί στην περίπτωση αυτού του μηνύματος είναι η ίδια με αυτή του μηνύματος <rwrite,S,t,mid>. Η μόνη διαφορά είναι στην απάντηση, και συγκεκριμένα η απάντηση στο μήνυμα <wwrite,S,t,mid> είναι <wwrite-ok,id>.

Ο έλεγχος για το μέγεθος του S που υπάρχει στη μηνύματα rwrite και wwrite είναι για να είμαστε βέβαιοι πως θα παραμείνει τουλάχιστον ένα εξυπηρετητής αρχείων που δεν κατέρρευσε και έχει την πιο πρόσφατη τιμή, ακόμη και στην περίπτωση που καταρρεύσουν f εξυπηρετητές αρχείων.

3.2.3 Εξυπηρετητής αρχείων

Οι εξυπηρετητές αρχείων αποθηκεύουν πολλαπλές τιμές του x όπου οι πελάτες μπορούν να τις διαβάσουν ή να τις αντικαταστήσουν με μια άλλη. Επίσης, για κάθε τιμή του x που έχει αποθηκευμένη ο εξυπηρετητής αρχείου αποθηκεύει και το αντίστοιχο tag της. Το tag όπως αναφέραμε και προηγούμενος δείχνει πόσο πρόσφατη είναι η τιμή, όσο πιο μεγάλο τόσο πιο πρόσφατη είναι και η τιμή. Επίσης, μαζί με κάθε τιμή αποθηκεύεται και το secure bit όπου οι μόνες τιμές που μπορεί να έχει είναι 0 ή 1. Αν είναι 0 τότε σημαίνει πως η εγγραφή της συγκεκριμένης τιμής δεν έχει ακόμη ολοκληρωθεί και το tag της είναι unsecured. Ενώ, αν είναι 1 τότε σημαίνει πως η εγγραφή της συγκεκριμένης τιμής έχει ολοκληρωθεί με επιτυχία και το tag της είναι secured. Τέλος, οι εξυπηρετητές αρχείων λαμβάνουν μηνύματα μόνο από τους πελάτες.

Οι εξυπηρετητές αρχείων μπορούν να λάβουν τα εξής μηνύματα από τους πελάτες:

- <read, t, mid>

  Αν λάβει αυτό το μήνυμα σημαίνει πως ο πελάτης βρίσκεται στην λειτουργία ανάγνωσης. Αρχικά ο εξυπηρετητής αρχείου ελέγχει αν έχει αποθηκευμένη τιμή του x που να έχει tag ίσο με το t. Αν υπάρχει, τότε απαντάει στον πελάτη με το μήνυμα <read-ok,v,tag,mid> όπου v είναι η τιμή της x και το tag είναι ίσο με το t. Όταν στείλει το μήνυμα τότε τερματίζει. Να σημειώσουμε πως στην συγκεκριμένη περίπτωση δεν μας ενδιαφέρει το secure bit γι' αυτό και δεν έχουμε κάποιο σχετικό έλεγχο. Αν όμως δεν υπάρχει τιμή με tag ίσο με το t τότε επιστρέφεται η μεγαλύτερη secured τιμή. Για να υπολογιστεί αυτή η τιμή χρησιμοποιούμε την συνάρτηση maxst όπου ο τρόπος λειτουργίας της φαίνεται πιο κάτω στο Σχήμα 3.4. Ακολούθως, όταν εντοπιστεί η μεγαλύτερη secured τιμή την στέλνει στον πελάτη μαζί με το tag της σε ένα μήνυμα της μορφής <read-ok,v,tag,mid> και μετά τερματίζει.

$$\text{maxst}(data) = \begin{cases} (v,t) & ((v,t,1) \in data) \wedge ((v',t',1) \in data \Rightarrow t \geq t') \\ (v_0,t_0) & \nexists v,t : (v,t,1) \in data \end{cases}$$

Σχήμα 3.4

- <write, v, t, mid>

  Αν λάβει αυτό το μήνυμα σημαίνει πως ο πελάτης βρίσκεται στην λειτουργία γραφής. Αυτό που κάνει ο εξυπηρετητής αρχείου είναι να προσθέσει μια νέα εγγραφή στα δεδομένα του με τα εξής στοιχεία: η τιμή θα είναι το v του μηνύματος, ως tag θα έχει το t του μηνύματος και το secure bit θα είναι 0. Ακολούθως, απαντάει στον πελάτη με το μήνυμα <write-ok,id> και τερματίζει.

- <secure, t, mid>

  Αν λάβει αυτό το μήνυμα σημαίνει πως ο πελάτης βρίσκεται στην λειτουργία γραφής και είναι σίγουρος πως μια πλειοψηφία εξυπηρετητών ευρετηρίου γνωρίζει για την νέα τιμή της x, και αυτή η τιμή έχει προστεθεί σε τουλάχιστον f+1 εξυπηρετητές αρχείων. Αρχικά, ο εξυπηρετητής αρχείου ελέγχει αν υπάρχει η εγγραφή (v,t,0) και αν υπάρχει την διαγράφει και μετά προσθέτει την εγγραφή (v,t,1). Αν δεν υπάρχει η εγγραφή (v,t,0) στα δεδομένα τότε απλώς προσθέτει την εγγραφή (v,t,1). Ακολούθως, χρησιμοποιείται ο garbage-collector όπου αναλαμβάνει να διαγράψει όλες τις τιμές που έχουν tag μικρότερο από το t του μηνύματος και όταν τελειώσει τερματίζει.

Ο λόγος που διαγράφουμε τις παλιές τιμές είναι επειδή είμαστε σίγουροι πως η νέα τιμή έχει γραφτεί σε τουλάχιστον f+1 εξυπηρετητές αρχείων και πως γνωρίζουν για την νέα τιμή η πλειοψηφία των εξυπηρετητών ευρετηρίου. Έτσι, ξέρουμε ότι οι παλιές τιμές (δηλαδή αυτές που έχουν ως tag μικρότερο από αυτό της νέας εγγραφής) δεν θα χρειαστούν ξανά αφού θα υπάρχει σίγουρα tag μεγαλύτερο από τις δικές τους και γι’ αυτό τον λόγο τις διαγράφουμε.

3.3 Τροποποιημένος αλγόριθμος LDR

Παρόλο που ο αλγόριθμος LDR [7,10] είναι αρκετά αποδοτικός για την μεταφορά μεγάλων αντικειμένων όπως αρχείων, δεν ελέγχει αν το αρχείο που προσπαθείς να γράψεις είχε τροποποιηθεί από κάποιον άλλο πελάτη πριν εκτελέσεις την λειτουργία γραφής. Ο τροποποιημένος αλγόριθμος LDR [1] έρχεται να επιλύσει το πρόβλημα αυτό και κατά την λειτουργία της γραφής ελέγχει 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 ακολουθεί της εξής ενέργειες: Αρχικά ο πελάτης Ci στέλνει αίτημα σε όλους τους εξυπηρετητές ευρετηρίου για να μάθει ποιο είναι το πιο πρόσφατο tag του αρχείου. Όπως και στον αλγόριθμο ABD, ο πελάτης Ci περιμένει απάντηση από μια πλειοψηφία εξυπηρετητών ευρετηρίου. Μόλις πάρει τις απαντήσεις διαλέγει το μεγαλύτερο tag που υπάρχει στις απαντήσεις και ας το ονομάσουμε t1. Έστω t2 το tag του αρχείου που έχει ο πελάτης στο σύστημα του και είναι πάνω σε αυτό που έκανε τις αλλαγές του. Αν το t1 ισούται με το t2 τότε συνεχίζει κανονικά η λειτουργία γραφής όπως εξηγήσαμε πιο πάνω. Αν όμως το t1 είναι μεγαλύτερο από το t2 σημαίνει πως οι αλλαγές που έκανε ο πελάτης ήτανε σε μια παλιά έκδοση του αρχείου. Έτσι, η λειτουργία γραφής μετατρέπεται σε λειτουργία ανάγνωσης και ο πελάτης ζήτα από τους εξυπηρετητές αρχείων να του στείλουν το αρχείο με το tag t1. Πριν όμως ζητήσει το αρχείο, ενημερώνει τους εξυπηρετητές ευρετηρίου με το t1 όπως συμβαίνει και στην δεύτερη φάση της λειτουργίας ανάγνωσης του αλγορίθμου ABD.

Κεφάλαιο 4

Περιγραφή συστήματος

4.1 Επικοινωνία 22

4.1.1 Μοντέλο Πελάτη-Εξυπηρετητή 22

4.1.2 Νήματα 23

4.1.3 Πρωτόκολλο TCP 23

4.1.4 Υποδοχές ροής 24

4.2 Διεπαφή 27

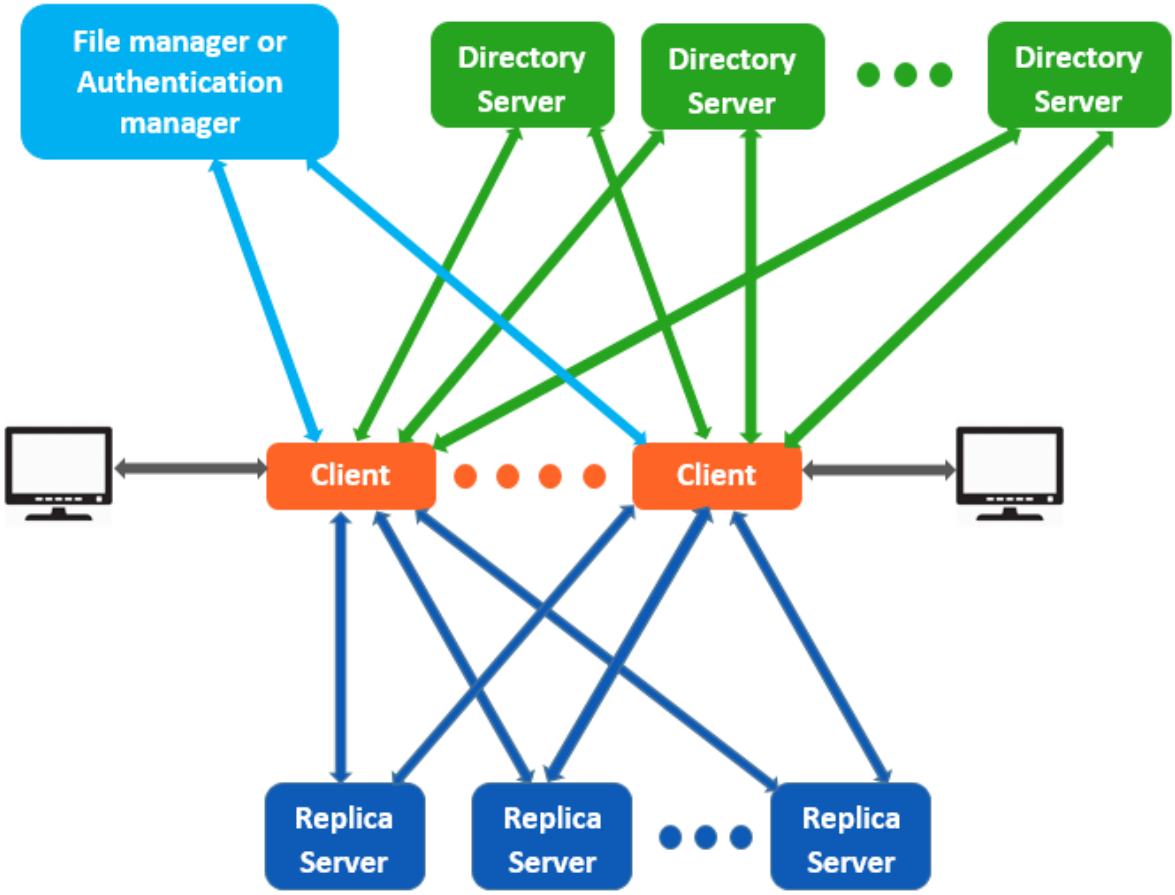
4.3 Πελάτης 28

4.4 Εξυπηρετητής ευρετηρίου 31

4.5 Εξυπηρετητής αρχείων 33

4.6 Διαχειριστής αρχείων και διαχειριστής πιστοποίησης 34

Στην Σχήμα 4.1 παρουσιάζεται μια γενική εικόνα του συστήματος που υλοποιήσαμε. Οι χρήστες, οι εξυπηρετητές ευρετηρίου (directories), οι εξυπηρετητές αρχείων (replicas) και ο διαχειριστής αρχείων (file manager) ή διαχειριστής πιστοποίησης (authentication manager) επικοινωνούν μόνο με τον πελάτη (client). Ενώ ο πελάτης επικοινωνεί με όλους αυτούς. Η επικοινωνία που χρησιμοποιήθηκε ακολουθεί το μοντέλο πελάτη-εξυπηρετητή με το πρωτόκολλο TCP και για να επιτευχθεί η επικοινωνία χρησιμοποιήθηκαν υποδοχές ροής. Επίσης, για τους εξυπηρετητές υπήρχε η ανάγκη για παράλληλη εξυπηρέτηση πολλαπλών χρηστών και για να επιτευχθεί αυτό χρησιμοποιήσαμε νήματα. Έχουμε δύο εκδοχές του συστήματος, όπου στην πρώτη υπάρχει διαχειριστής αρχείων ενώ στη δεύτερη εκδοχή υπάρχει διαχειριστής πιστοποίησης. Περισσότερες λεπτομέρειες για τις δύο αυτές εκδοχές θα δώσουμε στο Υποκεφάλαιο 4.6.



Σχήμα 4.1

4.1 Επικοινωνία

4.1.1 Μοντέλο Πελάτη-Εξυπηρετητή

Το μοντέλο πελάτη-εξυπηρετητή [3] έχει την δομή μιας κατανεμημένης εφαρμογής όπου οι πελάτες (Clients) επικοινωνούν με τον εξυπηρετητή (Server) μέσω ενός δικτύου. Οι πελάτες ζητάνε υπηρεσίες από τον εξυπηρετητή και ο εξυπηρετητής παρέχει υπηρεσίες στους πελάτες. Ένας πελάτης δεν μοιράζεται τα δεδομένα του με άλλους πελάτες, όμως τα δεδομένα του εξυπηρετητή μπορούν να μοιραστούν σε διάφορους πελάτες, αν του το ζητήσουν οι ίδιοι. Αυτό το μοντέλο χρησιμοποιείτε πολύ σε δίκτυα ευρείας εμβέλειας (WAN) και τοπικά δίκτυα (LAN). Επίσης, υπάρχουν αρκετές εφαρμογές που χρησιμοποιούν αυτό το μοντέλο και κάποιες από αυτές είναι ο παγκόσμιο ιστός (World Wide Web), το ηλεκτρονικό ταχυδρομείο (Email) κ.α.

Αρχικά εγκαθιδρύεται σύνδεση μεταξύ των πελατών και του εξυπηρετητή. Ακολούθως, οι πελάτες κάνουν αιτήσεις στον εξυπηρετητή και ο εξυπηρετητής αναλαμβάνει να εξυπηρετήσει τις αιτήσεις στέλνοντας τα κατάλληλα δεδομένα. Ο εξυπηρετητής μπορεί να εξυπηρετεί τις

αιτήσεις σειριακά ή ταυτόχρονα. Στην παρούσα διπλωματική οι εξυπηρετητές εξυπηρετούν τις αιτήσεις ταυτόχρονα και για να επιτευχθεί αυτό χρησιμοποιήσαμε νήματα (threads).

4.1.2 Νήματα

Όπως αναφέραμε και πιο πάνω για την ταυτόχρονη εξυπηρέτηση πελάτων χρησιμοποιήσαμε νήματα. Συγκεκριμένα, αν ο εξυπηρετητής λάβει μια αίτηση τότε δημιουργείται ένα νέο νήμα όπου σκοπό έχει να εξυπηρετήσει την συγκεκριμένη αίτηση. Το νήμα τερματίζεται όταν έχει εξυπηρετηθεί η αίτηση που ανέλαβε. Γενικά όμως τα νήματα αποτελούν σημαντικό κομμάτι για κάποιες εφαρμογές αφού σου προσφέρουν την δυνατότητα να εκτελείς παράλληλα διαφορετικές εργασίες χρησιμοποιώντας παραπάνω από ένα επεξεργαστή. Υπάρχουν και άλλοι τρόποι με τους οποίος μπορούμε να πετύχουμε ταυτόχρονη εξυπηρέτηση. Όπως για παράδειγμα με τη χρήση διεργασιών. Όμως, αν συγκρίνουμε τα νήματα με τις διεργασίες θα δούμε ότι τα νήματα είναι πιο αποδοτικά σε θέμα χρόνου και μνήμης καθώς η δημιουργία διεργασιών γίνεται με την εντολή fork() η οποία δεσμεύει αρκετή μνήμη και η εναλλαγή διεργασιών είναι 5 φορές πιο αργή από την εναλλαγή νημάτων.

Για την χρήση νημάτων χρησιμοποιήσαμε την βιβλιοθήκη “pthread.h”. Για την δημιουργία ενός νήματος χρησιμοποιούμε την συνάρτηση pthread_create(). Ενώ για τον τερματισμό ενός νήματος πρέπει να χρησιμοποιήσουμε την συνάρτηση pthread_exit().

Όμως με την χρήση νημάτων μπορεί να παρουσιαστούν προβλήματα με την συνέπεια μνήμης. Έστω ότι έχουμε 2 νήματα να εκτελούνται ταυτόχρονα και πάνε και τα δύο να γράψουνε μια διαφορετική τιμή σε μια κοινή μεταβλητή. Τότε δεν θα είμαστε βέβαιοι για την τελική τιμής της μεταβλητής. Για επίλυση αυτού του προβλήματος χρησιμοποιήσαμε σημαφόρους. Συγκεκριμένα, χρησιμοποιήσαμε την συνάρτηση pthread_mutex_lock() η οποία κλειδώνει ένα συγκεκριμένο χώρο της μνήμης και έτσι κανένα άλλο νήμα δεν θα έχει πρόσβαση σε αυτή. Ακολούθως, όταν το νήμα τελειώσει με την εργασία που είχε να κάνει με τον συγκεκριμένο χώρο μνήμης, για τον ξεκλειδώσουμε πρέπει να χρησιμοποιήσουμε την συνάρτηση pthread_mutex_unlock(). Οι δύο αυτές συναρτήσεις συμπεριλαμβάνονται στην βιβλιοθήκη “pthread.h”.

4.1.3 Πρωτόκολλο TCP

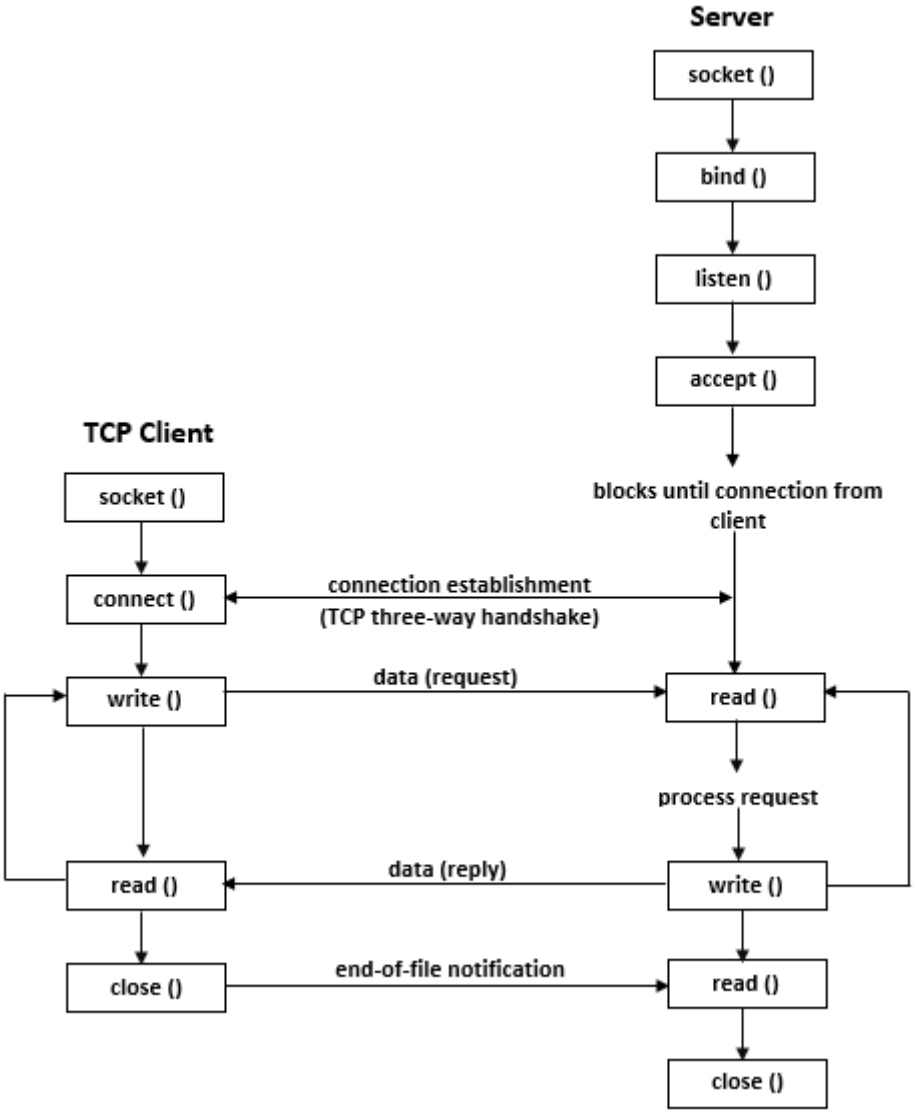
Στα δίκτυα υπολογιστών, πρωτόκολλο είναι ένα σύνολο από συμβάσεις που καθορίζουν το πως ανταλλάσσουν μεταξύ τους δεδομένα οι υπολογιστές του δικτύου. Οι περισσότερες

σύγχρονες υπηρεσίες στο διαδίκτυο βασίζονται στο πρωτόκολλο TCP [11] όπως για παράδειγμα το HTTP και World Wide Web. Το TCP παρέχει αξιοπιστία, εγγύηση ότι τα πακέτα θα λαμβάνονται με την σειρά που στέλνονται και ελέγχει αν υπήρξε κάποιο σφάλμα κατά την αποστολή των δεδομένων. Επίσης, μια σύνδεση TCP εμπλέκει μόνο δύο υπολογιστές και η αποστολή δεδομένων μεταξύ αυτών των δύο υπολογιστών είναι αμφίδρομη.

4.1.4 Υποδοχές ροής

Γνωρίζοντας ότι οι πελάτες και οι εξυπηρετητές μπορεί να βρίσκονται σε διαφορετικές γεωγραφικές τοποθεσίες, ο μόνος τρόπος επικοινωνίας μεταξύ αυτών των διεργασιών είναι μέσω ενός δικτύου. Χρησιμοποιώντας υποδοχές ροής μπορούμε να πετύχουμε επικοινωνία μεταξύ των διαφόρων διεργασιών. Επίσης, οι υποδοχές ροής προσφέρουν αξιοπιστία στην επικοινωνία και εγγυόνται ότι θα φτάσουν όλα τα μηνύματα με τη σωστή σειρά.

Όπως είδαμε και πιο πάνω στο Σχήμα 4.1 το σύστημα μας αποτελείται από 4 είδη διεργασιών: πελάτη, εξυπηρετητή ευρετηρίου, εξυπηρετητή αρχείων και διαχειριστή αρχείων (ή διαχειριστή πιστοποίησης). Όπως αναφέραμε και πιο πριν θα ακολουθήσουμε τον μοντέλο πελάτη-εξυπηρετητή. Άρα, όλοι οι εξυπηρετητές πρέπει να δημιουργήσουν από μια υποδοχή έτσι ώστε να μπορούν να λαμβάνουν μηνύματα από τους πελάτες. Ακόμη, οι πελάτες για να μπορέσουν να στείλουν μηνύματα στους εξυπηρετητές πρέπει να δημιουργήσουν μια υποδοχή ροής, να συνδεθούν με τους εξυπηρετητές και μετά να στέλνουν τα μηνύματα τους σε αυτούς. Όμως, ο πελάτης πρέπει να ξέρει από πριν την διεύθυνση των εξυπηρετητών έτσι ώστε να μπορεί να συνδεθεί με αυτή, γι’ αυτό τον λόγο οι διευθύνσεις των εξυπηρετητών είναι συγκεκριμένες και σταθερές. Στο Σχήμα 4.2 παρουσιάζεται η διαδικασία σύνδεσης και επικοινωνίας μεταξύ ενός πελάτη και εξυπηρετητή. Τέλος, για την υλοποίηση υποδοχών ροής χρησιμοποιήσαμε την βιβλιοθήκη “winsock2.h”, όπου είναι αποκλειστικά για Windows.



Σχήμα 4.2

Τώρα θα αναφέρουμε τις συναρτήσεις που χρειάζονται για την δημιουργία υποδοχών ροής μαζί με μια μικρή περιγραφή για την κάθε μια:

- socket (int domain, int type, int protocol)

  Για να δημιουργήσουμε ένα περιγραφέα υποδοχής χρησιμοποιούμε την συνάρτηση socket η οποία θα μας επιστρέψει το περιγραφέα υποδοχής που δημιούρησε. Οι παράμετροι που δέχεται η συνάρτηση socket είναι η δομή της διεύθυνσης (domain), το είδος της υποδοχής (type), και το πρωτόκολλο (protocol). Όπως βλέπουμε και πιο πάνω στο σχήμα αυτή την συνάρτηση θα την καλέσει και ο πελάτης αλλά και ο εξυπηρετητής. Σε περίπτωση αποτυχίας επιστρέφει -1.

- bind (int s,const struct sockaddr * addr,int namelen)

Η συνάρτηση bind θα συσχετίσει την υποδοχή s με κάποια διεύθυνση addr. Η μεταβλητή s είναι ο περιγραφέας υποδοχής που επέστρεψε η συνάρτηση socket. Όπως αναφέραμε και πιο πριν, για τις διεργασίες των εξυπηρετητών θα καθορίσουμε εμείς συγκεκριμένες διευθύνσεις στις οποίες θα καταφθάνουν οι αιτήσεις από τους πελάτες. Ενώ, για τη διεργασία του πελάτη θα αφήσουμε το σύστημα να αποφασίσει την διεύθυνση που θα του συσχετίσει έτσι δεν χρειάζεται να καλέσουμε αυτή την συνάρτηση στον πελάτη. Σε περίπτωση επιτυχίας επιστρέφει 0, διαφορετικά επιστρέφει -1.

- listen (int s, int max_req)
  Μετά από τις συναρτήσεις socket και bind οι εξυπηρετητές θα πρέπει να καλέσουν την συνάρτηση listen έτσι ώστε να μπορέσουν να λαμβάνουν μηνύματα από τους πελάτες. Ο κάθε εξυπηρετητής θα λαμβάνει μηνύματα από την δική του υποδοχή s. Επίσης, η μεταβλητή max_req καθορίζει το μέγιστο αριθμό αιτήσεων που μπορούν να περιμένουν στην ουρά για εξυπηρέτηση. Στην περίπτωση που ο αριθμός των αιτήσεων ξεπεράσει το max_req, τότε οι αιτήσεις που καταφθάνουν απορρίπτονται, μέχρι να εξυπηρετηθούν κάποιες από την ουρά έτσι ώστε να δημιουργηθεί χώρος. Σε περίπτωση επιτυχίας επιστρέφει 0, διαφορετικά -1.

- accept (int s, const struct sockaddr *addr, socklen_t len);
  Μετά από τις συναρτήσεις socket, bind και listen οι εξυπηρετητές πρέπει να καλέσουν την συνάρτηση accept η οποία αναλαμβάνει να δέχεται αιτήσεις από τους πελάτες. Η μεταβλητή s είναι η υποδοχή του εξυπηρετητή. Οι μεταβλητές addr και len αφοράνε στοιχεία του πελάτη. Σε περίπτωση επιτυχίας, η συνάρτηση επιστρέφει το νέο περιγραφέα υποδοχής της σύνδεσης, διαφορετικά επιστρέφει -1.

- connect (int s, const struct sockaddr *addr, socklen_t len)
  Μετά από τις συναρτήσεις socket και bind οι πελάτες πρέπει να καλέσουν την συνάρτηση connect για να εγκαθιδρύσουν σύνδεση μεταξύ της υποδοχής τους και της υποδοχής του εξυπηρετητή. Η μεταβλητή s είναι η υποδοχή του εξυπηρετητή στην οποία θα συνδεθεί ο πελάτης. Και η μεταβλητή addr είναι η διεύθυνση του εξυπηρετητή στην οποία θα συνδεθεί ο πελάτης. Σε περίπτωση επιτυχίας επιστρέφει 0, διαφορετικά επιστρέφει -1.

- send(int s, const void *buf, size_t b, int flag)/recv(int s, void *buf, size_t b, int flag)
  Αφού ολοκληρωθεί η διαδικασία της σύνδεσης μεταξύ πελάτη-εξυπηρετητή ακολουθεί η επικοινωνία μεταξύ των δύο. Η επικοινωνία επιτυγχάνεται με τις συναρτήσεις send και recv αφού επιτρέπουν την αποστολή και παραλαβή δεδομένων μέσω των υποδοχών ροής. Η συνάρτηση send στέλνει b bytes από την συμβολοσειρά buf στην υποδοχή s. Η συνάρτηση recv διαβάζει b bytes από την υποδοχή s και τα αποθηκεύει στην συμβολοσειρά buf. Και οι δύο συναρτήσεις επιστρέφουν τον πραγματικό αριθμό δεδομένων που έχουν σταλεί/διαβαστεί, διαφορετικά επιστρέφουν -1.

4.2 Διεπαφή

Για την δημιουργία της διεπαφής χρησιμοποιήσαμε το εργαλείο Qt και συγκεκριμένα το Qt Creator 4.1.0 [9]. Ο χρήστης ξεκινώντας το σύστημα το μόνο μέρος του συστήματος με το οποίο αλληλοεπιδρά είναι η διεπαφή. Ο κάθε χρήστης έχει τον δικό του μοναδικό πελάτη με τον οποίο επικοινωνεί. Συγκεκριμένα, ο χρήστης αλληλοεπιδρώντας με την διεπαφή και ανάλογα με τις επιλογές που κάνει στέλνει μηνύματα στον πελάτη και αυτός με την σειρά του αναλαμβάνει να στείλει τα κατάλληλα μνήματα στους εξυπηρετητές. Ακολούθως, ο πελάτης ενημερώνει τη διεπαφή για τα αποτελέσματα της εντολής που έδωσε ο χρήστης και εμφανίζονται στην οθόνη μέσω της διεπαφής.

Όταν ο χρήστης ξεκινάει την διεπαφή τότε αυτή θα πρέπει να συνδεθεί πρώτα με τον πελάτη. Ακολούθως ανάλογα με τις εντολές του χρήστη η διεπαφή μπορεί να στέλνει τα εξής μηνύματα στον πελάτη:

- **signup <*username*> <*password*>**
  Ο χρήστης ζητά να εγγραφεί στο σύστημα χρησιμοποιώντας ως όνομα το <*username*> και ως κωδικό το <*password*>. Αν το <*username*> υπάρχει ήδη τότε ζητά από τον χρήστη να δώσει ένα διαφορετικό όνομα.

- **login <*username*> <*password*>**
  Ο χρήστης ζητά να εισέλθει στο σύστημα δίνοντας ως όνομα το <*username*> και ως κωδικό το <*password*>. Αν το <*username*> ή/και το <*password*> είναι λάθος τότε ενημερώνεται ο χρήστης και ακολούθως του ζητά να γράψει σωστά τα στοιχεία του.

- **list**

  Ανανεώνεται η λίστα αρχείων που βλέπει ο χρήστης μέσω της διεπαφής. Αυτή η λίστα είναι ορατή από τον χρήστη συνεχώς και περιλαμβάνει όλα τα αρχεία του συστήματος. Αν κάποιος άλλος χρήστης ανεβάσει ένα νέο αρχείο τότε για να εμφανιστεί το συγκεκριμένο αρχείο στην λίστα αρχείων που βλέπει ο χρήστης πρέπει να στείλει αυτό το μήνυμα στον πελάτη.

- **rename <*previous name*> <*new name*>**

  Ο χρήστης ζητά από το σύστημα να αλλάξει το όνομα του αρχείου <*previous name*> σε <*new name*>.

- **delete <*filename*>**

  Ο χρήστης ζητά από το σύστημα να διαγραφεί το αρχείο με το όνομα <*filename*>.

- **write <*filename*> <*permission*>**

  Ο χρήστης επιθυμεί να ανεβάσει ένα νέο αρχείο στο σύστημα με το όνομα <*filename*>. Το <*permission*> καθορίζει τι εξουσιοδότηση θα έχουν οι άλλοι χρήστες στο αρχείο αυτό. Αν το <*permission*> είναι 1 τότε όλοι οι χρήστες θα μπορούν να διαβάσουν και να γράψουν στο αρχείο, αν το <*permission*> είναι 2 μόνο ο owner θα μπορεί να γράψει στο αρχείο ενώ οι υπόλοιποι χρήστες θα μπορούν μόνο να το διαβάσουν. Σε περίπτωση που υπάρχει ήδη αρχείο με το όνομα <*filename*> τότε δεν υπάρχει το <*permission*> και ο χρήστης επιθυμεί να «ανανεώσει» το αρχείο <*filename*>.

- **read <*filename*>**

  Ο χρήστης επιθυμεί να κατεβάσει το αρχείο με το όνομα <*filename*>. Όταν ολοκληρωθεί το κατέβασμα, το αρχείο θα βρίσκεται στον υπολογιστή του χρήστη και συγκεκριμένα στον προσωπικό του φάκελο.

Σημείωση: Για κάθε χρήστη υπάρχει ένας προσωπικός φάκελος όπου δημιουργείται κατά την διάρκεια της εγγραφής του και εκεί βρίσκονται τα αρχεία που διατηρεί ο κάθε χρήστης για το σύστημα.

4.3 Πελάτης

Ο πελάτης ξεκινάει την ίδια ώρα που ξεκινάει και η διεπαφή και εγκαθιδρύεται σύνδεση μεταξύ των δύο. Επίσης, ο πελάτης θα πρέπει να εγκαθιδρύσει σύνδεση και με όλες τους εξυπηρετητές ευρετηρίου και αρχείων και μετά με τον διαχειριστή πιστοποίησης. Κατά την

εκκίνηση του ο πελάτης θα διαβάσει το αρχείο “config.txt” όπου εκεί αναγράφονται όλες οι διευθύνσεις των εξυπηρετητών και του διαχειριστή πιστοποίησης με τις οποίες πρέπει να εγκαθίδρυση σύνδεση. Αυτό μπορούμε να το κάνουμε αφού οι διευθύνσεις των εξυπηρετητών και του διαχειριστή πιστοποίησης είναι προκαθορισμένες και σταθερές για τους λόγους που αναφέραμε και πιο πριν. Τέλος, η σύνδεση του πελάτη με την διεπαφή, τους εξυπηρετητές και τον διαχειριστή πιστοποίησης θα πραγματοποιηθεί μόνο μια φορά, και συγκεκριμένα κατά την διάρκεια έναρξης του. Η αποσύνδεση θα πραγματοποιηθεί μόνο στη περίπτωση που ο χρήστης εγκαταλείψει το σύστημα.

Τώρα θα αναφέρουμε τις ενέργειες που εκτελεί ο πελάτης στις εξής περιπτώσεις:

- **Εγγραφή χρήστη**

  Στη περίπτωση που ο πελάτης λάβει το μήνυμα signup τότε στέλνει μήνυμα στον διαχειριστή πιστοποίησης και ζητά να εγγράψει τον χρήστη με τα στοιχεία που του έδωσε. Ακολούθως λαμβάνει απάντηση από τον διαχειριστή πιστοποίησης και ενημερώνει την διεπαφή με το ανάλογο μήνυμα.

- **Είσοδος χρήστη**

  Στη περίπτωση που ο πελάτης λάβει το μήνυμα login τότε στέλνει μήνυμα στον διαχειριστή πιστοποίησης και ζητά να ελέγξει αν τα στοιχεία που έδωσε ο χρήστης είναι σωστά. Ακολούθως λαμβάνει απάντηση από τον διαχειριστή πιστοποίησης και ενημερώνει την διεπαφή με το ανάλογο μήνυμα.

- **Ανανέωση λίστας αρχείων**

  Αν λάβει το μήνυμα list τότε στέλνει ένα μήνυμα σε όλους τους εξυπηρετητές ευρετηρίου όπου ζητάει να του αποστείλουν την λίστα αρχείων και το ιστορικό που έχουν αποθηκευμένα. Αφού λάβει μια πλειοψηφία απαντήσεων από τους εξυπηρετητές ευρετηρίου υπολογίζει την τελική λίστα αρχείων και το ιστορικό, και ακολούθως τα στέλνει στην διεπαφή. Στο ιστορικό του συστήματος αναγράφονται είναι οι τελευταίες ενέργειες που έχουν πραγματοποιηθεί από τους χρήστες του συστήματος. Για να συμπεριληφθεί ένα αρχείο στην λίστα πρέπει να υπάρχει σε τουλάχιστον f+1 λίστες. Ο λόγος που περιμένουμε πλειοψηφία απαντήσεων και ελέγχουμε πόσες φορές εμφανίζεται ένα αρχείο στις λίστες είναι για να είμαστε βέβαιοι πως η τελική λίστα αρχείων που θα εμφανιστεί στον χρήστη είναι σωστή.

- **Μετονομασία αρχείου**

Αν λάβει το μήνυμα rename τότε ο πελάτης πρώτα ελέγχει αν ο χρήστης που ζητά να μετονομαστεί το αρχείο είναι ο owner του, αν δεν είναι τότε απορρίπτεται το αίτημα του, αλλιώς στέλνει μήνυμα σε όλους τους εξυπηρετητές αρχείων και σε όλους τους εξυπηρετητές ευρετηρίου για να τους ζητήσει να μετονομάσουν το αρχείο στο νέο όνομα που έδωσε ο χρήστης.

- **Διαγραφή αρχείου**

Αν λάβει το μήνυμα delete τότε ο πελάτης πρώτα ελέγχει αν ο χρήστης που ζητά να διαγραφεί το αρχείο είναι ο owner του, αν δεν είναι τότε απορρίπτεται το αίτημα του, αλλιώς στέλνει μήνυμα σε όλους τους εξυπηρετητές ευρετηρίου για να τους ζητήσει να διαγράψουν το αρχείο από την λίστα αρχείων. Επίσης, στέλνει μήνυμα και στους εξυπηρετητές αρχείων για να διαγράψουν το αρχείο από τον αποθηκευτικό τους χώρο. Τέλος, αν ολοκληρωθεί η διαδικασία χωρίς να παρουσιαστεί κάποιο σφάλμα τότε στην λίστα αρχείων που βλέπει ο χρήστης δεν θα υπάρχει πλέον το αρχείο που διαγράφθηκε.

- **Γράψιμο αρχείου**

Αν ο πελάτης λάβει το μήνυμα write τότε πρέπει πρώτα να εντοπίσει το αναγνωριστικό του αρχείου που επιθυμεί να γράψει ο χρήστης. Αν πρόκειται για νέο αρχείο τότε ο πελάτης δημιουργεί ένα νέο μοναδικό αναγνωριστικό αλλιώς εντοπίζει το αναγνωριστικό του αρχείο κάνοντας μια αναζήτηση στην λίστα αρχείων που έχει αποθηκευμένη. Μετά, επικοινωνεί με όλους τους εξυπηρετητές ευρετηρίου και περιμένει μια πλειοψηφία από απαντήσεις. Ο λόγος που περιμένει μια πλειοψηφία από απαντήσεις είναι για να είμαστε σίγουροι ότι θα εντοπίσει το πιο πρόσφατο tag. Όταν λάβει τα μηνύματα επιλέγει το μεγαλύτερο tag και το αυξάνει κατά ένα, και ακολούθως επικοινωνεί με τους εξυπηρετητές αρχείων όπου τους στέλνει το νέο tag και γράφει το αρχείο σε αυτούς. Όταν λάβει απάντηση από f+1 εξυπηρετητές αρχείων τότε ενημερώνει τους εξυπηρετητές ευρετηρίου για το νέο tag του αρχείου και ποιοι εξυπηρετητές αρχείων έχουν το πιο πρόσφατο αρχείο. Μετά περιμένει από μια πλειοψηφία εξυπηρετητών ευρετηρίου να του απαντήσουν. Μόλις λάβει την πλειοψηφία στέλνει μήνυμα στους εξυπηρετητές αρχείων και τους ζητά να διαγράψουν όλα τα αρχεία που δεν έχουν το πιο πρόσφατο tag.

- **Διάβασμα αρχείου**

Αν ο πελάτης λάβει το μήνυμα read τότε πρέπει πρώτα να εντοπίσει το αναγνωριστικό του αρχείου που επιθυμεί να διαβάσει ο χρήστης. Αρχικά, στέλνει μήνυμα σε όλους τους εξυπηρετητές ευρετηρίου και περιμένει μια πλειοψηφία από απαντήσεις. Ο λόγος που περιμένει πλειοψηφία από απαντήσεις είναι για να είμαστε σίγουροι πως σε τουλάχιστον μια απάντηση υπάρχει το πιο πρόσφατο tag. Μετά στέλνει μήνυμα σε όλους τους εξυπηρετητές ευρετηρίου για να τους ενημερώσει πιο είναι το πιο πρόσφατο tag του αρχείο που έχει εντοπίσει. Όταν λάβει μια πλειοψηφία απαντήσεων από τους εξυπηρετητές ευρετηρίου, θα στείλει μήνυμα σε ένα από τους διαθέσιμους εξυπηρετητές αρχείων που έχουν το πιο πρόσφατο αρχείο και θα του ζητήσει να του αποστείλει το αρχείο.

4.4 Εξυπηρετητής ευρετηρίου

Ο εξυπηρετητής ευρετηρίου όπως αναφέραμε και στο Κεφάλαιο 3 αποθηκεύει τα μεταδεδόμενα των αρχείων. Όλες τις πληροφορίες που χρειάζεται τις έχει αποθηκευμένες σε μορφή πίνακα. Ακόμη, έχει αποθηκευμένη την λίστα αρχείων του συστήματος. Κατά την εκκίνηση του ο εξυπηρετητής ευρετηρίου πρέπει να καθορίσει την διεύθυνση που θα δέχεται αιτήσεις από τους πελάτες. Επίσης, για να εξυπηρετήσει αρκετούς πελάτες ταυτόχρονα κάνει χρήση νημάτων.

Τώρα θα αναφέρουμε τις ενέργειες που εκτελεί ο εξυπηρετητής ευρετηρίου όταν λάβει τα εξής μηνύματα από τον πελάτη:

- **wread <*fileID*> <*filename*> <*permission*> <*username*> <*clientID*>**

  Πρώτα εντοπίζει την θέση του αρχείου <*filename*> στον πίνακα, αν το αρχείο είναι νέο τότε καταχωρείται στον πίνακα με τα στοιχεία που υπάρχουν στο μήνυμα, αλλιώς ελέγχει αν ο χρήστης έχει δικαίωμα να γράψει στο ήδη υπάρχων αρχείο. Αν δεν έχει δικαίωμα γραφής τότε ενημερώνεται με το ανάλογο μήνυμα ο πελάτης και σταματάει η διαδικασία. Ενώ, αν ο χρήστης έχει δικαίωμα γραφής τότε στέλνει στον πελάτη ποιοι εξυπηρετητές αρχείων έχουν το αρχείο μαζί με το tag.

- **wwrite <*replicaSet*> <*tag*> <*filename*>**

  Αρχικά εντοπίζει την θέση του αρχείου <*filename*> στον πίνακα και μετά αποθηκεύει το νέο <*tag*>. Επίσης, διαγράφει το σετ των εξυπηρετητών αρχείων που έχει αποθηκευμένο και προσθέτει το <*replicaSet*> στην θέση του.

- **rread <*fileID*> <*filename*> <*username*> <*clientID*>**

Πρώτα εντοπίζει την θέση του αρχείου <*filename*> στον πίνακα, αν το αρχείο είχε διαγραφεί πριν σταλεί το μήνυμα τότε ενημερώνει τον χρήστη πως έχει διαγραφεί το αρχείο και πλέον δεν είναι διαθέσιμο. Αλλιώς, εντοπίζει όλους τους εξυπηρετητές αρχείων που έχουν το συγκεκριμένο αρχείο και τους στέλνει στον πελάτη.

- **rwrite <*replicaSet*> <*tag*> <*filename*>**

Αρχικά εντοπίζει την θέση του αρχείου <*filename*> στον πίνακα και μετά αποθηκεύει το νέο <*tag*>. Επίσης, διαγράφει το σετ των εξυπηρετητών αρχείων που έχει αποθηκευμένο και προσθέτει το <*replicaSet*> στην θέση του.

- **done <*filename*>**

Αν λάβει αυτό το μήνυμα σημαίνει πως το ανέβασμα του αρχείου <*filename*> έχει ολοκληρωθεί επιτυχώς και άρα μπορεί να το προσθέσει στην λίστα αρχείων που έχει αποθηκευμένη.

- **reqfilelist**

Ο εξυπηρετητής ευρετηρίου ενημερώνει τον πελάτη με την λίστα αρχείων που έχει αποθηκευμένη μαζί με τα στοιχεία τους όπως το αναγνωριστικό, την ημερομηνία τελευταίας επεξεργασίας, το μέγεθος και τα δικαιώματα που έχει για το κάθε αρχείο. Επίσης, μαζί με την λίστα αρχείων στέλνεται και το ιστορικό του συστήματος.

- **rename <*previous name*> <*new name*> <*username*>**

Πρώτα ελέγχει αν ο χρήστης με το όνομα <*username*> είναι ο owner του αρχείου <*previous name*>, αν δεν είναι τότε απορρίπτεται το αίτημα του. Αλλιώς αλλάζει το όνομα του αρχείου σε <*new name*> στον πίνακα αλλά και στην λίστα αρχείων που έχει αποθηκευμένη.

- **delete <*filename*>**

Διαγράφει το αρχείο <*filename*> από τον πίνακα. Επίσης, το αρχείο διαγράφεται και από την λίστα αρχείων.

4.5 Εξυπηρετητής αρχείων

Ο εξυπηρετητής αρχείων όπως αναφέραμε και στο Κεφάλαιο 3 αποθηκεύει πολλαπλές εκδόσεις των αρχείων. Κατά την εκκίνηση του ο εξυπηρετητής αρχείων πρέπει να καθορίσει την διεύθυνση που θα δέχεται αιτήσεις από τους πελάτες. Επίσης, για να εξυπηρετήσει αρκετούς πελάτες ταυτόχρονα κάνει χρήση νημάτων.

Τώρα θα αναφέρουμε τις ενέργειες που εκτελεί ο εξυπηρετητής αρχείων όταν λάβει τα εξής μηνύματα από τον πελάτη:

- **write <*fileID*> <*filename*> <*clientID*> <*tag*> <*filesize*>**

  Πρώτα δημιουργείται ένα νέο αρχείο στον αποθηκευτικό του χώρο με το όνομα <*filename*> και μετά ξεκινά να λαμβάνει δεδομένα από τον πελάτη και τα γράφει στο αρχείο που έχει δημιουργήσει. Όταν τελειώσει με την παραλαβή των δεομένων τότε ελέγχει το checksum του αρχείου που δημιούργησε με αυτό που έχει ο πελάτης. Αν τα δύο checksum είναι διαφορετικά τότε διαγράφει το αρχείο που δημιούργησε και ενημερώνει ανάλογα τον πελάτη. Αλλιώς, ενημερώνει τον πελάτη πως η εγγραφή έχει ολοκληρωθεί επιτυχώς.

- **read <*fileID*> <*filename*> <*clientID*> <*tag*>**

  Αρχικά εντοπίζει το μεγαλύτερο tag του αρχείου <*filename*>. Ακολούθως, ξεκινά να στέλνει στον πελάτη τα δεδομένα του αρχείου που έχει το πιο μεγάλο tag. Όταν τελειώσει με την αποστολή του αρχείου, στέλνει στον πελάτη το checksum του αρχείου που μόλις έστειλε.

- **secure <*fileID*> <*filename*> <*clientID*> <*tag*>**

  Αρχικά κάνει το secure bit του αρχείου <*filename*> να ισούται με 1. Ακολούθως, διαγράφει όλες τις παλαιότερες εκδόσεις του αρχείου και συγκεκριμένα είναι τα αρχεία που έχουν tag μικρότερο από το <*tag*>.

- **rename <*previous name*> <*new name*>**

  Εντοπίζει το αρχείο <*previous name*> και αλλάζει το όνομα του σε <*new name*>.

4.6 Διαχειριστής αρχείων και διαχειριστής πιστοποίησης

Στο σύστημα μας έχουμε μόνο ένα διαχειριστή αρχείων. Κατά την εκκίνηση του ο διαχειριστή αρχείων πρέπει να καθορίσει την διεύθυνση που θα δέχεται αιτήσεις από τους πελάτες. Μία από τις υποχρεώσεις του διαχειριστή αρχείων είναι να δημιουργεί ένα νέο μοναδικό αναγνωριστικό για κάθε καινούργιο αρχείο που προστίθεται στο σύστημα. Επίσης, εγγράφει τους χρήστες του συστήματος και ελέγχει αν τα στοιχεία που δίνουνε για να κάνουν login στο σύστημα είναι σωστά. Ακόμη, αναθέτει ένα μοναδικό αναγνωριστικό για κάθε πελάτη έτσι ώστε βλέποντας το αναγνωριστικό να μπορεί να εντοπίσει ποιος πελάτης είναι. Τέλος, υπολογίζει την λίστα αρχείων και το ιστορικό και όποτε τα ζητάει ο πελάτης του τα στέλνει.

Τώρα θα αναφέρουμε τις ενέργειες που εκτελεί ο διαχειριστής αρχείων όταν λάβει τα εξής μηνύματα από τον πελάτη:

- **done <*filename*>**

  Αν λάβει αυτό το μήνυμα σημαίνει πως το ανέβασμα του αρχείου <*filename*> έχει ολοκληρωθεί επιτυχώς και άρα μπορεί να το προσθέσει στην λίστα αρχείων που έχει αποθηκευμένη.

- **rename <*previous name*> <*new name*>**

  Εντοπίζει το αρχείο <*previous name*> και αλλάζει το όνομα του σε <*new name*>.

- **delete <*filename*>**

  Εντοπίζει το αρχείο <*filename*> και το διαγράφει από την λίστα αρχείων.

- **reqclientid <*username*> <*password*>**

  Δημιουργεί ένα μοναδικό αναγνωριστικό για τον νέο πελάτη και το αποθηκεύει μαζί με το <*username*> και το <*password*>. Ακολούθως ενημερώνει τον πελάτη με το αναγνωριστικό που του έχει αναθέσει. Αν το <*username*> είναι ήδη καταχωρημένο τότε ενημερώνει τον πελάτη πως το συγκεκριμένο όνομα δεν μπορεί να χρησιμοποιηθεί.

- **reqlogin <*username*> <*password*>**

  Ελέγχει αν το <*username*> και το <*password*> είναι σωστά, αν είναι σωστά τότε ενημερώνει τον πελάτη πως ο χρήστης μπορεί να κάνει login, σε αντίθετη περίπτωση ενημερώνει τον πελάτη πως τα στοιχεία που έδωσε είναι λανθασμένα.

- **reqfileid <*filename*> <*owner*>**

Εντοπίζει το αναγνωριστικό για το αρχείο <*filename*> και το επιστρέφει στον πελάτη. Αν δεν εντοπίσει το αρχείο τότε δημιουργείται ένα νέο μοναδικό αναγνωριστικό για το συγκεκριμένο αρχείο και το επιστρέφει στον πελάτη.

- **reqfilelist**

  Ο διαχειριστής αρχείων ενημερώνει τον πελάτη με την λίστα αρχείων που έχει αποθηκευμένη μαζί με τα στοιχεία τους όπως το αναγνωριστικό, την ημερομηνία τελευταίας επεξεργασίας, το μέγεθος και τα δικαιώματα που έχει για το κάθε αρχείο. Επίσης, μαζί με την λίστα αρχείων στέλνεται και το ιστορικό του συστήματος.

Αρχικά ο διαχειριστής αρχείων ήταν κεντρικοποιημένος με αποτέλεσμα σε περίπτωση κατάρρευσης του το σύστημα δεν θα μπορούσε να λειτουργήσει και γι' τον κάναμε κατανεμημένο. Έτσι οι λειτουργίες που εκτελούσε διαμοιράστηκαν με τον εξής τρόπο:
Την λίστα αρχείων την αποθηκεύουν οι εξυπηρετητές ευρετηρίου και αν λάβουν το μήνυμα reqfilelist τότε ο κάθε εξυπηρετητής ευρετηρίου στέλνει την λίστα αρχείων που έχει αποθηκευμένη. Ακολούθως, ο πελάτης περιμένει μια πλειοψηφία απαντήσεων και για να συμπεριλάβει ένα αρχείο στην τελική λίστα πρέπει να υπάρχει σε τουλάχιστον f+1 λίστες από αυτές που έχει λάβει. Επίσης, όταν οι εξυπηρετητές ευρετηρίου λάβουν τα μηνύματα delete και rename πρέπει να κάνουν τις ανάλογες αλλαγές στον πίνακα τους αλλά τώρα πρέπει να αλλάξουν και την λίστα αρχείων που έχουν αποθηκευμένη. Τέλος, και το μήνυμα done στέλνεται στους εξυπηρετητές ευρετηρίου.

Για την εγγραφή των χρηστών και έλεγχο των στοιχείων τους κατά την είσοδο έχει προστεθεί ένας διαχειριστής πιστοποίησης. Ο διαχειριστής πιστοποίησης μπορεί να λάβει τα μηνύματα reqlogin και reqclientid και οι εντολές που εκτελεί όταν λάβει αυτά τα μηνύματα είναι οι ίδιες με αυτές του διαχειριστή αρχείων.

Τέλος, το αναγνωριστικό ενός αρχείου έχει την μορφή <owner, id> όπου το πεδίο owner είναι το αναγνωριστικό του χρήστη που έχει δημιουργήσει το αρχείο και το πεδίο id είναι το αναγνωριστικό του αρχείου το οποίο υπολογίστηκε από τον δημιουργό του αρχείου και είναι μοναδικό μόνο για τον δημιουργό του αρχείο (π.χ. έχουμε 2 χρήστες με τα ονόματα x και y. Ο x έχει 2 αρχεία άρα το ένα αρχείο θα έχει το αναγνωριστικό <x,1> και το άλλο <x,2>. Ο y έχει ένα αρχείο άρα το αρχείο θα έχει το αναγνωριστικό <y,1>). Έτσι, το μήνυμα reqfileid δεν χρειάζεται πλέον.

Σημείωση: Στην επεξήγηση του εξυπηρετητή ευρετηρίου και του πελάτη, συμπεριλαμβάνονται οι αλλαγές που χρειάστηκε να κάνουμε για να μπορέσουμε να κάνουμε τον διαχειριστή αρχείων κατανεμημένο.

Κεφάλαιο 5

Περιγραφή διεπαφής

5.1 Επεξήγηση οθονών	38
5.2 Εγγραφή	40
5.3 Είσοδο	41
5.4 Ανέβασμα αρχείου	42
5.5 Ανανέωση αρχείου	43
5.6 Κατέβασμα αρχείου	45
5.7 Προεπισκόπηση αρχείου	46
5.8 Διαγραφή αρχείου	46
5.9 Μετονομασία αρχείου	48
5.10 Ιστορικό	48

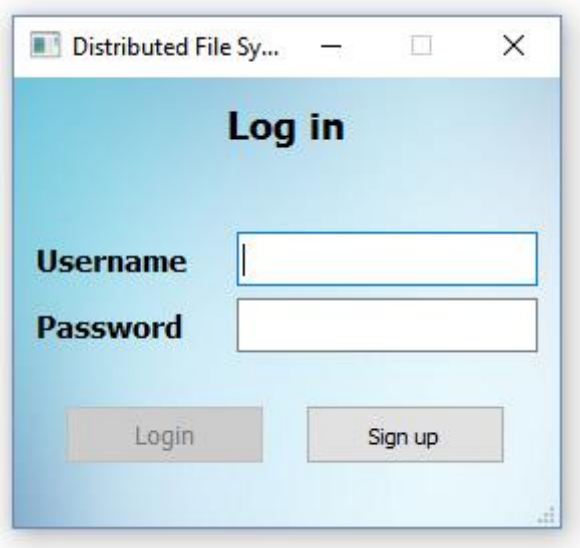
Σε αυτό το κεφάλαιο θα περιγράψουμε την διεπαφή του συστήματος. Συγκεκριμένα, θα επεξηγήσουμε τις οθόνες του συστήματος, τι σημαίνει το κάθε κουμπί που υπάρχει στην διεπαφή και τέλος θα παρουσιάσουμε αναλυτικά τη σωστή ακολουθία των ενεργειών που πρέπει να ακολουθήσουν οι χρήστες για να εκτελέσουν τις λειτουργείες που προσφέρει το σύστημα.

5.1 Επεξήγηση οθονών

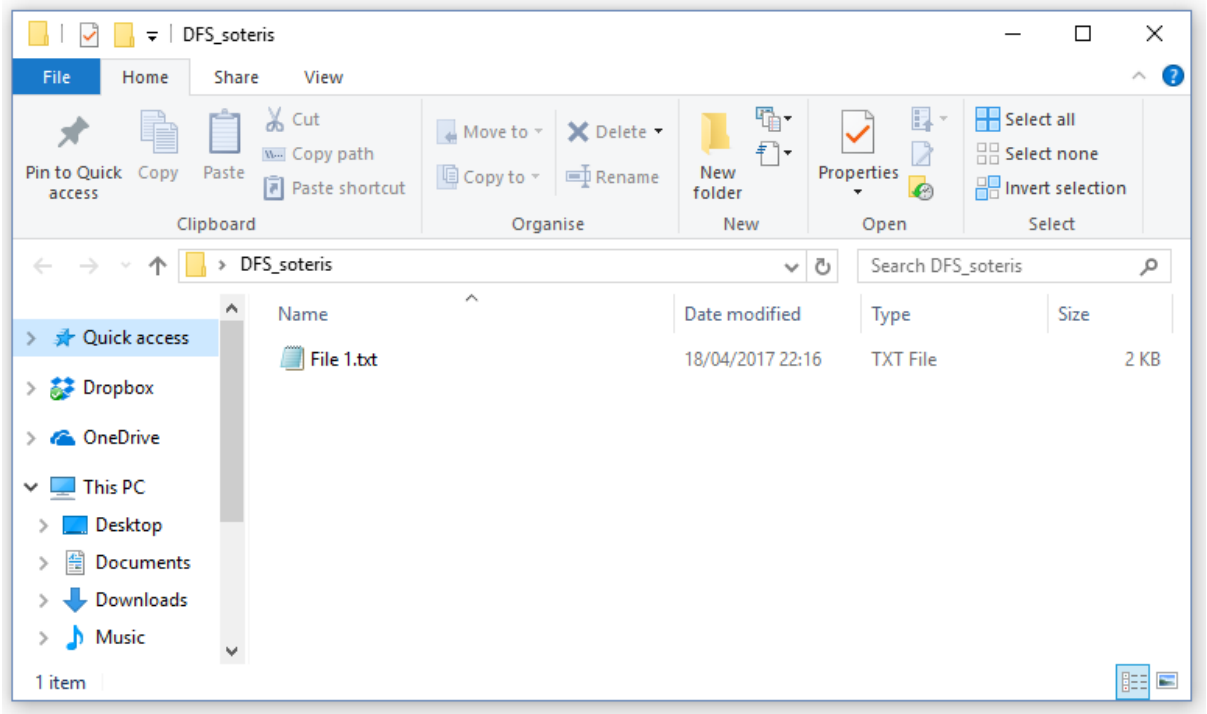
Στην πιο κάτω οθόνη ο χρήστης καλείτε να καταχωρήσει το username και το password που επιθυμεί να έχει. Μετά χρησιμοποιώντας αυτά τα στοιχεία θα μπορεί να έχει πρόσβαση στο σύστημα.



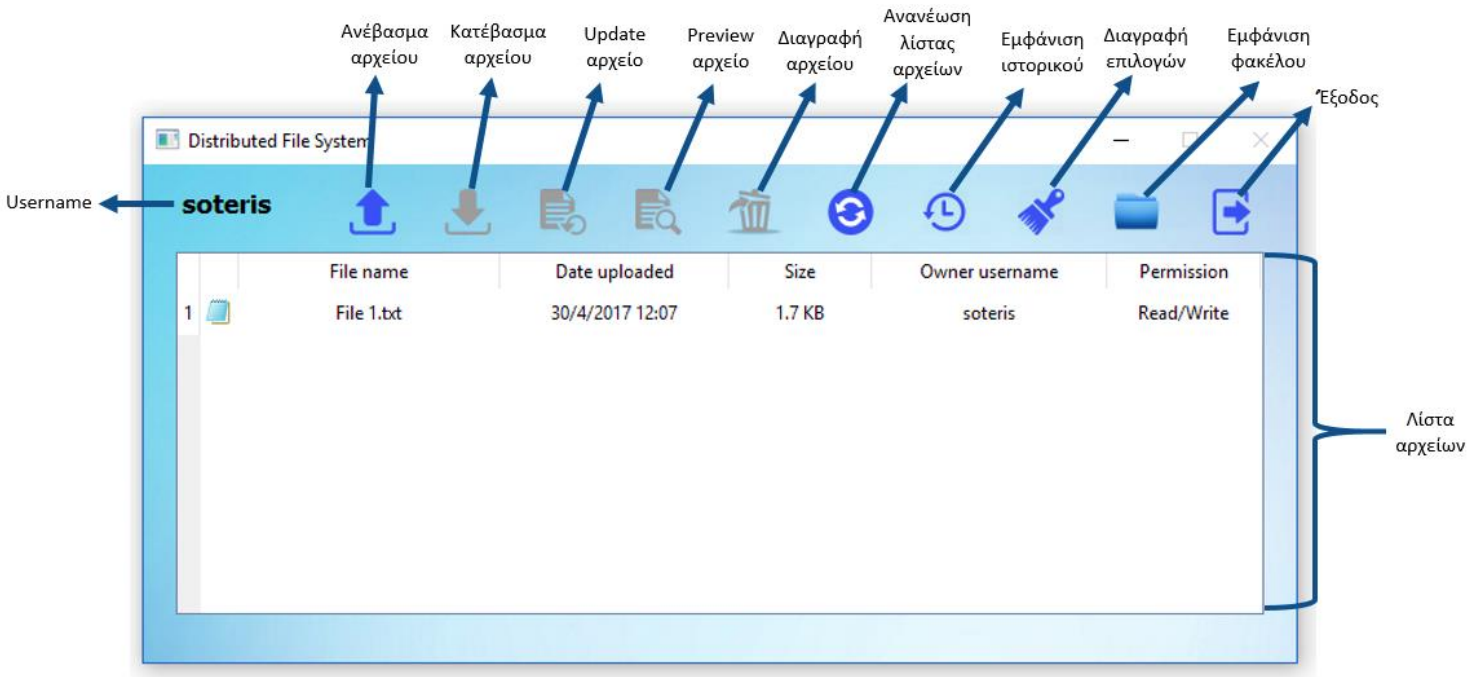
Στην πιο κάτω οθόνη ο χρήστης καλείτε να καταχωρήσει το username του και το password του για να μπορέσει να εισέλθει στο σύστημα.



Μετά την εγγραφή δημιουργείται ένας φάκελος στο προσωπικό υπολογιστή του κάθε χρήστη όπου εκεί θα βρίσκονται τα αρχεία που ανέβασε στο σύστημα και τα αρχεία που κατέβασε από το σύστημα. Στην πιο κάτω οθόνη βλέπουμε ένα παράδειγμα προσωπικού φακέλου όπου ο χρήστης soteris ανέβασε/κατέβασε ένα αρχείο με το όνομα «File 1.txt»

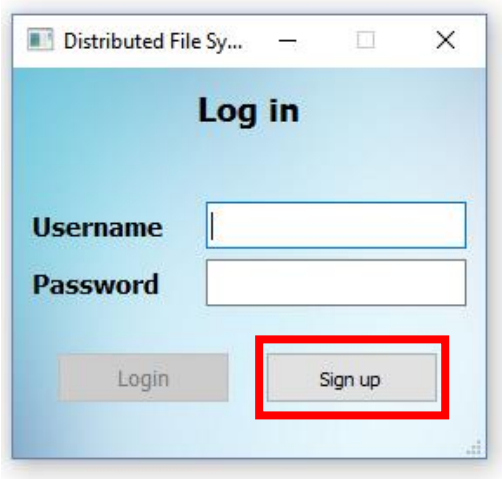


Μετά την επιτυχή είσοδο του χρήστη στο σύστημα εμφανίζεται η πιο κάτω οθόνη

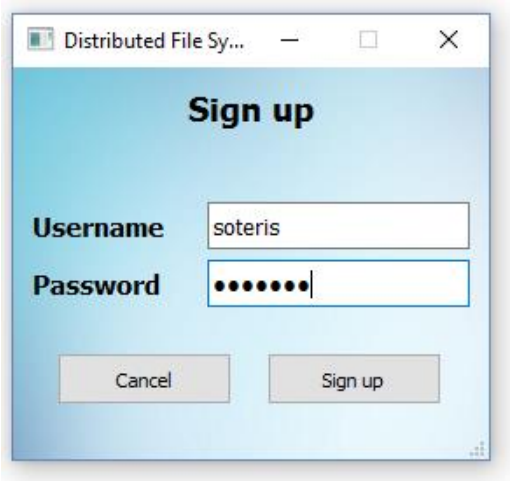


5.2 Εγγραφή

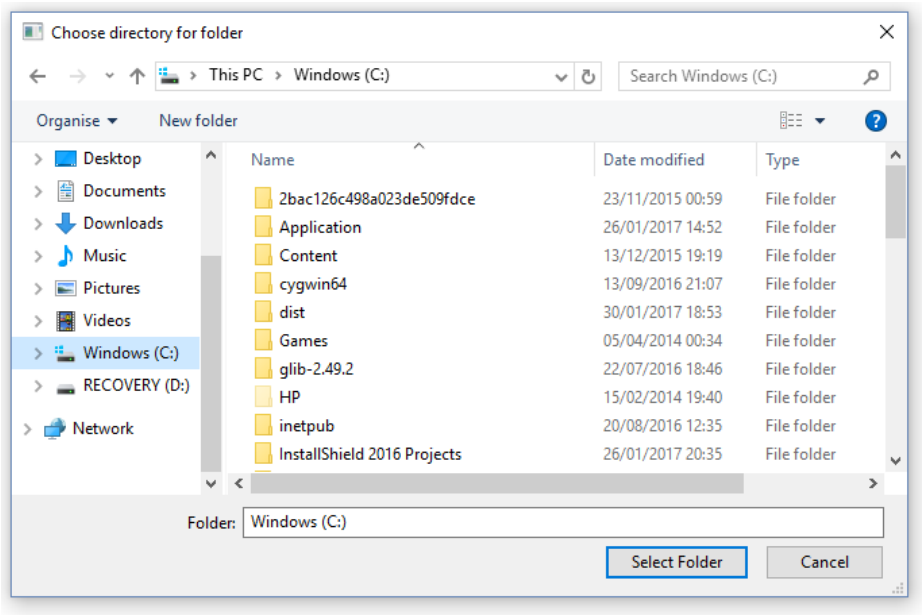
1. Επιλέξτε το κουμπί που αναγράφει «sign up»



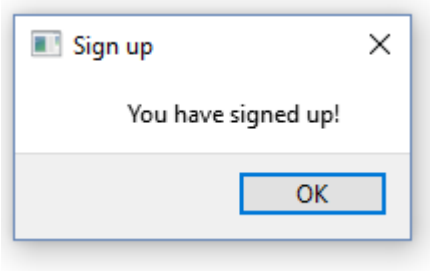
2. Καταχωρήστε το όνομα και κωδικό που επιθυμείτε να έχετε



3. Πατήστε το κουμπί «sign up»
4. Μέσω ενός νέου παραθύρου καλείστε να επιλέξτε το directory που επιθυμείτε να βρίσκεται ο προσωπικό σας φάκελος

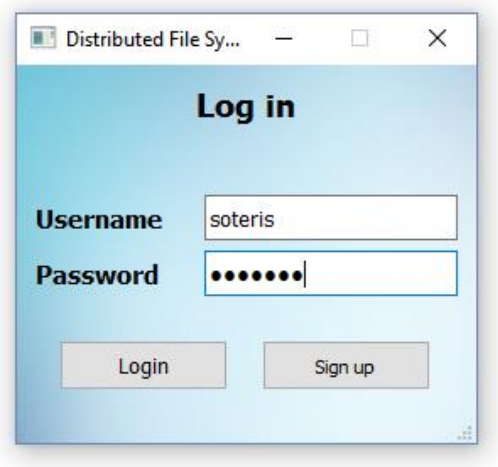


5. Αν εμφανιστεί το πιο κάτω παράθυρο σημαίνει πως η εγγραφή σας έχει ολοκληρωθεί επιτυχώς και έχει δημιουργηθεί ο προσωπικός σας φάκελος



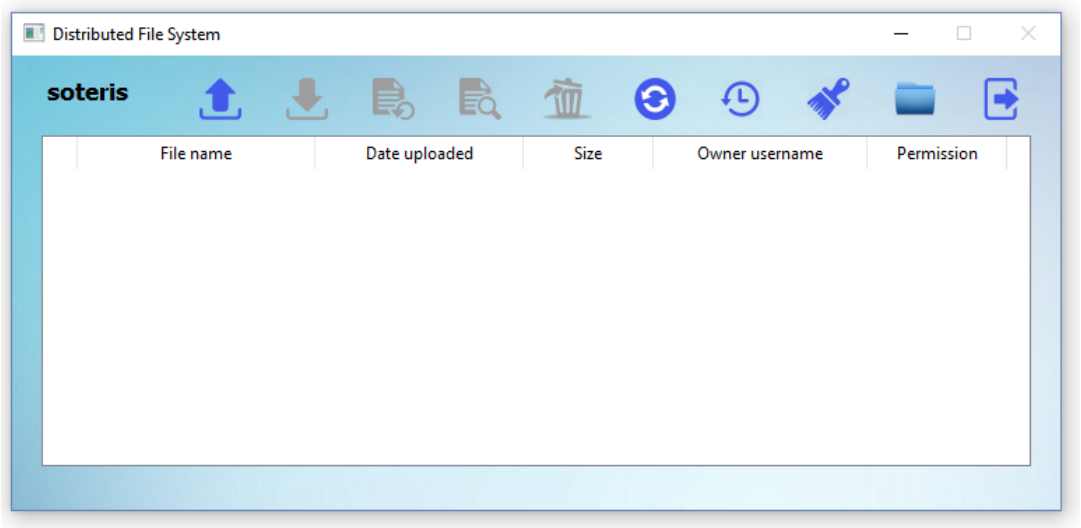
5.3 Είσοδος

1. Καταχωρήστε το όνομα και τον κωδικό σας



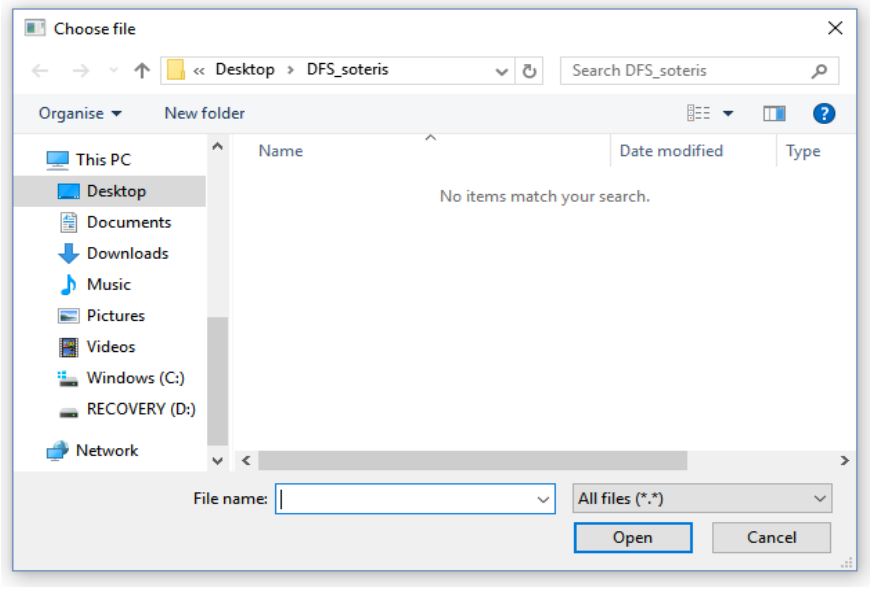
2. Πατήστε το κουμπί «login»

3. Αν τα στοιχεία που καταχωρήσατε στο βήμα 1 είναι σωστά τότε θα εμφανιστεί το πιο κάτω παράθυρο

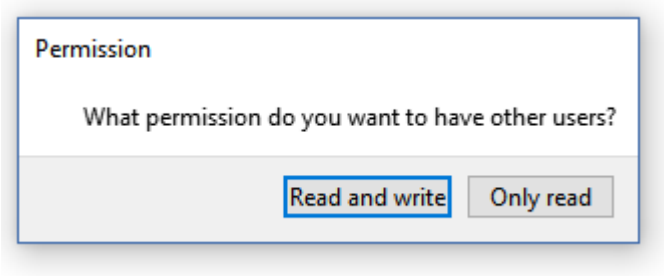


5.4 Ανέβασμα αρχείου

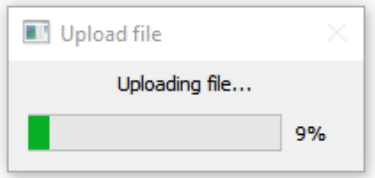
1. Επιλέξτε το κουμπί 
2. Θα εμφανιστεί ένα νέο παράθυρο όπου καλείστε να επιλέξετε το αρχείο που θέλετε να ανεβάσετε. Το παράθυρο θα μοιάζει κάπως έτσι:



3. Επιλέξτε τι δικαιώματα θέλετε να έχουν οι άλλοι χρήστες για το συγκεκριμένο αρχείο που θα ανεβάσετε

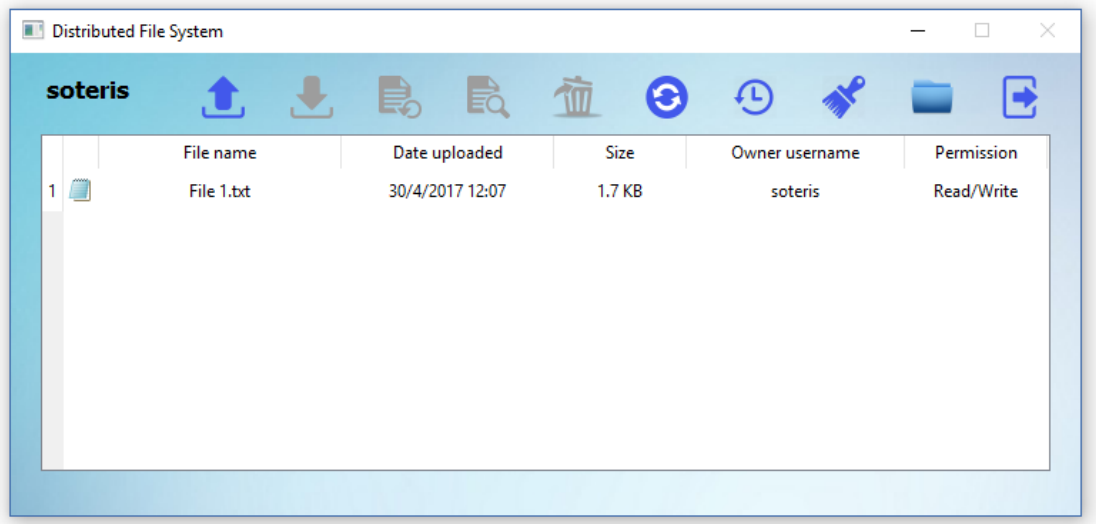


4. Αν το αρχείο είναι μεγάλο τότε η διαδικασία του ανεβάσματος θα φαίνεται από ένα παράθυρο όπως το πιο κάτω



5. Αν δεν προκύψει κάποιο σφάλμα τότε το αρχείο θα προστεθεί αυτόματα στην λίστα αρχείων

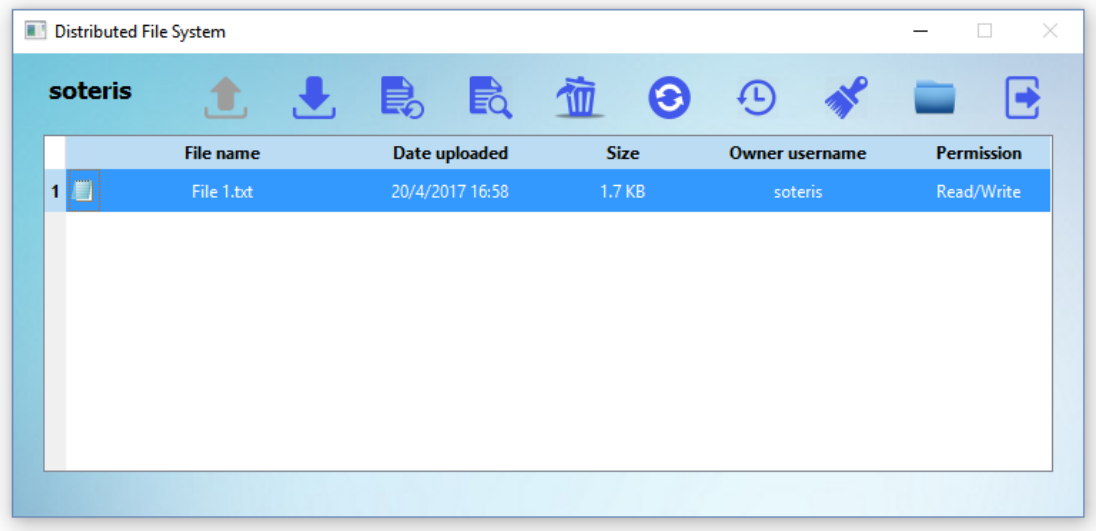
   Στο πιο κάτω παράδειγμα ο χρήστης soteris ανέβασε ένα αρχείο με το όνομα «File 1.txt»



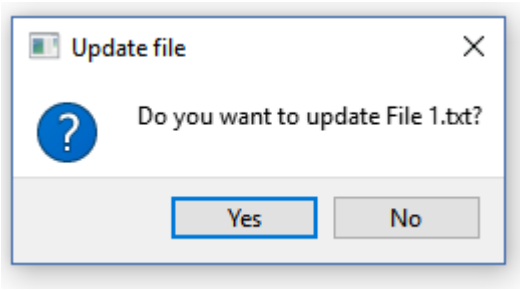
Σημείωση: Επίσης μπορείτε να ανεβάσετε και ένα αρχείο με το να μεταφέρετε στον προσωπικό σας φάκελο. Αφού τοποθέτησε το αρχείο στο φάκελο σας θα συνεχίσετε την ίδια διαδικασία ξεκινώντας από το βήμα 3.

5.5 Ανανέωση αρχείου

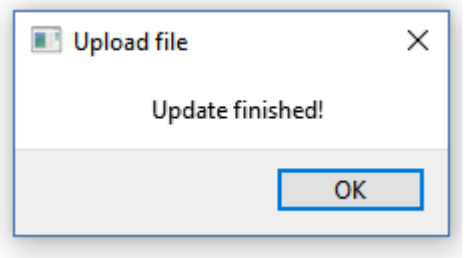
1. Επιλέξτε το αρχείο που θέλετε να ανανεώσετε από την λίστα αρχείων



2. Επιλέξτε το κουμπί 
3. Θα εμφανιστεί παράθυρο και θα σας ρωτάει αν είστε σίγουροι να ανανεώσετε το επιλεγμένο αρχείο



4. Αν επιλέξετε «Yes» στο βήμα 3 και δεν εμφανιστεί κάποιο σφάλμα τότε θα εμφανιστεί το πιο κάτω παράθυρο

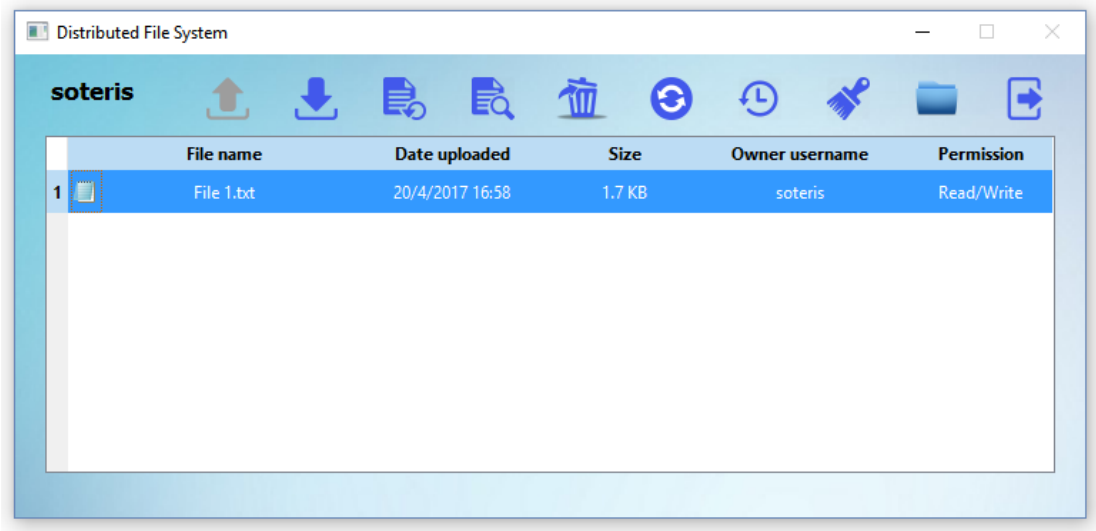


Σημείωση 1: Αν δεν είστε ο owner του αρχείου και τα δικαιώματα που έδωσε ο owner του αρχείου είναι μόνο για διάβασμα τότε δεν θα μπορείτε να ανανεώσετε το συγκεκριμένο αρχείο.

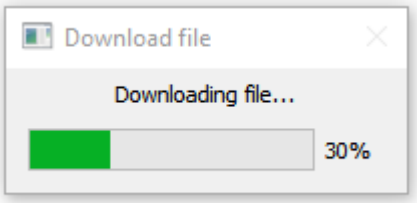
Σημείωση 2: Μπορείτε να ανανεώσετε ένα αρχείο πηγαίνοντας στον προσωπικό σας φάκελο, ανοίξτε το αρχείο που επιθυμείτε, και μετά όταν αποθηκεύσετε τις αλλαγές τότε αυτόματα θα ξεκινήσει η διαδικασία ανανέωσης από το βήμα 3 μόνο για το αρχείο στο οποίο κάνατε τις αλλαγές.

5.6 Κατέβασμα αρχείου

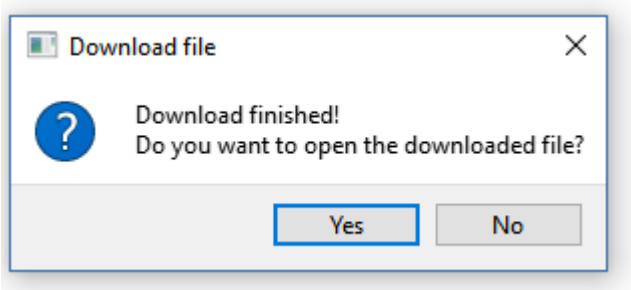
1. Επιλέξτε το αρχείο που θέλετε να κατεβάσετε από την λίστα αρχείων



2. Επιλέξτε το κουμπί 
3. Αν το αρχείο είναι μεγάλο τότε η διαδικασία του κατεβάσματος θα φαίνεται από ένα παράθυρο όπως το πιο κάτω:

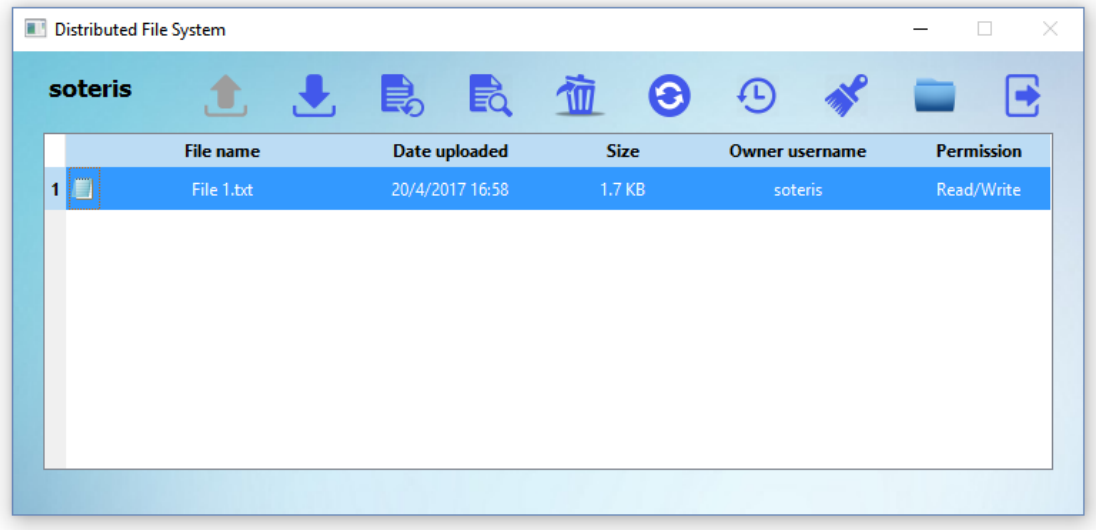


4. Αν δεν υπάρξει κάποιο σφάλμα τότε το αρχείο θα βρίσκεται στον προσωπικό σας φάκελο
5. Θα εμφανιστεί ένα νέο παράθυρο όπου καλείστε να επιλέξετε αν θέλετε να ανοίξετε το αρχείο που μόλις κατέβηκε.

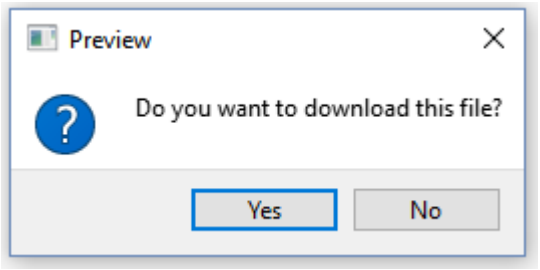


5.7 Προεπισκόπηση αρχείου

1. Επιλέξτε το αρχείο από την λίστα αρχείων που επιθυμείτε να δείτε

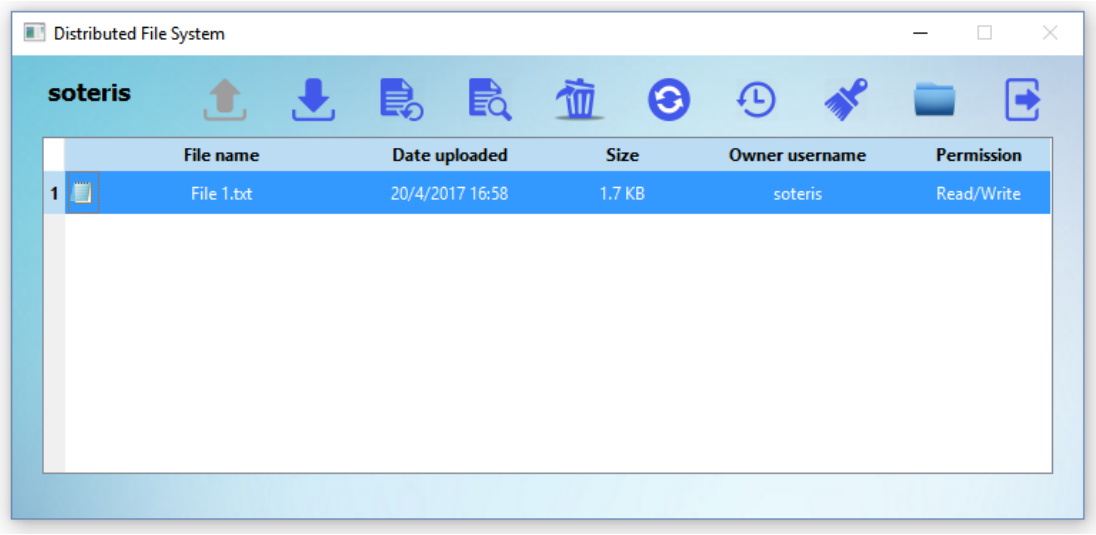


2. Επιλέξτε το κουμπί 
3. Όταν κατεβεί το αρχείο σε ένα προσωρινό χώρο τότε θα ανοίξει αυτόματα
4. Θα ερωτηθείτε αν θέλετε να κατεβάσετε το αρχείο στον φάκελο σας. Αν επιλέξετε ναι τότε το αρχείο θα βρίσκεται στον φάκελο σας.

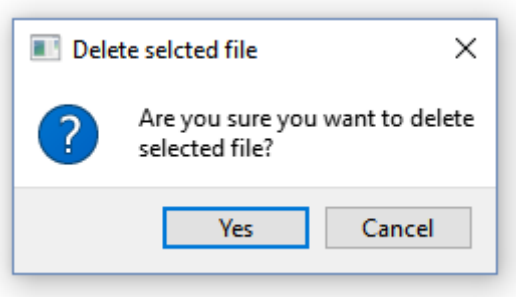


5.8 Διαγραφή αρχείου

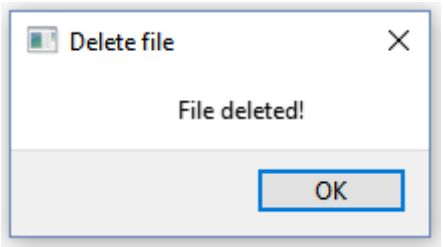
1. Επιλέξτε το αρχείο που επιθυμείτε να διαγράψετε από την λίστα αρχείων.



2. Επιλέξτε το κουμπί 
3. Αν είστε ο owner του αρχείου και είστε σίγουροι πως θέλετε να διαγράψετε το συγκεκριμένο αρχείο τότε απαντήστε «Yes» στην ερώτηση αλλιώς επιλέξτε «Cancel»



4. Αν ολοκληρωθεί επιτυχώς η διαγραφή τότε θα εμφανιστεί το πιο κάτω παράθυρο και το συγκεκριμένο αρχείο δεν θα βρίσκετε πλέον στην λίστα αρχείων.



Σημείωση 1: Αν δεν είστε ο owner του αρχείου τότε δεν μπορείτε να διαγράψετε το αρχείο από το σύστημα.

Σημείωση 2: Επίσης μπορείτε να διαγράψετε αρχείο μέσω του προσωπικού σας φάκελου. Απλώς διαγράψετε από τον προσωπικό σας φάκελο το αρχείο που επιθυμείτε και μετά θα συνεχίσετε από το βήμα 3.

5.9 Μετονομασία αρχείου

1. Μετονομάστε το αρχείο που επιθυμείτε το οποίο βρίσκετε στον προσωπικό σας φάκελο.
2. Αν είστε ο owner του αρχείου και δεν προκύψει κάποιο σφάλμα τότε θα προσέξετε πως το όνομα του αρχείου στην λίστα αρχείων θα αλλάξει και θα έχει το ίδιο όνομα με αυτό που έχετε στον φάκελο σας.

Σημείωση: Αν δεν είστε ο owner του αρχείου τότε δεν μπορείτε να το μετονομάσετε.

5.10 Ιστορικό

Στο ιστορικό αναγράφονται οι τελευταίες ενέργειες που συνέβησαν από τους χρήστες του συστήματος. Συγκεκριμένα αναφέρει τι έχει συμβεί, σε πιο αρχείο, πότε συνέβη η ενέργεια και από ποιόν χρήστη (π.χ. «user soteris add File_1.txt file at 20/4/2017 19:23»). Για να δείτε το ιστορικό πατήστε το κουμπί . Επίσης, κάθε φορά που επιλέγετε το κουμπί  ανανεώνεται και το ιστορικό μαζί με την λίστα αρχείων.

Κεφάλαιο 6

Πειραματική αξιολόγηση

6.1 Προετοιμασία	49
6.2 Πλατφόρμα υλοποίησης	49
6.3 Σενάρια	50
6.4 Αποτελέσματα	51

6.1 Προετοιμασία

Για τον έλεγχο του συστήματος σε πραγματικό περιβάλλον εκτελέσαμε διάφορα σενάρια όπου το καθ' ένα είχε διαφορετικό αριθμό γραφέων και αναγνωστών, και διαφορετικό μέγεθος αρχείων. Επίσης, για την συλλογή των δεδομένων χρησιμοποιήθηκαν scripts. Ακόμη, το κάθε σενάριο εκτελέστηκε 10 φορές όπου κάθε φορά καταγράφαμε τον χρόνο εκτέλεσης και στο τέλος υπολογίσαμε τον μέσο όρο χρόνου εκτέλεσης για κάθε σενάριο ξεχωριστά. Να αναφέρουμε ότι σκοπός μας δεν είναι η σε βάθος πειραματική αξιολόγηση του συστήματος αλλά παραπάνω πραγματοποιείται ως δείγμα ορθότητας (proof of concept) της υλοποίησης και για να έχουμε μια κατ' αρχήν αξιολόγηση σε περιβάλλον LAN.

6.2 Πλατφόρμα υλοποίησης

Οι εξυπηρετητές τρέχανε στους ηλεκτρονικούς υπολογιστές του Πανεπιστημίου Κύπρου. Συγκεκριμένα, οι υπολογιστές αυτοί είναι 64-bit και έχουν επεξεργαστή Intel® Core i5,με 8 GB RAM και δουλεύουν στο λειτουργικό σύστημα Windows 7. Ενώ ο πελάτης έτρεχε σε 64-bit ηλεκτρονικό υπολογιστή με επεξεργαστή Intel® Core i5 2.60 GHz, με 4GB RAM και δουλεύει στο λειτουργικό σύστημα Windows 10. Επίσης το τοπικό δίκτυο (LAN) είχε ταχύτητα αποστολής 689 Mbps και ταχύτητα παραλαβής 220 Mbps. Τέλος να αναφέρουμε ότι τα 3 πρώτα σενάρια εκτελέστηκαν μόνο με την πρώτη εκδοχή του συστήματος όπου ο διαχειριστής αρχείων είναι κεντρικοποιημένος. Σε αντίθεση με το τέταρτο σενάριο όπου συγκρίναμε την πρώτη εκδοχή με την δεύτερη, όπου στην δεύτερη εκδοχή ο διαχειριστής αρχείων είναι κατανεμημένος.

6.3 Σενάρια

Συνολικά εκτελέσαμε τέσσερα (4) διαφορετικά σενάρια. Στο 1ο σενάριο εξετάζουμε πως επηρεάζεται το σύστημα ανάλογα με τον αριθμό των ταυτόχρονων αναγνωστών. Στο 2ο εξετάζουμε πως επηρεάζεται το σύστημα ανάλογα με τον αριθμό των ταυτόχρονων γραφέων. Στο 3ο εξετάζουμε πως επηρεάζεται το σύστημα ανάλογα με το μέγεθος του αρχείου. Και στο 4ο σενάριο εξετάζουμε πόσο επηρέασε την απόδοση του συστήματος η μετατροπή του διαχειριστή αρχείων από κεντρικοποιημένο σε κατανεμημένο.

Σενάριο 1 – Ταυτόχρονοι αναγνώστες

Στο σενάριο αυτό έχουμε 4 διαφορετικές εκδοχές. Συγκεκριμένα, στην κάθε εκδοχή το μόνο που αλλάζουμε είναι ο αριθμός των ταυτόχρονων αναγνωστών και έχουμε εκτελέσεις με 10, 20, 30 και 40 ταυτόχρονους αναγνώστες. Σε όλες τις εκδοχές είχαμε 1 διαχειριστή αρχείων, 3 εξυπηρετητές ευρετηρίου και 3 εξυπηρετητές αρχείων. Ο παράγοντας f είναι 2. Σε αυτό το σενάριο θέλουμε να δούμε πόσο επηρεάζεται η απόδοση του συστήματος ανάλογα με τον αριθμό των ταυτόχρονων αναγνωστών. Επίσης, θα ελέγξουμε και την συνέπεια του συστήματος. Πριν όμως πραγματοποιηθεί η ταυτόχρονη ανάγνωση ανεβάσαμε το αρχείο έτσι ώστε να μπορέσουμε να εκτελέσουμε το σενάριο. Αυτό το σενάριο δοκιμάστηκε σε τοπικό δίκτυο (LAN) και το αρχείο που χρησιμοποιήθηκε είχε μέγεθος 70KB.

Σενάριο 2 – Ταυτόχρονοι εγγραφείς

Στο σενάριο αυτό έχουμε 4 διαφορετικές εκδοχές. Συγκεκριμένα, στην κάθε εκδοχή το μόνο που αλλάζουμε είναι ο αριθμός των ταυτόχρονων εγγραφέων και έχουμε εκτελέσεις με 10, 20, 30 και 40 ταυτόχρονους εγγραφείς. Σε όλες τις εκδοχές είχαμε 1 διαχειριστή αρχείων, 3 εξυπηρετητές ευρετηρίου και 3 εξυπηρετητές αρχείων. Ο παράγοντας f είναι 2. Σε αυτό το σενάριο θέλουμε να δούμε πόσο επηρεάζεται η απόδοση του συστήματος ανάλογα με τον αριθμό των ταυτόχρονων εγγραφέων. Επίσης, θα ελέγξουμε και την συνέπεια του συστήματος. Σε αυτό το σενάριο δεν υπάρχει ανάγκη για να ανεβάσουμε κάποιο αρχείο όπως συνέβη στο Σενάριο 1. Αυτό το σενάριο δοκιμάστηκε σε τοπικό δίκτυο (LAN) και το αρχείο που χρησιμοποιήθηκε είχε μέγεθος 70KB.

Σενάριο 3 – Μέγεθος αρχείου

Στο σενάριο αυτό έχουμε 5 διαφορετικές εκδοχές. Συγκεκριμένα, στην κάθε εκδοχή το μόνο που αλλάζουμε είναι το μέγεθος του αρχείου και έχουμε εκτελέσεις με 2.6, 57, 262, 447 και 1300 MB μέγεθος αρχείου. Σε όλες τις εκδοχές είχαμε 1 διαχειριστή αρχείων, 3 εξυπηρετητές

ευρετηρίου και 3 εξυπηρετητές αρχείων. Ο παράγοντας f είναι 2. Σε αυτό το σενάριο θέλουμε να δούμε πόσο επηρεάζεται η απόδοση του συστήματος ανάλογα με το μέγεθος του αρχείου. Συγκεκριμένα, έχουμε 2 εγγραφείς όπου ανεβάζουν το ίδιο αρχείο κάθε φορά. Τέλος, αυτό το σενάριο δοκιμάστηκε σε τοπικό δίκτυο (LAN).

Σενάριο 4 – Μετατροπή διαχειριστή αρχείων

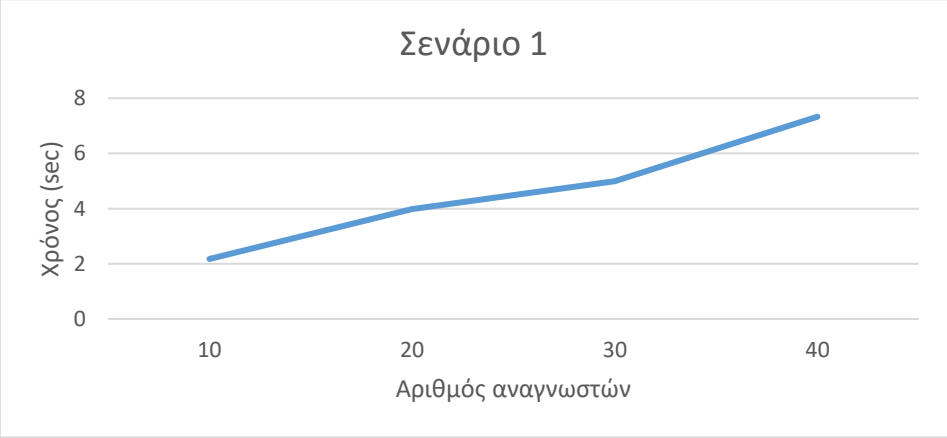
Στο σενάριο αυτό έχουμε 2 διαφορετικές εκδοχές. Συγκεκριμένα, στην μια εκδοχή έχουμε 1 διαχειριστή αρχείων, 3 εξυπηρετητές ευρετηρίου και 3 εξυπηρετητές αρχείων. Ενώ στην άλλη εκδοχή έχουμε 1 διαχειριστή πιστοποίησης, 3 εξυπηρετητές ευρετηρίου και 3 εξυπηρετητές αρχείων. Ο παράγοντας f είναι 2 και στις δύο εκδοχές. Σε αυτό το σενάριο θέλουμε να δούμε πως επηρεάστηκε η απόδοση του συστήματος με την μετατροπή του διαχειριστή αρχείων σε κατανεμημένο και συγκεκριμένα θα εξετάσουμε πως επηρεάστηκε ο χρόνος που χρειάζεται ο πελάτης για να λάβει την λίστα αρχείων και να υπολογίσει την τελική λίστα.

6.4 Αποτελέσματα

Τώρα θα δούμε τα αποτελέσματα για το κάθε σενάριο ξεχωριστά και εξάγουμε συμπεράσματα.

Σενάριο 1 – Ταυτόχρονοι αναγνώστες

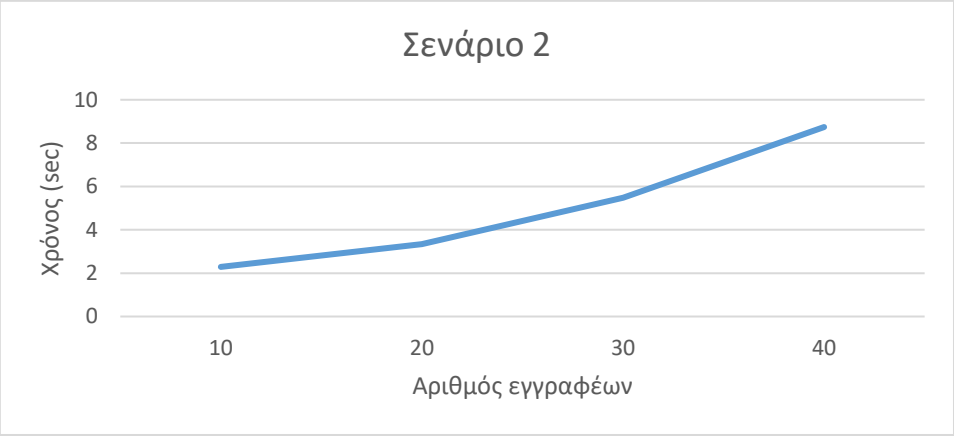
Όπως βλέπουμε και στο Σχήμα 6.1 (γραφική 1^{ου} σεναρίου) όσο αυξάνεται ο αριθμός των ταυτόχρονων αναγνωστών αυξάνεται και ο χρόνος εκτέλεσης. Αυτό είναι αναμενόμενο αφού με την αύξηση του αριθμού των αναγνωστών υπάρχει και αύξηση στον αριθμό των αιτήσεων προς τους εξυπηρετητές. Έτσι οι εξυπηρετητές χρειάζονται παραπάνω χρόνο για εξυπηρετήσουν τις αιτήσεις.



Σχήμα 6.1

Σενάριο 2 – Ταυτόχρονοι εγγραφείς

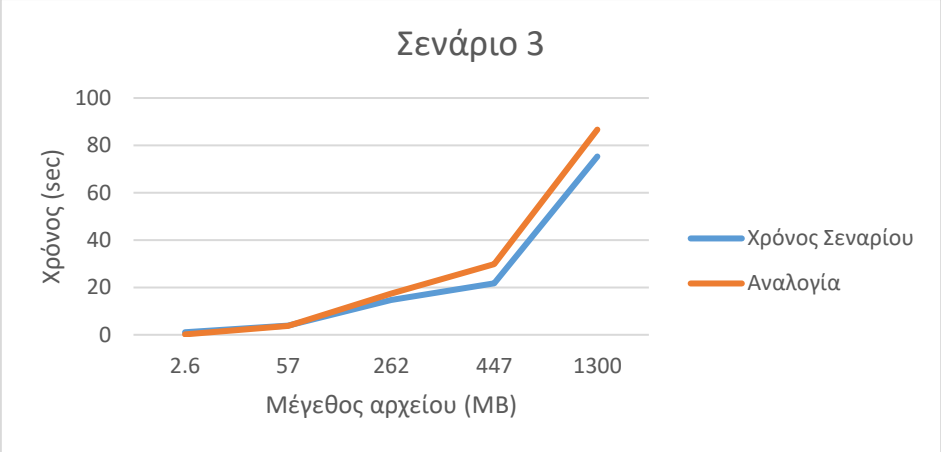
Όπως βλέπουμε και στο Σχήμα 6.2 (γραφική 2ου σεναρίου) όσο αυξάνεται ο αριθμός των ταυτόχρονων εγγραφέων αυξάνεται και ο χρόνος εκτέλεσης. Αυτό είναι αναμενόμενο αφού με την αύξηση του αριθμού των εγγραφέων υπάρχει και αύξηση στον αριθμό των αιτήσεων προς τους εξυπηρετητές. Έτσι οι εξυπηρετητές χρειάζονται παραπάνω χρόνο για εξυπηρετήσουν τις αιτήσεις.



Σχήμα 6.2

Σενάριο 3 – Μέγεθος αρχείου

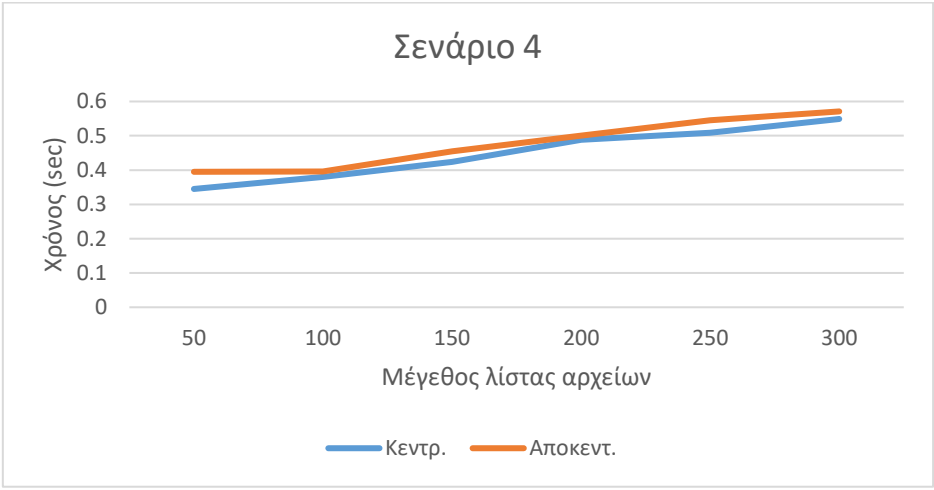
Όπως βλέπουμε και στο Σχήμα 6.3 (γραφική 3ου σεναρίου) όσο αυξάνεται το μέγεθος του αρχείου αυξάνεται και ο χρόνος εκτέλεσης. Αυτό είναι αναμενόμενο αφού με την αύξηση του μεγέθους του αρχείου ο πελάτης χρειάζεται παραπάνω χρόνο για την αποστολή του και άρα η γραφή παίρνει παραπάνω χρόνο. Η γραφική της αναλογίας δείχνει την αναλογία που έχουν τα μεγέθη μεταξύ τους. Συγκεκριμένα, πήρα το κάθε μέγεθος αρχείου και το διαίρεσα με το 15 έτσι ώστε να είναι κοντά στις τιμές του χρόνου. Επίσης αν συγκρίνουμε την συμπεριφορά της γραφικής της αναλογίας με αυτή του χρόνου, βλέπουμε ότι έχουν σχεδόν την ίδια συμπεριφορά και αυτό μας λέει βασικά ότι ανεξαρτήτου μεγέθους δεν υπάρχει κάποια έξτρα επιβάρυνση και ουσιαστικά ο χρόνος αυξάνεται ανάλογα με το μέγεθος του αρχείου. Άρα με αυτό το συμπέρασμα βλέπουμε ότι το σύστημα είναι επεκτάσιμο ως προς το μέγεθος του αρχείου.



Σχήμα 6.3

Σενάριο 4 – Μετατροπή διαχειριστή αρχείων

Όπως βλέπουμε και στο Σχήμα 6.4 (γραφική 4ου σεναρίου) με την μετατροπή του διαχειριστή αρχείων σε κατανεμημένο έχουμε μια μικρή αύξηση του χρόνου και αυτό ήταν αναμενόμενο αφού στην κατανεμημένη εκδοχή ο πελάτης περιμένει απαντήσεις και από τους 3 εξυπηρετητές ευρετηρίου και ακολούθως πρέπει να υπολογίσει πόσες φορές εμφανίστηκε το κάθε αρχείο έτσι ώστε να υπολογίσει την τελική λίστα. Ενώ στην κεντρικοποιημένη εκδοχή ο πελάτης περιμένει μόνο μια απάντηση, από τον διαχειριστή αρχείων και ακολούθως δεν χρειάζεται να κάνει οποιουσδήποτε άλλους υπολογισμούς.



Σχήμα 6.4

Συμπεράσματα της πειραματικής αξιολόγησης:

Βλέποντας τα αποτελέσματα της πειραματικής αξιολόγησης συμπεραίνουμε πως όσο αυξάνεται ο αριθμός των ταυτόχρονων αναγνωστών και εγγραφέων αυξάνεται και ο χρόνος

εκτέλεσης. Επίσης, όταν αυξάνεται το μέγεθος του αρχείου πάλι παρατηρούμε αύξηση στον χρόνο εκτέλεσης. Όμως, η αύξηση του χρόνου είναι ανάλογη με την αύξηση του μεγέθους του αρχείου και άρα το σύστημα μας είναι επεκτάσιμο ως προς το μέγεθος του αρχείου. Τέλος, με την μετατροπή του διαχειριστή αρχείων από κεντρικοποιημένο σε κατανεμημένο ο πελάτης χρειάζεται ελάχιστα περισσότερο χρόνο για να υπολογίσει την λίστα αρχείων, γεγονός που υποδεικνύει ότι η κατανομή του διαχειριστή αρχείων επιφέρει σχεδόν μηδενική επιφόρτιση (overhead) στο σύστημα, ενώ παρέχει πολύ υψηλότερου βαθμού ανοχής σφαλμάτων (ο κεντρικοποιημένος διαχειριστής αρχείων στην πρώτη εκδοχή είναι ουσιαστικά μοναδικό σημείο αποτυχίας – single point of failure).

Κεφάλαιο 7

Συμπεράσματα

7.1 Γενικά συμπεράσματα 55

7.2 Προβλήματα και παραδοχές 56

7.3 Μελλοντική εργασία 56

7.1 Γενικά συμπεράσματα

Στα πλαίσια της ΑΔΕ επεκτείναμε σημαντικά ένα πρώιμο κατανεμημένο σύστημα αρχείων [7] καταλήγοντας στην υλοποίηση ενός ολοκληρωμένου κατανεμημένου συστήματος αρχείων για Windows όπου υπόσχεται ισχυρή συνέπεια δεδομένων στην παρουσία ταυτοχρονισμού. Επίσης, μαζί με το σύστημα έχει δημιουργηθεί και μια διεπαφή μέσω της οποία οι χρήστες μπορούν να εκτελούν τις δυνατότητες που παρέχει το σύστημα. Συγκεκριμένα, το σύστημα παρέχει τις εξής δυνατότητες στους χρήστες: δημιουργία αρχείων, άνοιγμα αρχείων, ανταλλαγή αρχείων με άλλους χρήστες, επεξεργασία αρχείων, διαγραφή και μετονομασία αρχείων, να βλέπουν το ιστορικό του συστήματος, αλληλοεπιδρούν με το σύστημα χρησιμοποιώντας τον προσωπικό τους φάκελο, βλέπουν συνεχώς την λίστα αρχείων του συστήματος όπου περιλαμβάνει όλα τα στοιχεία που χρειάζονται να ξέρουν οι χρήστες, να έχουν προεπισκόπηση ενός αρχείου, καθορίζουν τι δικαιώματα θα έχουν οι άλλοι χρήστες για τα αρχεία που ανέβασαν. Άρα, το σύστημα που υλοποιήσαμε παρέχει όλες τις λειτουργίες που αναμένεται να έχει ένα ΚΣΑ. Ακόμη, έχουμε κάνει απασφαλμάτωση τον κώδικα υλοποίησης των εξυπηρετητών αρχείων και διορθώσαμε σημαντικές παραλείψεις και αδυναμίες. Επιπλέον, τώρα στο σύστημα υπάρχουν εφεδρικά δεδομένα των εξυπηρετητών και των πελατών. Τέλος, πραγματοποιήθηκε μια πειραματική αξιολόγηση του συστήματος ως δείγμα ορθότητας (proof of concept) του συστήματος.

Ένα γενικό συμπέρασμα που βγάζουμε είναι ότι το σύστημα που υλοποιήσαμε έχει συνέπεια ακόμη και στην παρουσία ταυτοχρονισμού αφού αυτό φάνηκε και από την πειραματική αξιολόγηση. Επίσης, το σύστημα είναι επεκτάσιμο ως προς το μέγεθος του αρχείου. Ακόμη, όπως φάνηκε και από την πειραματική αξιολόγηση, όταν αυξάνεται ο αριθμός των

ταυτόχρονων αναγνωστών, των ταυτόχρονων εγγραφέων ή το μέγεθος του αρχείου τότε αυξάνεται αναλόγως και ο χρόνος εκτέλεσης, συμπέρασμα το οποίο αναμέναμε. Τέλος, η μετατροπή του διαχειριστή αρχείων σε κατανεμημένο είχε σχεδόν μηδαμινή αύξηση στον χρόνο που χρειάζεται ο χρήστης για να ανακτήσει την λίστα αρχείων που υπάρχουν στο σύστημα.

7.2 Προβλήματα και παραδοχές

Κατά την διάρκεια της διπλωματικής μου εργασίας υπήρξαν κάποια προβλήματα. Ένα από αυτά ήταν με την εγκατάσταση του μεταγλωττιστή. Για να μπορέσω να μεταγλωττίσω των κώδικα της γλώσσας C χρειαζόταν να εγκαταστήσω στον υπολογιστή μου μεταγλωττιστή που να δέχεται γλώσσα C και να λειτουργεί σε Windows. Ένας τέτοιος μεταγλωττιστής είναι ο MinGW [8] όπου είναι και αυτός που τελικά χρησιμοποίησα. Ακολούθως, έπρεπε να εγκαταστήσω την βιβλιοθήκη glib στον μεταγλωττιστή MinGW, όπου και εκεί υπήρξε πάλι μια δυσκολία. Ένα άλλο πρόβλημα ήταν η αντικατάσταση της συνάρτησης send_file. Η συνάρτηση send_file που χρησιμοποίησε ο Ηλίας στη δική του υλοποίηση [7] για την αποστολή των αρχείων δεν είναι διαθέσιμη σε λειτουργικό σύστημα Windows και χρειαζόταν η εύρεση μιας εναλλακτικής μεθόδου για αποστολή αρχείων. Η εύρεση όμως παρόμοιας μεθόδου ήταν δύσκολη καθώς δεν υπήρχε μια ξεκάθαρη και απλή συνάρτηση για αποστολή αρχείων όπως η send_file. Έτσι χρειάστηκε να χρησιμοποιήσω την συνάρτηση send των νημάτων, όπου η ίδια συνάρτηση χρησιμοποιείται και για την αποστολή των μηνυμάτων μεταξύ πελάτη-εξυπηρετητών. Επίσης, το εργαλείο Qt δεν μπορούσε να μεταγλωττίσει τον κώδικα του πελάτη καθώς δεν μπόρεσα να προσθέσω τις βιβλιοθήκες που έχει το MinGW σε αυτό. Άρα, για την επίλυση αυτού του προβλήματος είχα το Qt να μεταγλωττίζει μόνο τον κώδικα της διεπαφής όπου είναι γραμμένος σε C/C++, ενώ ο πελάτης εξακολουθεί να μεταγλωττίζεται μέσω του MinGW. Έτσι, το Qt πηγαίνει και ξεκινά το εκτελέσιμο αρχείο του πελάτη που παράγεται κατά την διάρκεια μεταγλώττισης του από το MinGW. Να σημειώσουμε ότι όταν το Qt ξεκινάει το εκτελέσιμο του πελάτη δεν αλλάζει κάτι στο εκτελέσιμο και έτσι τα αποτελέσματα των ενεργειών του χρήστη παραμένουν τα ίδια. Τέλος, να σημειώσουμε πως η διεπαφή λειτουργεί μόνο σε 64-bit επεξεργαστές.

7.3 Μελλοντική εργασία

Όπως είναι υλοποιημένο το σύστημα τώρα, ο χρήστης μπορεί να έχει πρόσβαση στο σύστημα (να κάνει login) από διαφορετικούς υπολογιστές. Ανεξάρτητα με ποιο υπολογιστή έχει κάνει login, ο χρήστης θα βλέπει την ίδια λίστα αρχείων. Όμως ο προσωπικός του φάκελος είναι

τοπικός, οπότε θα είναι διαφορετικός και θα είναι ανάλογα με ποιο υπολογιστή έχει κάνει login. Σαν μελλοντική εργασία θα μπορούσαμε να αποθηκεύουμε ποια αρχεία έχει ο χρήστης στον προσωπικό του φάκελο. Έτσι όταν κάνει login από ένα άλλο υπολογιστή, τα αρχεία που θα πρέπει να βρίσκονται στον προσωπικό του φάκελο θα κατεβαίνουν αυτόματα (στο υπόβαθρο) οπότε ο χρήσης δεν θα παρατηρεί οποιαδήποτε διαφορά, ασχέτως από ποιο υπολογιστή μπήκε στο σύστημα.

Όπως αναφέραμε, ο διαχειριστής πιστοποίησης είναι κεντρικοποιημένος. Σαν μελλοντική εργασία θα μπορούσε να μετατραπεί σε κατανεμημένο. Ένας τρόπος για μετατροπή του διαχειριστή πιστοποίησης σε κατανεμημένο θα μπορούσε να είναι ο εξής: να υπάρχουν πολλαπλοί διαχειριστές πιστοποίησης όπως συμβαίνει και με τους εξυπηρετητές, και όταν ένας χρήστης κάνει sign-up στο σύστημα τότε να ενημερώνονται όλοι οι διαχειριστές πιστοποίησης για την νέα εγγραφή και ο πελάτης θα περιμένει f+1 απαντήσεις. Όταν ένας χρήστης θα θέλει να κάνει login, τότε θα στέλνεται το αίτημα του τυχαία σε κάποιο διαχειριστή πιστοποίησης και θα του απαντάει αυτός ανάλογα. Σε περίπτωση που αυτός έχει καταρρεύσει, τότε το αίτημα θα λήγει (θα κάνει timeout) και θα στέλνεται σε άλλον διαχειριστή πιστοποίησης (όλα αυτά στο υπόβαθρο, οπότε ο χρήστης δεν θα αντιλαμβάνεται την όλη διαδικασία). Μια πιο απλή λύση θα μπορούσε να είναι η εξής: να υπάρχει ένας εφεδρικός διαχειριστής πιστοποίησης σε περίπτωση κατάρρευσης του πρώτου (δηλαδή να εφαρμόσουμε παθητική αναπαραγωγή, αντί ενεργής).

Ακόμη μια επέκταση που θα μπορούσε να πραγματοποιηθεί είναι η εξής: τα αρχεία θα μπορούσαν να σπάζουν έτσι ώστε πολλαπλοί χρήστες να μπορούν να δουλεύουν ταυτόχρονα στο ίδιο αρχείο αλλά σε διαφορετικά κομμάτια του αρχείου, και κάθε κομμάτι θα έχει ατομική πρόσβαση. Αυτό δύναται να αυξήσει ακόμη περισσότερο τον ταυτοχρονισμό του συστήματος και την ανοχή σφαλμάτων.

Τέλος, θα πρέπει να πραγματοποιηθεί μια εκτεταμένη και εις βάθος πειραματική αξιολόγηση του όλου συστήματος σε δίκτυο ευρείας εμβέλειας (WAN) όπου θα εντοπιστούν τυχόν προβλήματα ή καθυστερήσεις που πιθανόν να οδηγήσουν σε βελτίωση της επίδοσης και της ευχρηστίας του συστήματος.

Βιβλιογραφία

[1] Antonio Fernandez Anta, Chryssis Georgiou, and Nicolas Nicolaou, CoVer-ability: Consistent Versioning in Asynchronous, Fail-prone, Message-passing Environment, Proc. of 15th IEEE International Symposium on Network Computing and Applications (NCA 2016), pp. 224-231, Cambridge, MA, USA, 2016.

[2] Christos Ploutarchou, A survey on the Consistency Semantics of Distributed File Systems, Department of Computer Science, University of Cyprus, Nicosia, Cyprus, 2014.

[3] Client-server model. https://en.wikipedia.org/wiki/Client%E2%80%93server_model.

[4] Distributed system. https://www.techopedia.com/definition/18909/distributed-system.

[5] Distributer shared memory. https://en.wikipedia.org/wiki/Distributed_shared_memory.

[6] Dropbox. http://www.dropbox.com.

[7] Elias Spanos, An Implementation of a Distributed File System based on Atomic Read/Write Object Algorithms, Bachelor Thesis, Department of Computer Science, University of Cyprus, June 2016.

[8] MinGW. http://www.mingw.org/.

[9] Qt Creator Manual. http://doc.qt.io/qtcreator/.

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

[11] Transmission Control Protocol. https://en.wikipedia.org/wiki/Transmission_Control_Protocol