HY486: Αρχές Κατανεμημένου Υπολογισμού

Εαρινό Εξάμηνο 2019-2020

Δεύτερη Προγραμματιστική Εργασία

Προθεσμία Παράδοσης: 15/6/2020

1. Γενική Περιγραφή

Στη δεύτερη προγραμματιστική εργασία καλείστε να υλοποιήσετε ένα κατανεμημένο σύστημα αποθήκευσης και αντιγραφής (replication) αρχείων το οποίο θα χειρίζεται λειτουργίες που αφορούν τη μεταφόρτωση, ενημέρωση και ανάκτηση αρχείων. Η προγραμματιστική εργασία θα πρέπει να υλοποιηθεί στη γλώσσα C με τη χρήση του Message Passing Interface (MPI) που υπάρχει εγκατεστημένο στα μηχανήματα της σχολής.

2.Υλοποίηση

Στην εργασία αυτή θα υλοποιήσετε ένα κατανεμημένο σύστημα αποθήκευσης με δημιουργία αντιγράφων των δεδομένων, το οποίο θα περιέχει διεργασίες πελάτες (clients) που θα μπορούν να αιτούνται τη μεταφόρτωση, την ενημέρωση και την ανάκτηση αρχείων, καθώς και διεργασίες εξυπηρετητές (servers) που θα είναι υπεύθυνες για την πραγματοποίηση των λειτουργιών αυτών. Προκειμένου να διασφαλισθεί μεγαλύτερη αξιοπιστία στο σύστημα θα πρέπει να γίνει χρήση της δημιουργίας πολλαπλών αντιγράφων (replication) των δεδομένων που εισάγει η κάθε διεργασία πελάτη στο σύστημα. Για το λόγο αυτό, όταν μια διεργασία πελάτη αιτείται κάποια λειτουργία π.χ. ενημέρωση αρχείου , η λειτουργία αυτή θα πρέπει να λαμβάνει χώρα σε παραπάνω από μια διεργασίες εξυπηρετητή στο σύστημα. Συγκεκριμένα, θα πρέπει να λαμβάνει χώρα σε μια πλειοψηφία των διεργασιών εξυπηρετητών που υπάρχουν στο σύστημα.

Με βάση τα παραπάνω, το σύστημα θα πρέπει να διαμορφωθεί ως εξής:

- Στο σύστημα θα υπάρχουν διεργασίες πελάτες που ζητούν την εκτέλεση αιτημάτων και διεργασίες εξυπηρετητές που τα εξυπηρετούν.
- Η διεργασία με αναγνωριστικό (MPI rank) 0 θα παίζει το ρόλο του συντονιστή (coordinator), του οποίου μοναδικός σκοπός είναι να διαβάζει ένα testfile που θα περιέχει την περιγραφή των διαφόρων γεγονότων που πρέπει να προσομοιώνουν οι υπόλοιπες διεργασίες και να αποστέλλει αντίστοιχα μηνύματα στις κατάλληλες διεργασίες για να πραγματοποιηθεί αυτό. Άρα, η διεργασία συντονιστής δεν αποτελεί ούτε διεργασία πελάτη, ούτε διεργασία εξυπηρετητή.

- Υπάρχει ένας πεπερασμένος αριθμός διεργασιών εξυπηρετητών και είναι συνδεδεμένες μεταξύ τους σχηματίζοντας ένα λογικό δακτύλιο. Οι διεργασίες που θα λειτουργούν ως εξυπηρετητές θα έχουν πλήθος **NUM_SERVERS**, όπου το **NUM_SERVERS** θα δίνεται ως παράμετρος στη γραμμή εντολών κατά την εκκίνηση (δείτε Ενότητα 2.1).
- Κάθε διεργασία εξυπηρετητής θα διατηρεί σε μια τοπική δομή όλα τα αρχεία που έχουν αποθηκευτεί στον αποθηκευτικό της χώρο, καθώς και επιπρόσθετη πληροφορία σχετικά με το καθένα από αυτά (βλ. παρακάτω). Κάθε αρχείο έχει μια έκδοση η οποία είναι αρχικά 1 και αυξάνει κατά ένα κάθε φορά που γίνεται ενημέρωση του αρχείου. Μπορείτε να αποφασίσετε μόνοι σας ποια δομή σας εξυπηρετεί για να αποθηκεύετε αυτή την πληροφορία.
- Μια εκ των διεργασιών εξυπηρετητών θα εκλέγεται αρχηγός (με τρόπο που περιγράφεται παρακάτω) και θα αναλαμβάνει καθήκοντα δρομολογητή. Θα επιλέγει δηλαδή ποιες διεργασίες εξυπηρετητές θα επιτελούν τις λειτουργίες μεταφόρτωσης, ενημέρωσης ή ανάκτησης αρχείων που αιτείται κάποια διεργασία πελάτης. Θα διατηρεί πληροφορία σχετικά με τα αναγνωριστικά των υπόλοιπων εξυπηρετητών σε έναν πίνακα μεγέθους ίσου με το πλήθος, **NUM_SERVERS**, των διεργασιών εξυπηρετητών που υπάρχουν στο σύστημα (με τη σειρά που εμφανίζονται στο δακτύλιο). Η πληροφορία αυτή θα του γνωστοποιείται κατά την εκτέλεση του αλγορίθμου εκλογής αρχηγού.

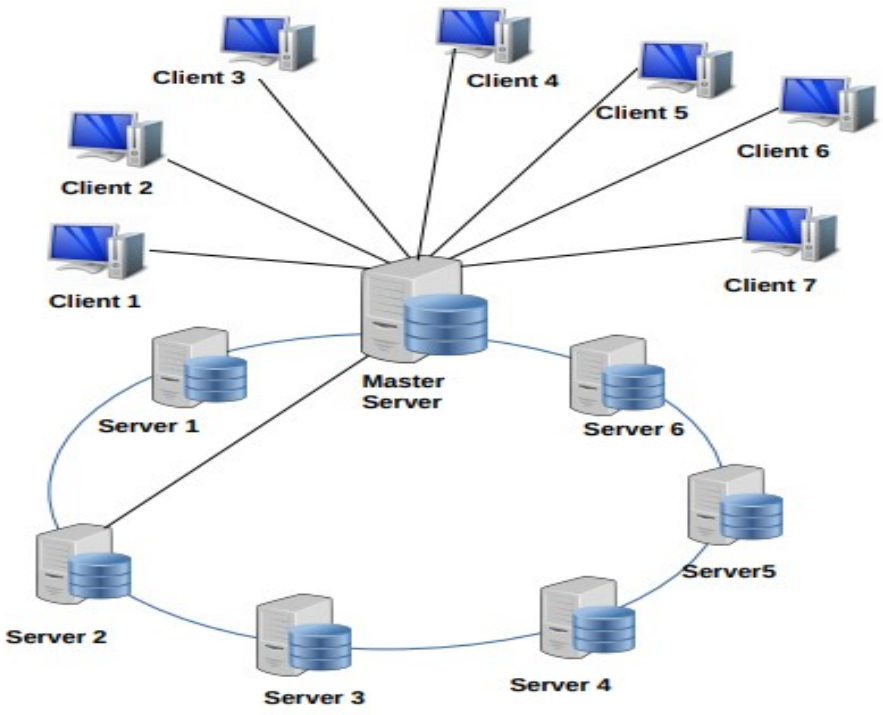
  Ο εξυπηρετητής αρχηγός θα πρέπει να διατηρεί επίσης έναν πίνακα κατακερματισμού ο οποίος θα αποθηκεύει εγγραφές, μια για κάθε αρχείο που είναι αποθηκευμένο στο σύστημα. Άρα, το πλήθος των εγγραφών στον πίνακα κατακερματισμού είναι όσος και ο συνολικός αριθμός αρχείων που είναι αποθηκευμένα στο σύστημα. Κάθε εγγραφή (struct) θα έχει ένα πεδίο key με το αναγνωριστικά του αρχείου στο οποίο αντιστοιχεί. Θα περιέχει επίσης δύο δείκτες, front και back, που θα δείχνουν στο πρώτο και στο τελευταίο στοιχείο μιας ουράς, η οποία λέγεται **ουρά αιτημάτων**, και αποθηκεύει αιτήματα που αφορούν το αρχείο και η εκτέλεσή τους είναι σε εκκρεμότητα. Η ουρά είναι μια FIFO δομή δεδομένων και διασφαλίζει ότι ένα αίτημα που έφτασε πρώτο θα εξυπηρετηθεί πριν τα υπόλοιπα. Κάθε κόμβος της ουράς αντιστοιχεί σε ένα τέτοιο αίτημα και περιλαμβάνει την εξής πληροφορία:

  1. Έναν ακέραιο, id, που αντιστοιχεί στο αναγνωριστικό της διεργασίας πελάτη που υπέβαλε την αίτηση.
  2. Έναν μετρητή (ακέραιο), count, που αντιστοιχεί στο πλήθος των διεργασιών εξυπηρετητών που δεν έχουν ολοκληρώσει την εκτέλεση της αίτησης. Όταν ο μετρητής γίνει 0, η εκτέλεση της αίτησης έχει ολοκληρωθεί.
  3. Το είδος, type, της αίτησης (μεταφόρτωση, ενημέρωση, ανάκτηση).
  4. Έναν βοηθητικό ακέραιο, version, που δηλώνει το μέγιστο αριθμό έκδοσης του αρχείου που υπάρχει στο σύστημα (χρειάζεται σε λειτουργίες ανάκτησης και ενημέρωσης)

  Από τη διεργασία εξυπηρετητή αρχηγού ξεκινά ένας περιορισμένος αριθμός άμεσων συνδέσμων (links), που ονομάζονται μακρινές ακμές, προς άλλες διεργασίες εξυπηρετητών που επιλέγονται τυχαία στο τέλος της εκτέλεσης του αλγορίθμου εκλογής αρχηγού (και αφού έχει αποφασιστεί ποιος θα είναι ο αρχηγός).

- Οι υπόλοιπες διεργασίες στο σύστημα είναι διεργασίες πελάτες. Για κάθε διεργασία πελάτη θα υπάρχει μόνο ένας σύνδεσμος που της επιτρέπει να επικοινωνήσει με τη διεργασία εξυπηρετητή αρχηγό μετά την εκλογή της. Συνεπώς, η τοπολογία του δικτύου θα είναι υβριδική. Οι διεργασίες εξυπηρετητές θα συνδέονται μεταξύ τους μέσω του λογικού δακτυλίου (και των μακρινών ακμών), ενώ οι διεργασίες πελάτες θα συνδέονται **μόνο** με τον εξυπηρετητή αρχηγό και **όχι** μεταξύ τους. Επίσης, η διεργασία εξυπηρετητής αρχηγός θα συνδέεται άμεσα με περιορισμένο αριθμό διεργασιών εξυπηρετητών. Η τοπολογία του δικτύου απεικονίζεται στο Σχήμα 1.
- Κάθε διεργασία πελάτης θα πρέπει επίσης να διατηρεί μια τοπική δομή στην οποία θα αποθηκεύει τα αναγνωριστικά των αρχείων που έχει μεταφορτώσει ή ανακτήσει καθώς και τον αριθμό της έκδοσης του αρχείου που κατέχει. Όταν μια διεργασία πελάτης αρχίζει να μεταφορτώνει ένα αρχείο, θεωρούμε ότι κατέχει την έκδοση 1 του αρχείου αυτού. Όσο μεγαλύτερος είναι ο αριθμός της έκδοσης, τόσο πιο πρόσφατη θεωρείται. Μπορείτε να αποφασίσετε μόνοι σας ποια δομή σας εξυπηρετεί για να αποθηκεύσετε αυτή την πληροφορία.
- Ο συντονιστής μπορεί να ζητήσει από μια διεργασία πελάτη να εκτελέσει ένα αίτημα πριν τερματίσει η εκτέλεση του προηγούμενου αιτήματος. Επομένως, κάθε διεργασία πελάτη μπορεί να έχει παραπάνω από ένα αιτήματα ενεργά, τα οποία έχει αποστείλει προς εξυπηρέτηση στον εξυπηρετητή αρχηγό. Για το λόγο αυτό, κάθε διεργασία πελάτη θα πρέπει να διατηρεί έναν τοπικό μετρητή, **ο οποίος ονομάζεται active_requests,** με αρχική τιμή 0 τον οποίο θα αυξομειώνει ως εξής: κάθε φορά που στέλνει μήνυμα προς τον αρχηγό εξυπηρετητή, ο μετρητής θα αυξάνεται κατά 1, ενώ κάθε φορά που λαμβάνει μήνυμα από τον αρχηγό, ο μετρητής θα μειώνεται κατά 1. Αυτό χρησιμεύει ώστε να γνωρίζει η διεργασία πελάτη αν εκκρεμεί κάποια απάντηση από τον εξυπηρετητή αρχηγό.

Η δομή του testfile, καθώς και τα γεγονότα περιγράφονται αναλυτικά στη συνέχεια.



Σχήμα 1: Παράδειγμα δικτύου

2.1 Εκκίνηση

Κατά την εκκίνηση του συστήματος, εκτελείται μια ρουτίνα, που ονομάζεται MPI_init(), η οποία αρχικοποιεί όλες τις διεργασίες που θα εκτελεστούν στο σύστημα (μία σε κάθε κόμβο του συστήματος).

Η διεργασία συντονιστής ξεκινάει να διαβάζει το testfile, το οποίο αρχικά περιέχει γεγονότα που καθορίζουν ποιες διεργασίες έχουν ρόλο εξυπηρετητή και ταυτόχρονα περιγράφουν τη συνδεσμολογία των διεργασιών προκειμένου να δημιουργηθεί ο δακτύλιος που τις συνδέει. Συγκεκριμένα, τα γεγονότα αυτά έχουν την εξής μορφή:

SERVER <server_rank> <left_neighbour_rank> <right_neighbour_rank>

Γεγονός καθορισμού διεργασίας εξυπηρετητή. Κατά το γεγονός αυτό, ο συντονιστής στέλνει ένα μήνυμα τύπου **<SERVER>** στη διεργασία με αναγνωριστικό **<server_rank>,** ώστε να την ενημερώσει ότι είναι μια διεργασία εξυπηρετητής, καθώς και για το ποιες είναι οι γειτονικές της διεργασίες στο λογικό δακτύλιο. Κάθε διεργασία που λαμβάνει ένα τέτοιο μήνυμα, αποθηκεύει την πληροφορία ότι η διεργασία με αναγνωριστικό **<right_neighbour_rank>** αποτελεί το δεξιό της γείτονα, ενώ η διεργασία με αναγνωριστικό **<left_neighbour_rank>** αποτελεί τον αριστερό της γείτονα στο λογικό δακτύλιο. Κάθε διεργασία που λαμβάνει ένα τέτοιο μήνυμα από το συντονιστή, απλά καταγράφει ότι είναι διεργασία εξυπηρετητής, καθώς και τα αναγνωριστικά του αριστερού και του δεξιού της γειτονικού κόμβου. Ο σχηματιζόμενος δακτύλιος είναι μονής κατεύθυνσης.

Για καθένα από τα παραπάνω μηνύματα τύπου **<SERVER>**, η διεργασία συντονιστής θα πρέπει να περιμένει ένα μήνυμα τύπου **<ACK>** από τη διεργασία στην οποία έστειλε το μήνυμα, πριν προχωρήσει στην επεξεργασία του επόμενου γεγονότος του testfile. Με αυτόν τον τρόπο εξασφαλίζεται ότι όλες οι διεργασίες εξυπηρετητές γνωρίζουν το ρόλο τους και τη θέση τους στο λογικό δακτύλιο κι έτσι μπορεί να ξεκινήσει ομαλά η προσομοίωση του συστήματος.

Μετά το πέρας των γεγονότων <**SERVER>**, το testfile περιέχει ένα γεγονός τύπου **<START_LEADER_ELECTION>**. Όταν ο συντονιστής διαβάσει ένα τέτοιο γεγονός αποστέλλει ένα μήνυμα τύπου **<START_LEADER_ELECTION>** σε κάθε διεργασία εξυπηρετητή που υπάρχει στο σύστημα και έπειτα περιμένει να λάβει ένα μήνυμα τύπου **<LEADER_ELECTION_DONE>** από τη διεργασία που θα εκλεγεί αρχηγός (την οποία δεν γνωρίζει) (βλέπε 2.2).

Όταν ληφθεί αυτό το μήνυμα, η διεργασία συντονιστής αποστέλλει ένα μήνυμα τύπου <**CLIENT>** που περιέχει το **αναγνωριστικό του αρχηγού** σε όλες τις διεργασίες πελάτες, ενημερώνοντάς τες για τον ρόλο τους ως πελάτες, καθώς και για το αναγνωριστικό του εκλεγμένου εξυπηρετητή αρχηγού. Τέλος, ο συντονιστής πρέπει να λάβει μηνύματα τύπου **<ACK>** από κάθε διεργασία πελάτη πριν συνεχίσει με την ανάγνωση του testfile.

2.2 Εκλογή Αρχηγού (Leader Election) και Επιλογή Μακρινών Ακμών

Προκειμένου οι διεργασίες πελάτες να έχουν πρόσβαση στις λειτουργίες του κατανεμημένου συστήματος αποθήκευσης, θα πρέπει να εκλεγεί ένας εξυπηρετητής αρχηγός με τον οποίο θα επικοινωνούν.

Ο αλγόριθμος εκλογής αρχηγού περιγράφεται στη συνέχεια. Κάθε διεργασία εξυπηρετητής διατηρεί μια τοπική μεταβλητή leader_id που αρχικά έχει ως τιμή το δικό της αναγνωριστικό.

- Όταν μια διεργασία εξυπηρετητής, s, λάβει μήνυμα τύπου **<START_LEADER_ELECTION>** από το συντονιστή, αποστέλλει ένα μήνυμα τύπου **<CANDIDATE_ID>** που περιλαμβάνει το δικό της αναγνωριστικό στον αριστερό της γείτονα (αν δεν έχει στείλει ήδη τέτοιου είδους μήνυμα νωρίτερα). Αν το μήνυμα **<START_LEADER_ELECTION>** ληφθεί αφού η διεργασία έχει ήδη αποστείλει μήνυμα τύπου **<CANDIDATE_ID>** στον αριστερό της γείτονα, τότε η s το αγνοεί.
- Όταν κάποια διεργασία εξυπηρετητής λάβει μήνυμα τύπου **<CANDIDATE_ID>** από το δεξιό της γείτονα τότε:
    - Αν δεν έχει στείλει ήδη μήνυμα τύπου **<CANDIDATE_ID>** που να περιλαμβάνει το δικό της αναγνωριστικό στον αριστερό της γείτονα, αποστέλλει τέτοιο μήνυμα διεκδικώντας να γίνει ο αρχηγός.
    - Σε κάθε περίπτωση, η διεργασία εκτελεί τα βήματα που περιγράφονται στη συνέχεια:
    - Αν το αναγνωριστικό, id, που περιέχεται στο ληφθέν μήνυμα είναι μεγαλύτερο του αναγνωριστικού που είναι αποθηκευμένο στη μεταβλητή leader_id, ενημερώνει τη μεταβλητή leader_id να περιέχει το id. Σε κάθε περίπτωση, καταγράφει σε έναν τοπικό πίνακα των NUM_SERVERS θέσεων, ο οποίος ονομάζεται **πίνακας εξυπηρετητών**, το αναγνωριστικό αυτό, προσθέτοντάς το στην πρώτη διαθέσιμη θέση (μετά από όσα αναγνωριστικά έχει προσθέσει ήδη). Όταν έχει λάβει NUM_SERVERS διαφορετικά αναγνωριστικά μέσω μηνυμάτων τύπου **<CANDIDATE_ID>** γνωρίζει πλέον ότι ο κόμβος με το αναγνωριστικό που είναι αποθηκευμένο στην μεταβλητή leader_id είναι ο αρχηγός. Επίσης, γνωρίζει όλα τα αναγνωριστικά των υπολοίπων κόμβων στο δακτύλιο (με τη σειρά που εμφανίζονται σ' αυτόν, με τον δεξιό γειτονικό του κόμβο να βρίσκεται στην πρώτη θέση του πίνακα κ.ο.κ.[1]).
    - Σε κάθε περίπτωση, η διεργασία προωθεί κάθε μήνυμα τύπου **<CANDIDATE_ID>** που λαμβάνει στον αριστερό της γείτονα εκτός αν αυτό περιέχει το δικό της αναγνωριστικό id.
    - Αν μια διεργασία εξυπηρετητής έχει λάβει μηνύματα τύπου **<CANDIDATE_ID>** από NUM_SERVERS διαφορετικές διεργασίες και εξακολουθεί να έχει στη μεταβλητή leader_id το δικό της αναγνωριστικό, εκλέγει τον εαυτό της αρχηγό. Στη συνέχεια, η διεργασία αυτή θα πρέπει επιλέγει τυχαία K/4 διεργασίες εξυπηρετητές, όπου K είναι το πλήθος των διεργασιών εξυπηρετητών εξαιρουμένου του εαυτού της και των 2 γειτόνων της (δηλαδή K = NUM_SERVERS – 3), και να τους αποστέλλει ένα μήνυμα τύπου **<CONNECT>** (προκειμένου να δημιουργήσει τις μακρινές ακμές στο δακτύλιο). Κάθε διεργασία εξυπηρετητή που λαμβάνει ένα τέτοιο μήνυμα, καταγράφει το νέο σύνδεσμο προς τον εξυπηρετητή αρχηγό και μπορεί πλέον να επικοινωνήσει άμεσα

[1] Σημειώνεται ότι το αναγνωριστικό του κόμβου θα κατέχει την τελευταία θέση του πίνακα.

μαζί του χωρίς τη χρήση του λογικού δακτυλίου. Στη συνέχεια, θα πρέπει να αποστείλει στη διεργασία εξυπηρετητή αρχηγό ένα μήνυμα τύπου **<ACK>** και να τυπώσει το εξής:

<SERVER_ID> CONNECTED TO <LEADER_ID>, όπου **<SERVER_ID>** είναι το αναγνωριστικό της διεργασίας εξυπηρετητή και **<LEADER_ID>** το αναγνωριστικό της διεργασίας εξυπηρετητή αρχηγού.

Όταν η διεργασία εξυπηρετητής αρχηγός παραλάβει όλα τα αναμενόμενα μηνύματα τύπου **<ACK>,** αποστέλλει ένα μήνυμα τύπου **<LEADER_ELECTION_DONE>** στη διεργασία συντονιστή.

2.3 Δρομολόγηση μηνυμάτων προς τις Διεργασίες Εξυπηρετητές από τη Διεργασία Εξυπηρετητή Αρχηγό

Στα γεγονότα που περιγράφονται στην ενότητα 2.4, η διεργασία εξυπηρετητής καλείται να αποστείλει μηνύματα σε κάποιες διεργασίες εξυπηρετητές που επιλέγει. Για κάθε ένα από τα μηνύματα της διεργασίας εξυπηρετητή αρχηγού που έχουν προορισμό μια διεργασία εξυπηρετητή θα πρέπει να επιλέγεται το πιο σύντομο μονοπάτι προς αυτήν. Πιο συγκεκριμένα:

- Αν η διεργασία εξυπηρετητής αρχηγός επιθυμεί να στείλει ένα μήνυμα σε διεργασία εξυπηρετητή με την οποία είναι συνδεδεμένη μέσω κάποιας από τις μακρινές ακμές, τότε της αποστέλλει άμεσα το μήνυμα χωρίς να γίνει χρήση του λογικού δακτυλίου.
- Σε αντίθετη περίπτωση, η διεργασία εξυπηρετητής αρχηγός θα πρέπει να επιλέξει το συντομότερο μονοπάτι μεταξύ του εαυτού της και της διεργασίας προορισμού, δηλαδή εκείνο το οποίο θα περιέχει τους λιγότερους δυνατούς ενδιάμεσους κόμβους. Για να το πετύχει αυτό, θα πρέπει να συμβουλευτεί τον πίνακα εξυπηρετητών που έχει δημιουργήσει κατά τη διάρκεια της εκτέλεσης του αλγορίθμου εκλογής αρχηγού. Με βάση την πληροφορία που είναι αποθηκευμένη σ' αυτόν και το γεγονός ότι τα μηνύματα στο δακτύλιο θα μεταδίδονται αριστερόστροφα, η διεργασία αρχηγός είναι σε θέση να εξάγει το πιο σύντομο μονοπάτι προς μια διεργασία εξυπηρετητή εντοπίζοντας το αναγνωριστικό της διεργασίας αυτής μέσα στον πίνακα και έπειτα διασχίζοντας τον πίνακα προς τα δεξιά έως ότου εντοπίσει το αναγνωριστικό κάποιας διεργασίας εξυπηρετητή με την οποία είναι συνδεδεμένη με μακρινή ακμή. Στη συνέχεια, θα αποστείλει το μήνυμα άμεσα σε εκείνη τη διεργασία που εντοπίστηκε ως πλησιέστερη η οποία με τη σειρά της θα το προωθήσει στο λογικό δακτύλιο.

2.4 Λειτουργία του Συστήματος και Γεγονότα

Αφού ο συντονιστής λάβει τα απαραίτητα μηνύματα τύπου **<ACK>** από τις διεργασίες πελάτες, μπορεί να συνεχίσει με την ανάγνωση του testfile. Τα γεγονότα που θα διαβάζει, καθώς και οι ενέργειες που θα πρέπει να γίνονται, περιγράφονται αναλυτικά στη συνέχεια.

UPLOAD <client_rank> <FILE_ID>

Γεγονός μεταφόρτωσης αρχείου. Το γεγονός αυτό σηματοδοτεί ότι ο πελάτης με αναγνωριστικό **<client_rank>** επιθυμεί να μεταφορτώσει ένα αρχείο με αναγνωριστικό **<FILE_ID>** στο κατανεμημένο σύστημα. Για την εκτέλεση του γεγονότος αυτού, ο συντονιστής

πρέπει να στείλει ένα μήνυμα τύπου **<UPLOAD>** στη διεργασία πελάτη με αναγνωριστικό **<client_rank>** που θα περιέχει το αναγνωριστικό αρχείου **<FILE_ID>.** Η διεργασία πελάτη αφού λάβει αυτό το μήνυμα, προσθέτει το αρχείο στην τοπική δομή που διατηρεί θέτοντας αριθμό ως έκδοσης 1 το αποστέλλει αυτούσιο στον εξυπηρετητή αρχηγό. Όταν ο εξυπηρετητής αρχηγός λάβει ένα τέτοιο μήνυμα, θα πρέπει να ελέγχει αν ο πίνακας κατακερματισμού που διατηρεί περιέχει κλειδί ίσο με το **<FILE_ID>**.

- Αν υπάρχει ήδη τέτοιο κλειδί, ο εξυπηρετητής αρχηγός αποστέλλει στη διεργασία πελάτη που αιτήθηκε τη μεταφόρτωση, μήνυμα τύπου **<UPLOAD_FAILED>** που περιέχει το αναγνωριστικό του αρχείου.
- Σε αντίθετη περίπτωση, ο εξυπηρετητής αρχηγός εισάγει το κλειδί **<FILE_ID>** στον πίνακα κατακερματισμού. Η ουρά αιτημάτων για το αρχείο θα πρέπει να περιέχει αρχικά έναν μόνο κόμβο που αντιστοιχεί στην τρέχουσα αίτηση μεταφόρτωσης. Η πληροφορία που θα περιέχει κάθε κόμβος της ουράς έχει περιγραφεί στην Ενότητας 2 (Υλοποίηση). Ο μετρητής που δηλώνει το πλήθος των διεργασιών εξυπηρετητών που δεν έχουν ολοκληρώσει την εκτέλεση της αίτησης θα έχει αρχικά την τιμή (N/2) + 1, όπου N είναι το πλήθος των διεργασιών εξυπηρετητών εξαιρουμένου του εξυπηρετητή αρχηγού (δηλαδή N = NUM_SERVERS-1). Μετά την εισαγωγή στον πίνακα κατακερματισμού, ο εξυπηρετητής αρχηγός επιλέγει τυχαία (N/2) + 1 διεργασίες εξυπηρετητές και αποστέλλει ένα μήνυμα τύπου **<UPLOAD>** σε κάθε μία εξ αυτών ακολουθώντας το πρωτόκολλο δρομολόγησης που περιγράφηκε στην Ενότητα 2.3. Το μήνυμα αυτό περιέχει το αναγνωριστικό της διεργασίας εξυπηρετητή για την οποία προορίζεται το μήνυμα καθώς και το αναγνωριστικό του αρχείου **<FILE_ID>**.

Όταν μια διεργασία εξυπηρετητή λάβει ένα μήνυμα τύπου **<UPLOAD>**, αν δεν προορίζεται για εκείνη, το προωθεί στον αριστερό της γείτονα.

Σε αντίθετη περίπτωση, καταχωρεί το αρχείο στην τοπική δομή στην οποία διατηρεί πληροφορίες για τα αρχεία που είναι αποθηκευμένα στον αποθηκευτικό της χώρο. Για κάθε τέτοιο αρχείο, η τοπική αυτή δομή θα πρέπει να περιέχει το αναγνωριστικό του αρχείου και την έκδοσή του (όταν ένα αρχείο μεταφορτώνεται έχει έκδοση 1). Στη συνέχεια, η διεργασία εξυπηρετητή αν είναι άμεσα συνδεδεμένη με τη διεργασία εξυπηρετητή αρχηγό, αποστέλλει σε εκείνην ένα μήνυμα τύπου **<UPLOAD_ACK>** που θα περιέχει το αναγνωριστικό του αποθηκευμένου αρχείου (μέσω της μακρινής ακμής που τους συνδέει). Σε διαφορετική περίπτωση, προωθεί ένα τέτοιο μήνυμα μέσω του λογικού δακτυλίου αποστέλλοντάς το στον αριστερό της γείτονα. Τα μηνύματα αυτού του τύπου έχουν πάντα ως προορισμό τη διεργασία εξυπηρετητή αρχηγό, συνεπώς οποιαδήποτε άλλη διεργασία εξυπηρετητής τα λάβει τα προωθεί αυτούσια στον αριστερό της γείτονα ή τα αποστέλλει άμεσα στη διεργασία εξυπηρετητή αρχηγό αν συνδέεται με αυτή μέσω κάποιας μακρινής ακμής.

Όταν η διεργασία εξυπηρετητής αρχηγός λάβει ένα τέτοιο μήνυμα, αναζητά στον πίνακα κατακερματισμού το κλειδί που αντιστοιχεί στο **<FILE_ID>**, και στη συνέχεια μειώνει κατά ένα, στον πρώτο κόμβο (στον οποίο δείχνει ο δείκτης front) της ουράς αιτημάτων του αρχείου, τη μεταβλητή που δηλώνει το πλήθος των διεργασιών εξυπηρετητών στους οποίους εκκρεμεί η αίτηση αυτή. Όταν η μεταβλητή αυτή φτάσει στο 0, η διεργασία εξυπηρετητής αρχηγός αποστέλλει στη διεργασία πελάτη που την αιτήθηκε ένα μήνυμα τύπου **<UPLOAD_OK>** που

περιέχει το αναγνωριστικό του αρχείου που μεταφορτώθηκε και αφαιρεί από την ουρά αιτημάτων την ολοκληρωμένη αίτηση μεταφόρτωσης. Στη συνέχεια, η διεργασία εξυπηρετητής αρχηγός συνεχίζει με την εξυπηρέτηση του επόμενου αιτήματος που περιέχεται στην ουρά αιτημάτων του αρχείου (δηλαδή, ανάλογα με τον τύπο αυτού του αιτήματος, εκτελεί εκ νέου τις ενέργειες που περιγράφονται κάτω από την εκάστοτε λειτουργία).

Προσοχή! Αξίζει να σημειωθεί ότι η διεργασία εξυπηρετητής αρχηγός **δεν** περιμένει να ληφθούν μηνύματα τύπου **<UPLOAD_ACK>** για κάθε αίτημα που εξυπηρετεί. Αντίθετα, συνεχίζει να εξυπηρετεί νέα αιτήματα διεργασιών πελατών ενόσω βρίσκονται σε εξέλιξη παλαιότερα αιτήματα.

Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος, η διεργασία πελάτη που αιτήθηκε τη μεταφόρτωση, θα πρέπει να τυπώσει:

- είτε **CLIENT <client_rank> UPLOADED <FILE_ID>,** αν έλαβε μήνυμα τύπου **<UPLOAD_OK>** από τον εξυπηρετητή αρχηγό,
- ή **CLIENT <client_rank> FAILED TO UPLOAD <FILE_ID>,** αν έλαβε μήνυμα τύπου **<UPLOAD_FAILED>** από τον εξυπηρετητή αρχηγό.

RETRIEVE <client_rank> <FILE_ID>

Γεγονός ανάκτησης αρχείου. Το γεγονός αυτό σηματοδοτεί ότι ο πελάτης με αναγνωριστικό **<client_rank>** αιτείται την ανάκτηση αρχείου με αναγνωριστικό **<FILE_ID>**. Ο συντονιστής θα στέλνει ένα μήνυμα τύπου **<RETRIEVE>** στη διεργασία πελάτη με αναγνωριστικό **<client_rank>** που θα περιέχει το αναγνωριστικό αρχείου **<FILE_ID>** . Όταν η διεργασία πελάτη λάβει ένα τέτοιο μήνυμα, το αποστέλλει στη διεργασία εξυπηρετητή αρχηγό **και περιμένει την απάντησή της**.

Όταν λάβει ένα τέτοιο μήνυμα, η διεργασία εξυπηρετητής αρχηγός θα πρέπει να ελέγξει αν το αναγνωριστικό αρχείου **<FILE_ID>** υπάρχει ως κλειδί στον πίνακα κατακερματισμού που διατηρεί.

- Αν το **<FILE_ID>** δεν περιέχεται στον πίνακα κατακερματισμού, η διεργασία εξυπηρετητής αρχηγός θα πρέπει να αποστέλλει, ως απάντηση στη διεργασία πελάτη με αναγνωριστικό **<client_rank>**, ένα μήνυμα τύπου **<RETRIEVE_FAILED>** που θα περιέχει το αναγνωριστικό του αρχείου.
- Σε διαφορετική περίπτωση, θα πρέπει να εισάγει ένα νέο κόμβο στην ουρά αιτημάτων που αντιστοιχεί στο αρχείο με αναγνωριστικό **<FILE_ID>** στον πίνακα κατακερματισμού που διατηρεί, ο οποίος θα αντιπροσωπεύει την τρέχουσα αίτηση ανάκτησης. Η αίτηση αυτή θα αρχίσει να εξυπηρετείται μονάχα όταν καταστεί πρώτη στην ουρά αιτημάτων. Όταν συμβεί αυτό, η διεργασία εξυπηρετητής αρχηγός θα πρέπει να επιλέξει τυχαία (N/2) + 1 διεργασίες εξυπηρετητές και για κάθε μία από αυτές να αποστείλει μηνύματα τύπου **<RETRIEVE>** ακολουθώντας το πρωτόκολλο δρομολόγησης που περιγράφηκε στην Ενότητα 2.3. Κάθε μήνυμα τέτοιου τύπου θα πρέπει να περιέχει το αναγνωριστικό της διεργασίας εξυπηρετητή για την οποία προορίζεται καθώς και το αναγνωριστικό αρχείου **<FILE_ID>**.

Όταν μια διεργασία εξυπηρετητή λάβει μήνυμα τύπου **<RETRIEVE>** που δεν προορίζεται για την ίδια, απλά συνεχίζει την προώθησή του στο δακτύλιο. Σε διαφορετική περίπτωση, αναζητά το αναγνωριστικό του αρχείου **<FILE_ID>** που περιέχεται στο μήνυμα στην τοπική δομή που διατηρεί. Αν βρεθεί αρχείο με τέτοιο αναγνωριστικό, τότε η διεργασία εξυπηρετητής αποστέλλει ένα μήνυμα τύπου **<RETRIEVE_ACK>** που περιέχει το αναγνωριστικό αλλά και την έκδοση του αρχείου που έχει αποθηκευμένο με προορισμό τον εξυπηρετητή αρχηγό. Αν το αρχείο δεν βρεθεί στην τοπική δομή της διεργασίας εξυπηρετητή, τότε αποστέλλεται παρόμοιο μήνυμα με μόνη διαφορά ότι θα περιέχει ως αριθμό έκδοσης το 0. Αν η διεργασία εξυπηρετητής συνδέεται με τη διεργασία εξυπηρετητή αρχηγό με μακρινή ακμή τότε της αποστέλλει αυτό το μήνυμα άμεσα (μέσω αυτής της ακμής). Σε διαφορετική περίπτωση το μήνυμα προωθείται στο λογικό δακτύλιο έως ότου το παραλάβει κάποια διεργασία εξυπηρετητής που συνδέεται με τη διεργασία εξυπηρετητή αρχηγό μέσω μακρινής ακμής, οπότε και μπορεί να της το αποστείλει άμεσα. Για λόγους απλοποίησης της εργασίας, υποθέτουμε ότι ένα μήνυμα αυτού του τύπου περιέχει ολόκληρο το αρχείο που θέλει να ανακτήσει η διεργασία πελάτης (στην έκδοση που βρέθηκε), παρότι το μήνυμα στη πραγματικότητα περιέχει απλά το νούμερο της έκδοσής του.

Για κάθε μήνυμα τύπου **<RETRIEVE_ACK>** για το αρχείο με αναγνωριστικό **<FILE_ID>** που λαμβάνει η διεργασία εξυπηρετητής αρχηγός θα πρέπει να μειώνει κατά ένα τον μετρητή που δηλώνει πόσες διεργασίες εξυπηρετητές δεν έχουν ολοκληρώσει την αίτηση και να αποθηκεύει την τιμή του μεγαλύτερου αριθμού έκδοσης που έχει λάβει στο αντίστοιχο πεδίο (version) του κόμβου της ουράς αιτημάτων. Όταν η διεργασία εξυπηρετητή αρχηγού λάβει όλα τα αναμενόμενα μηνύματα τύπου **<RETRIEVE_ACK>,** αποστέλλει στη διεργασία πελάτη που αιτήθηκε την ανάκτηση ένα μήνυμα τύπου **<RETRIEVE_OK>** που περιέχει το αναγνωριστικό του αρχείου **<FILE_ID>** καθώς και τον αριθμό της πιο πρόσφατης έκδοσής του που βρέθηκε στο σύστημα. Για λόγους απλοποίησης της εργασίας, υποθέτουμε και πάλι ότι σε αυτό το μήνυμα περιέχεται αυτούσια η πιο πρόσφατη έκδοση του αρχείου που θέλει να ανακτήσει η διεργασία πελάτη και όχι απλά το νούμερο της έκδοσης. Στη συνέχεια, αφαιρεί την τρέχουσα αίτηση από την ουρά αιτημάτων που υπάρχει στον πίνακα κατακερματισμού. Πλέον, η διεργασία εξυπηρετητής αρχηγός μπορεί να εξυπηρετήσει την επόμενη εκκρεμή αίτηση για το αρχείο με αναγνωριστικό **<FILE_ID>** (αν υπάρχει).

Αξίζει να σημειωθεί ότι η διεργασία εξυπηρετητής αρχηγός, όπως και στα προηγούμενα γεγονότα, συνεχίζει να λαμβάνει αιτήματα από διεργασίες πελάτες και να εξυπηρετεί αιτήματα που αναφέρονται σε άλλα αρχεία, χωρίς να περιμένει πρώτα τη λήψη όλων των αναμενόμενων μηνυμάτων τύπου **<RETRIEVE_ACK>.**

Μετά το πέρας της εκτέλεσης ενός τέτοιου γεγονότος, η διεργασία πελάτη που αιτήθηκε την ανάκτηση, θα πρέπει να τυπώσει:

- Αν λάβει μήνυμα τύπου **<RETRIEVE_OK>**: **CLIENT <client_rank> RETRIEVED VERSION <VERSION_NUMBER> OF <FILE_ID>** καθώς και να αποθηκεύσει στην τοπική δομή που διατηρεί, το αναγνωριστικό του αρχείου (αν δεν υπάρχει ήδη) και τον αριθμό έκδοσης του αρχείου που έλαβε.

- Αν λάβει μήνυμα τύπου **<RETRIEVE_FAILED>**: **CLIENT <client_rank> FAILED TO RETRIEVE <FILE_ID>**

UPDATE <client_rank> <FILE_ID>

Γεγονός ενημέρωσης αρχείου. Το γεγονός αυτό σηματοδοτεί ότι ο πελάτης με αναγνωριστικό **<client_rank>** αιτείται να ενημερώσει αρχείο με αναγνωριστικό **<FILE_ID>.** Ο συντονιστής θα πρέπει να στείλει ένα μήνυμα τύπου **<UPDATE>** στη διεργασία πελάτη με αναγνωριστικό **<client_rank>** που θα περιέχει το αναγνωριστικό αρχείου **<FILE_ID>.** Η διεργασία πελάτη, αφού λάβει αυτό το μήνυμα, θα αποστέλλει το ίδιο μήνυμα στη διεργασία εξυπηρετητή αρχηγό αφού προσθέσει σε αυτό τον αριθμό της έκδοσης του αρχείου που κατέχει. Σε περίπτωση που δεν κατέχει καμία έκδοση του αρχείου θα προσθέτει στο μήνυμα τον αριθμό 0. Για λόγους απλοποίησης της εργασίας, υποθέτουμε ότι το μήνυμα περιλαμβάνει και τις τροποποιήσεις που θέλει να πραγματοποιήσει η διεργασία πελάτης στο αρχείο με αναγνωριστικό **<FILE_ID>.** Η διεργασία εξυπηρετητής αρχηγός όταν λάβει ένα τέτοιο μήνυμα, θα πρέπει να ελέγξει αν το αναγνωριστικό του αρχείου **<FILE_ID>** υπάρχει ως κλειδί στον πίνακα κατακερματισμού που διατηρεί.

- Αν το **<FILE_ID>** δεν περιέχεται στον πίνακα κατακερματισμού ή η διεργασία πελάτης δεν κατέχει ήδη καμία έκδοση του αρχείου, η διεργασία εξυπηρετητής αρχηγός θα πρέπει να αποστέλλει, ως απάντηση στη διεργασία πελάτη που αιτήθηκε την ενημέρωση, ένα μήνυμα τύπου **<UPDATE_FAILED>** που θα περιέχει το αναγνωριστικό του αρχείου.
- Σε διαφορετική περίπτωση, θα πρέπει να εισάγει έναν νέο κόμβο που θα αντιπροσωπεύει την τρέχουσα αίτηση ενημέρωσης στην ουρά αιτημάτων που αντιστοιχεί στο αρχείο με αναγνωριστικό **<FILE_ID>** και να θέτει την τιμή του πεδίου **version** του κόμβου ίσο με τον αριθμό έκδοσης που περιέχεται στο μήνυμα τύπου **<UPDATE>.** Η αίτηση αυτή θα αρχίσει να εξυπηρετείται από τη διεργασία εξυπηρετητή αρχηγό μονάχα όταν καταστεί πρώτη στην ουρά αιτημάτων του αρχείου με αναγνωριστικό **<FILE_ID>.** Όταν συμβεί αυτό, τότε η διεργασία συντονιστής αρχηγός αποστέλλει στον αριστερό της γείτονα ένα μήνυμα τύπου **<VERSION_CHECK>** το οποίο θα περιλαμβάνει το αναγνωριστικό του αρχείου **<FILE_ID>**, την έκδοση του αρχείου που κατέχει η διεργασία πελάτη καθώς και τον αριθμό **1** ο οποίος θα αποτελεί flag bit. Κάθε διεργασία εξυπηρετητής που λαμβάνει ένα μήνυμα τέτοιου τύπου θα ελέγχει αρχικά την τιμή του flag bit.
    - Αν αυτή είναι 1, η διεργασία εξυπηρετητή αναζητά στην τοπική δομή που διατηρεί το αναγνωριστικό του αρχείου **<FILE_ID>** που περιέχεται στο μήνυμα. Αν το αρχείο δε βρεθεί ή ο αριθμός έκδοσης του αρχείου που υπάρχει στον αποθηκευτικό χώρο της διεργασίας εξυπηρετητή είναι μικρότερος ή ίσος με τον αριθμό έκδοσης που περιέχεται στο μήνυμα, η διεργασία εξυπηρετητής αποστέλλει αυτούσιο το μήνυμα **στον αριστερό της γείτονα**. Σε αντίθετη περίπτωση, θέτει την τιμή του flag bit ίση με 0 και αποστέλλει το μήνυμα με προορισμό τον εξυπηρετητή αρχηγό. Αν η διεργασία εξυπηρετητής συνδέεται με τη διεργασία εξυπηρετητή αρχηγό με μακρινή ακμή τότε της αποστέλλει αυτό το μήνυμα άμεσα. Σε διαφορετική

περίπτωση το μήνυμα προωθείται στο λογικό δακτύλιο έως ότου το παραλάβει κάποια διεργασία εξυπηρετητής που συνδέεται με τη διεργασία εξυπηρετητή αρχηγό μέσω μακρινής ακμής, οπότε και μπορεί να της το αποστείλει άμεσα.

- Αν η τιμή του flag bit στο μήνυμα που παρέλαβε μια διεργασία εξυπηρετητής είναι 0, τότε η διεργασία εξυπηρετητής αποστέλλει αυτούσιο το μήνυμα με προορισμό τη διεργασία εξυπηρετητή αρχηγό δίχως να κάποια άλλη ενέργεια. Η αποστολή γίνεται με τον τρόπο που περιγράφηκε παραπάνω.

Όταν η διεργασία εξυπηρετητής αρχηγός παραλάβει ένα μήνυμα τύπου **<VERSION_CHECK>**, βρίσκει στον πίνακα κατακερματισμού που διατηρεί τον κόμβο που αντιπροσωπεύει την τρέχουσα αίτηση ενημέρωσης που αφορά το αρχείο **<FILE_ID>** και αν το flag bit που περιέχεται στο μήνυμα έχει τιμή 0, η διεργασία εξυπηρετητής αρχηγός αποστέλλει ένα μήνυμα τύπου **<VERSION_OUTDATED>** στη διεργασία πελάτη που αιτήθηκε την ενημέρωση. Το μήνυμα αυτό περιέχει το αναγνωριστικό του αρχείου **<FILE_ID>**. Το μήνυμα αυτού του τύπου ενημερώνει τη διεργασία πελάτη ότι απαγορεύεται να ενημερώσει το αρχείο διότι η ίδια δεν έχει την πιο πρόσφατη έκδοση. Έπειτα, η διεργασία εξυπηρετητής αρχηγός αφαιρεί από την ουρά αιτημάτων του κλειδιού **<FILE_ID>** την τρέχουσα αίτηση. Στη συνέχεια, η διεργασία εξυπηρετητής αρχηγός εξυπηρετεί την επόμενη εκκρεμή αίτηση για το αρχείο με αναγνωριστικό **<FILE_ID>** (αν υπάρχει).

Αν το flag bit έχει τιμή 1, η διεργασία εξυπηρετητής αρχηγός επιλέγει τυχαία (N/2)+1 διεργασίες εξυπηρετητές και τους αποστέλλει μηνύματα τύπου **<UPDATE>** που περιέχουν το αναγνωριστικό της διεργασίας εξυπηρετητή για την οποία προορίζονται, το αναγνωριστικό του αρχείου **<FILE_ID>** καθώς και τον αριθμό της έκδοσης του αρχείου που κατέχει η διεργασία πελάτης. Η αποστολή αυτών των μηνυμάτων γίνεται όπως περιγράφηκε στην Ενότητα 2.3.

Κάθε διεργασία εξυπηρετητής που λαμβάνει μήνυμα τύπου **<UPDATE>**, αν δεν είναι ο προορισμός του μηνύματος, το προωθεί στον αριστερό της γείτονα, αλλιώς αναζητά στην τοπική δομή που διατηρεί το αναγνωριστικό του αρχείου **<FILE_ID>** που περιέχεται στο μήνυμα.

- Αν το αρχείο δεν βρεθεί, προστίθεται στην τοπική δομή με αριθμό έκδοσης **<έκδοση που περιέχεται στο μήνυμα> + 1.**
- Σε αντίθετη περίπτωση, εκτελούνται οι ακόλουθες ενέργειες. Αν η έκδοση του αρχείου με αναγνωριστικό **<FILE_ID>** που υπάρχει στον αποθηκευτικό χώρο της διεργασίας εξυπηρετητή είναι μικρότερη ή ίση της έκδοσης που κατέχει η διεργασία πελάτη που αιτήθηκε την ενημέρωση τότε η διεργασία εξυπηρετητής αλλάζει την έκδοση του αρχείου που υπάρχει στον αποθηκευτικό της χώρο σε **<έκδοση που περιέχεται στο μήνυμα> + 1.** Για λόγους απλοποίησης της εργασίας, υποθέτουμε ότι αυτή η ενέργεια προσομοιώνει επίσης τις αλλαγές στα περιεχόμενα του αρχείου που αιτήθηκε η διεργασία πελάτης.

- Σε κάθε περίπτωση, η διεργασία εξυπηρετητής αποστέλλει ένα μήνυμα τύπου **<UPDATE_ACK>** με προορισμό τη διεργασία εξυπηρετητή αρχηγό το οποίο περιλαμβάνει το αναγνωριστικό του αρχείου **<FILE_ID>**. Η αποστολή του μηνύματος επιβεβαίωσης προς τη διεργασία εξυπηρετητή αρχηγό γίνεται με τον τρόπο που περιγράφηκε για τα μηνύματα επιβεβαίωσης των προηγούμενων γεγονότων.
- Για κάθε μήνυμα τύπου **<UPDATE_ACK>** που λαμβάνει η διεργασία εξυπηρετητής αρχηγός θα πρέπει να βρίσκει στον πίνακα κατακερματισμού που διατηρεί τον κόμβο που αντιπροσωπεύει την τρέχουσα αίτηση ενημέρωσης που αφορά το αρχείο **<FILE_ID>** και να μειώνει κατά ένα τον μετρητή count που δηλώνει το πλήθος των διεργασιών εξυπηρετητών οι οποίες δεν έχουν ολοκληρώσει την αίτηση.
- Όταν η τιμή του μετρητή count καταστεί 0, η διεργασία εξυπηρετητής αρχηγός αποστέλλει στη διεργασία πελάτη που αιτήθηκε την ενημέρωση ένα μήνυμα τύπου **<UPDATE_OK>** το οποίο περιέχει το αναγνωριστικό του αρχείου **<FILE_ID>**. Έπειτα, η διεργασία εξυπηρετητής αρχηγός αφαιρεί από την ουρά αιτημάτων του κλειδιού **<FILE_ID>** την τρέχουσα αίτηση. Στη συνέχεια, η διεργασία εξυπηρετητής αρχηγός εξυπηρετεί την επόμενη εκκρεμή αίτηση για το αρχείο με αναγνωριστικό **<FILE_ID>** (αν υπάρχει).

Αξίζει να σημειωθεί ότι η διεργασία εξυπηρετητής αρχηγός μπορεί να συνεχίσει να λαμβάνει αιτήματα από διεργασίες πελάτες δίχως να περιμένει πρώτα να λάβει όλα τα αναμενόμενα μηνύματα **<UPDATE_ACK>** ή το μήνυμα **<VERSION_CHECK>** για την τρέχουσα αίτηση.

Στο τέλος ενός τέτοιου γεγονότος, η διεργασία πελάτης που αιτήθηκε την ενημέρωση θα πρέπει:

- Αν έλαβε μήνυμα τύπου **<UPDATE_OK>** να αυξήσει τον αριθμό της έκδοσης για το συγκεκριμένο αρχείο στην τοπική δομή που διατηρεί και να τυπώσει: **CLIENT <client_rank> UPDATED <FILE_ID>**
- Αν έλαβε μήνυμα τύπου **<VERSION_OUTDATED>** θα πρέπει να τυπώσει: **CLIENT <client_rank> CANNOT UPDATE <FILE_ID> WITHOUT MOST RECENT VERSION** ενώ
- Αν έλαβε μήνυμα τύπου **<UPDATE_FAILED>** θα πρέπει να τυπώσει: **CLIENT <client_rank> UPDATE FAILED <FILE_ID>**

LEAVE <server_rank> [Υποχρεωτικό για μεταπτυχιακούς, Bonus για προπτυχιακούς: 15%]

Γεγονός αποχώρησης εξυπηρετητή. Το γεγονός αυτό σηματοδοτεί ότι η διεργασία εξυπηρετητή με αναγνωριστικό **<server_rank>** δεν μπορεί να συνεχίσει να αποτελεί κομμάτι του κατανεμημένου συστήματος και θα πρέπει να αποχωρήσει. Ο συντονιστής πρέπει να αποστείλει ένα μήνυμα τύπου **<BARRIER>** σε κάθε διεργασία πελάτη και να περιμένει να λάβει μήνυμα τύπου **<BARRIER_ACK>** από όλες. Αυτό γίνεται για λόγους απλότητας ώστε να διασφαλίσουμε ότι η διεργασία εξυπηρετητής αρχηγός έχει ολοκληρώσει όλα τα αιτήματα που έχει λάβει μέχρι στιγμής από κάθε διεργασία πελάτη.

Όταν μια διεργασία πελάτης λάβει ένα μήνυμα τύπου **<BARRIER>**, απαντάει στο συντονιστή με μήνυμα **<BARRIER_ACK> αφού πρώτα λάβει όλες τις εκκρεμείς απαντήσεις που πιθανότατα**

υπάρχουν από τη διεργασία εξυπηρετητή αρχηγό. Συγκεκριμένα, η διεργασία πελάτης θα πρέπει να ελέγχει το μετρητή active_requests που διατηρεί (και δηλώνει το πλήθος των αιτήσεων που έχει υποβάλλει η διεργασία πελάτης στη διεργασία εξυπηρετητή αρχηγό και δεν έχουν ακόμα απαντηθεί). Αν ο μετρητής αυτός έχει τιμή 0, η διεργασία πελάτης θα αποστέλλει ένα μήνυμα τύπου **<BARRIER_ACK>** στο συντονιστή. Αν η τιμή του μετρητή δεν είναι 0 θα πρέπει η διεργασία πελάτης να απομνημονεύσει το γεγονός ότι έλαβε μήνυμα **<BARRIER>** και όταν η τιμή του μετρητή καταστεί 0 να αποστείλει μήνυμα τύπου **<BARRIER_ACK>** στο συντονιστή.

Όταν ο συντονιστής λάβει μήνυμα τύπου **<BARRIER_ACK>** από κάθε διεργασία πελάτη, αποστέλλει ένα μήνυμα τύπου **<LEAVE>** στη διεργασία εξυπηρετητή με αναγνωριστικό **<server_rank>.**

Στη συνέχεια, η διεργασία εξυπηρετητή με αναγνωριστικό **<server_rank>** θα πρέπει να αποστείλει ένα μήνυμα τύπου **<LEAVE_REQUEST>** με προορισμό τη διεργασία εξυπηρετητή αρχηγό. Αν η διεργασία εξυπηρετητής προς αποχώρηση είναι συνδεδεμένη με τη διεργασία εξυπηρετητή αρχηγό μέσω κάποιας μακρινής ακμής, τότε της αποστέλλει το μήνυμα άμεσα. Σε διαφορετική περίπτωση, το προωθεί στο λογικό δακτύλιο και η πρώτη διεργασία που έχει μακρινή ακμή προς τον εξυπηρετητή αρχηγό που θα το παραλάβει θα του το αποστείλει άμεσα (ή θα φτάσει σ' αυτόν μέσω του δακτυλίου). Όταν η διεργασία εξυπηρετητής αρχηγός παραλάβει ένα μήνυμα τέτοιου τύπου, διαγράφει τη διεργασία με αναγνωριστικό **<server_rank>** από τον πίνακα μεγέθους NUM_SERVERS που διατηρεί καθώς επίσης μειώνει και το πλήθος των διεργασιών το οποίο γνωρίζει κατά 1 (συνεπώς πλέον NUM_SERVERS = NUM_SERVERS -1).

Στη συνέχεια, η διεργασία εξυπηρετητής αρχηγός προωθεί ένα μήνυμα τύπου **<LEAVE_CONFIRMATION> μέσω του λογικού δακτυλίου** που περιλαμβάνει το αναγνωριστικό της διεργασίας εξυπηρετητή προς αποχώρηση. Κάθε διεργασία εξυπηρετητής που λαμβάνει ένα τέτοιο μήνυμα, αφαιρεί από τον πίνακα NUM_SERVERS θέσεων που διατηρεί τη διεργασία εξυπηρετητή προς αποχώρηση. Σε περίπτωση που η διεργασία προς αποχώρηση αποτελεί αριστερό γείτονά της στο δακτύλιο, την αντικαθιστά με την αμέσως επόμενη γειτονική διεργασία εξυπηρετητή και στη συνέχεια αποστέλλει το μήνυμα τύπου **<LEAVE_CONFIRMATION>** στον αριστερό της γείτονα που είναι η διεργασία προς αποχώρηση (θεωρείστε ότι υπάρχει ακόμα ο σύνδεσμος μεταξύ τους και χρησιμοποιείται για τελευταία φορά). Η πληροφορία του ποιά είναι η επόμενη διεργασία που θα αποτελέσει το νέο αριστερό γείτονα γίνεται γνωστή από τον πίνακα NUM_SERVERS θέσεων που διατηρείται σε κάθε διεργασία εξυπηρετητή. Όταν η διεργασία προς αποχώρηση λάβει μήνυμα τύπου **<LEAVE_CONFIRMATION>** που περιέχει το δικό της αναγνωριστικό, για κάθε αρχείο που περιέχεται στον αποθηκευτικό της χώρο, αποστέλλει στον αριστερό της γείτονα μήνυμα τύπου **<TRANSFER>** που περιέχει το αναγνωριστικό του αρχείου καθώς και την έκδοσή του που υπάρχει στον αποθηκευτικό της χώρο (δηλαδή στην τοπική της δομή). Η διεργασία εξυπηρετητής που θα λάβει ένα τέτοιο μήνυμα αναζητά το ληφθέν αναγνωριστικό αρχείου στον δικό της αποθηκευτικό χώρο.

- Αν δεν υπάρχει τέτοιο αρχείο τότε το προσθέτει στον αποθηκευτικό της χώρο και θέτει ως αριθμό έκδοσης τον αριθμό που περιλαμβάνεται στο μήνυμα τύπου **<TRANSFER>.**

- Αν η έκδοση που υπάρχει στον αποθηκευτικό της χώρο είναι μικρότερη της ληφθείσας, τότε η διεργασία εξυπηρετητής θέτει ως νέο αριθμό αίτησης αυτόν που περιέχεται στο μήνυμα τύπου **<TRANSFER>**
- Αν η έκδοση που υπάρχει στον αποθηκευτικό της χώρο είναι μεγαλύτερη ή ίση της ληφθείσας τότε το μήνυμα τύπου **<TRANSFER>** αγνοείται.

Μόλις η μεταφορά των αρχείων στον αριστερό γείτονα ολοκληρωθεί, τότε η διεργασία εξυπηρετητής προς αποχώρηση προωθεί το μήνυμα **<LEAVE_CONFIRMATION>**, που περιλαμβάνει το αναγνωριστικό της, στον αριστερό της γείτονα **και τερματίζει** (καλώντας την MPI_Finalize()). Κάθε διεργασία εξυπηρετητής που λαμβάνει ένα τέτοιο μήνυμα, αφαιρεί από τον πίνακα NUM_SERVERS θέσεων που διατηρεί τη διεργασία εξυπηρετητή προς αποχώρηση και έπειτα συνεχίζει την προώθηση του μηνύματος στο λογικό δακτύλιο. Ο αριστερός γείτονας της διεργασίας που αποχώρησε, επιπρόσθετα ορίζει το νέο δεξιό του γείτονα πριν προωθήσει περαιτέρω το μήνυμα.

Όταν το μήνυμα αυτό ληφθεί από τη διεργασία εξυπηρετητή αρχηγό, η αποχώρηση έχει ολοκληρωθεί και η διεργασία εξυπηρετητής αρχηγός αποστέλλει στη διεργασία συντονιστή μήνυμα τύπου **<LEAVE_ACK>** αφού τυπώσει:

<server_rank> HAS LEFT THE SYSTEM, όπου **<server_rank>** είναι το αναγνωριστικό της διεργασίας εξυπηρετητή που αποχώρησε.

Επίσης, σε περίπτωση που η διεργασία που αποχώρησε ήταν ο αριστερός γείτονας της διεργασίας εξυπηρετητή αρχηγού, η διεργασία εξυπηρετητής αρχηγός θα πρέπει να ορίσει νέο αριστερό γείτονα **πριν** την αποστολή του μηνύματος τύπου **<LEAVE_ACK>** στο συντονιστή.

Υποθέστε για λόγους απλότητας ότι δεν δύναται να αποχωρήσει ποτέ η διεργασία εξυπηρετητής αρχηγός και ο δεξιός της γείτονας.

Μετά τη λήψη του μηνύματος **<LEAVE_ACK>**, ο συντονιστής συνεχίζει κανονικά την ανάγνωση του testfile.

2.5 Τερματισμός Λειτουργίας

Όταν η διεργασία συντονιστή τελειώσει την ανάγνωση του testfile, θα πρέπει να περιμένει όλες τις υπόλοιπες διεργασίες να ολοκληρώσουν τις ανταλλαγές μηνυμάτων που εκκρεμούν. Επομένως, ο συντονιστής αρχικά θα αποστέλλει ένα μήνυμα τύπου **<REQUEST_SHUTDOWN>** σε όλες τις διεργασίες πελάτες. Κάθε διεργασία πελάτης που λαμβάνει ένα τέτοιο μήνυμα θα πρέπει να ελέγχει τον μετρητή που αναφέρθηκε παραπάνω και δηλώνει το πλήθος των αιτήσεων που έχει υποβάλλει η διεργασία πελάτης στη διεργασία εξυπηρετητή αρχηγό και δεν έχουν ακόμα απαντηθεί. Αν ο μετρητής αυτός έχει τιμή 0, τότε η διεργασία πελάτης θα αποστέλλει ένα μήνυμα τύπου **<SHUTDOWN_OK>** στο συντονιστή και θα τερματίζει. Αν η τιμή του μετρητή δεν είναι 0 θα πρέπει η διεργασία πελάτης να απομνημονεύσει το γεγονός ότι έλαβε μήνυμα **<REQUEST_SHUTDOWN>** και όταν η τιμή του μετρητή καταστεί 0 να αποστείλει μήνυμα τύπου **<SHUTDOWN_OK>** στο συντονιστή και να τερματίσει. Όταν ο συντονιστής λάβει μήνυμα τύπου **<SHUTDOWN_OK>** από **όλες** τις διεργασίες πελάτες θα πρέπει να αποστείλει μήνυμα τύπου **<REQUEST_SHUTDOWN>** στη διεργασία εξυπηρετητή αρχηγό, ο

οποίος με τη σειρά του θα πρέπει να το προωθήσει μέσα στον λογικό δακτύλιο. Κάθε διεργασία εξυπηρετητή που λαμβάνει μήνυμα αυτού του τύπου το προωθεί στον αριστερό της γείτονα και τερματίζει. Όταν η διεργασία εξυπηρετητής αρχηγός λάβει ξανά μήνυμα τύπου **<REQUEST_SHUTDOWN>,** αποστέλλει στον συντονιστή μήνυμα **<SHUTDOWN_OK>** και τερματίζει. Όταν ο συντονιστής έχει λάβει μήνυμα τύπου **<SHUTDOWN_OK>** και από τη διεργασία εξυπηρετητή αρχηγό, γνωρίζοντας ότι δεν έχει συμβεί κάτι αναπάντεχο και ότι το υπόλοιπο σύστημα τερμάτισε ομαλά, τερματίζει και ο ίδιος.

3. Message Passing Interface

Για την υλοποίηση της εργασίας θα πρέπει να χρησιμοποιήσετε τη βιβλιοθήκη Message Passing Interface (MPI). Η βιβλιοθήκη αυτή παρέχει τη δυνατότητα ανταλλαγής μηνυμάτων μεταξύ διεργασιών (που μπορεί να εκτελούνται στο ίδιο ή σε διαφορετικά μηχανήματα) μέσω ενός καθιερωμένου Application Programming Interface (API), ανεξαρτήτως γλώσσας και υλοποίησης. Το API αυτό παρέχει τη δυνατότητα ανταλλαγής μηνυμάτων μεταξύ των διεργασιών όχι μόνο με τη χρήση μηνυμάτων σημείο-προς-σημείο (point-to-point messaging), αλλά και με συλλογικό τρόπο, π.χ. υποστηρίζει επικοινωνία μεταξύ ομάδων διεργασιών (multicast), καθώς και καθολική επικοινωνία (broadcasts). Για την προσομοίωση που ζητείται, κάθε κόμβος του συστήματος θα προσομοιώνεται από μία διεργασία MPI. Είναι αξιοσημείωτο ότι περισσότερες από μία διεργασίες MPI μπορεί να τρέχουν στο ίδιο μηχάνημα (παρότι προσομοιώνουν διαφορετικούς κόμβους του συστήματος). Το σύστημα MPI είναι εγκατεστημένο στα μηχανήματα του Τμήματος. Οι δύο βασικές εντολές που απαιτούνται για τη λειτουργία του είναι οι mpicc και mpirun. Η εντολή mpicc χρησιμοποιείται για την μεταγλώττιση προγραμμάτων που χρησιμοποιούν το API του MPI. Με την εντολή mpirun μπορείτε να τρέξετε το εκτελέσιμο ταυτόχρονα σε περισσότερα του ενός μηχανήματα. Ένα παράδειγμα χρήσης παρατίθεται στη συνέχεια:

$ mpirun -np <count> --hostfile <file containing hostnames> <executable> <NUM_SERVERS> <testfile>

όπου η επιλογή **<np>** καθορίζει το συνολικό αριθμό των MPI διεργασιών που θα εκτελούν το εκτελέσιμο αρχείο που δίνεται σαν τελευταίο όρισμα στην παραπάνω γραμμή εντολών.

Στην επιλογή **<hostfile>** δίνεται το όνομα του αρχείου στο οποίο περιέχονται τα hostnames των μηχανημάτων όπου θα εκτελεστούν διεργασίες MPI. Κάθε διεργασία που εκκινείται με τη χρήση της εντολής mpirun έχει ένα ξεχωριστό αναγνωριστικό στο σύστημα MPI, το οποίο ονομάζεται βαθμός (rank) MPI. Επίσης, μπορεί να μάθει πόσες άλλες διεργασίες MPI τρέχουν στο σύστημα και να τους στείλει μηνύματα, χρησιμοποιώντας το rank τους σαν αναγνωριστικό. Είναι αξιοσημείωτο πως μία διεργασία μπορεί να στείλει μηνύματα μόνο σε διεργασίες που έχουν ξεκινήσει από την ίδια εντολή mpirun και είναι κατά συνέπεια μέλη του ίδιου "κόσμου" MPI. Για παραπάνω πληροφορίες μπορείτε να διαβάσετε το υλικό που βρίσκεται στους παρακάτω συνδέσμους:

https://computing.llnl.gov/tutorials/mpi/

http://mpitutorial.com/tutorials/

4. Παράδοση Εργασίας

Για την παράδοση της εργασίας θα πρέπει να χρησιμοποιήσετε το πρόγραμμα turnin, που υπάρχει εγκατεστημένο στα μηχανήματα του τμήματος. Συγκεκριμένα, η εντολή παράδοσης είναι:

turnin project2@hy486

Για να επιβεβαιώσετε ότι η υποβολή της εργασίας ήταν επιτυχής, μπορείτε να χρησιμοποιήσετε την εντολή:

verify-turnin project2@hy486

Προσοχή, τα παραδοτέα σας θα πρέπει να περιέχουν ό,τι χρειάζεται και να είναι σωστά δομημένα ώστε να κάνουν compile και να εκτελούνται στα μηχανήματα της σχολής, όπου και θα γίνει η εξέταση της εργασίας. Η προθεσμία παράδοσης είναι την: 15 Ιουνίου 2020.