ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Κ24: Προγραμματισμός Συστήματος – Εαρινό Εξάμηνο 2013
2η Προγραμματιστική Εργασία
Ημερομηνία Ανακοίνωσης: 25/4/2013
Ημερομηνία Υποβολής: 30/5/2013

Εισαγωγή στην Εργασία

Ο στόχος αυτής της εργασίας είναι να εξοικειωθείτε με τον προγραμματισμό συστήματος σε Unix, και συγκεκριμένα με την δημιουργία νημάτων και την δικτυακή επικοινωνία.

Θα υλοποιήσετε έναν ζεύγος προγραμμάτων (client και server) τα οποία θα φροντίζουν να συγχρονίζουν όλα τα αρχεία σε έναν κατάλογο από τον server προς τον client με αποδοτικό τρόπο, τόσο όσον αφορά την δικτυακή επικοινωνία, όσο και την παραλληλία σε τοπικό και δικτυακό επίπεδο.

Συγκεκριμένα, το πρόγραμμα πελάτης θα συνδεθεί με το πρόγραμμα-εξυπηρετητής και θα ζητήσει από το τελευταίο να υπολογίσει ένα hash-tree για κάθε αρχείο ενός συγκεκριμένου καταλόγου. Ταυτόχρονα, και αυτό από την πλευρά του θα υπολογίζει τα αντίστοιχα hash-trees του δικού του αντίστοιχου καταλόγου. Όταν λάβει σαν απάντηση τα αρχεία και τα hash-trees από τον εξυπηρετητή και έχει υπολογίσει και τα δικά του, θα τα συγκρίνει και θα ζητήσει από τον εξυπηρετητή μόνο τις σελίδες που έχουν διαφορές. Όταν τις λάβει, θα ενημερώσει τα αρχεία του.

Διαδικαστικά:

- Το πρόγραμμά σας θα πρέπει να τρέχει στα μηχανήματα Linux της σχολής. Για επιπρόσθετες ανακοινώσεις, παρακολουθείτε τη λίστα του μαθήματος και το URL: cgi.di.uoa.gr/~mema.
- Υπεύθυνοι για την άσκηση αυτή (ερωτήσεις, αξιολόγηση, βαθμολόγηση, κτλ) είναι οι: Nikos Chondros, Panagiotis Diamantopoulos, Yannis Pappas, and Christos Patsonakis. . Email θα βρείτε στην ιστοσελίδα του μαθήματος.
- Εγγραφείτε στην ηλεκτρονική λίστα (mailman) του μαθήματος και παρακολουθείτε ερωτήσεις/απαντήσεις/διευκρινήσεις που δίνονται σχετικά με την άσκηση (σημείωση: η η-λίστα αυτή δεν έχει καμία σχέση με την hard-copy λίστα που κυκλοφορησε στα πρώτα μαθήματα και στην οποία θα έπρεπε να γράψετε το όνομά σας και το Unix user-id σας).

Τι πρέπει να παραδοθεί:

1. Όλη η δουλειά σας σε ένα tar-file που να περιέχει όλα τα source files, header files, makefile. ΠΡΟΣΟΧΗ: φροντίστε να φτιάξετε τα δικαιώματα αυτού του αρχείου πριν την υποβολή, π.χ. με chmod 755 OnomaEponymoProject1.tar (περισσότερα στη σελίδα του μαθήματος).
2. Μια σύντομη περιγραφή (2-3 σελίδες) για τις επιλογές που κάνατε στο σχεδιασμό της άσκησης, σε μορφή PDF.
3. Οποιαδήποτε πηγή πληροφορίας, συμπεριλαμβανομένου και κώδικα που μπορεί να βρήκατε στο Διαδίκτυο θα πρέπει να αναφερθεί και στον πηγαίο κώδικά σας αλλά και

στην παραπάνω αναφορά.

Υπόβαθρο

Hash-Trees

Ισχύει η εισαγωγή της πρώτης εργασίας σχετικά με συναρτήσεις κατακερματισμού και δέντρα κατακερματισμού.

Αυτή τη φορά όμως, το χτίσιμο του δέντρου είναι πολύ απλούστερο.

Τα δέντρα θα είναι σταθερού μεγέθους, αφού ύψος και fan-out ορίζονται από παραμέτρους.

Έστω λοιπόν δέντρο ύψους height, όπου num_levels=height+1

Για να φτιαχτεί ένα hash-tree σταθερού μεγέθους, που αντιπροσωπεύει ένα αρχείο με αυτή τη λογική, αρκεί η εξής διαδικασία (το σύμβολο ^ σημαίνει "στη δύναμη"):

```
for i = 0 .. height
     level[i] = allocate memory for fanout^i elements
page=0
while not end of file  //iterate through every page of the file
     read page_data (remeber to pad the last with zeroes)
     level[height][page]=hash(page_data)
     page++
while page < fanout^(height) //fill the rest with hash of blank page
     level[height][page] = hash(blank_page)
     page++
```

Έχοντας τα φύλλα, κάνετε ένα πέρασμα προς τα πάνω για να συμπληρώσετε και όλα τα εσωτερικά digests μέχρι το root:

```
level=height - 1
while level >= 0
     for i = 0 .. fanout^level-1   //for each node in this level
          levels[level][i] = hash of elements:
               levels[level+1][fanout*i]
               up to but not including
               levels[level+1][fanout*(i+1)]
     level--
```

Στο τέλος, το levels[0][0] είναι το root digest που "αντιπροσωπεύει" όλο το δέντρο.

Αλγόριθμος σύγκρισης ταξινομημένων σειρών αρχείων

Επίσης θα χρειαστείτε έναν αλγόριθμο που συγκρίνει δύο ταξινομημένες σειρές αρχείων και αποφασίζει τις ενέργειες που πρέπει να εφαρμόσει για να τα συγχρονίσει.

Έστω λοιπόν πίνακες source[] και target[] με ονόματα αρχείων. Ο παρακάτω αλγόριθμος θα τα συγχρονίσει με ένα πέρασμα στον source[] (όπου #πίνακας εννοεί το πλήθος των στοιχείων του):

```
iSource=iTarget=0
while iSource < #source
     while iTarget < #target and target[iTarget] < source[iSource]
          delete file target[iTarget] from target path
          iTarget++
```

```
if iTarget < #target and target[iTarget] == source[iSource]
     compare the contents of the two files and syncronize
     iTarget++
else
     copy file source[iSource] to target path
iSource++
```

Αναλυτική περιγραφή

Θέμα 1: Server, Μονάδες 60 ***

Αναπτύξτε την εφαρμογή *filesyncd*, η οποία θα εξυπηρετεί πολλαπλές συνδέσεις από "πελάτες", έτσι ώστε να τους δίνει τη δυνατότητα να συγχρονίσουν έναν κατάλογο την φορά με τον αντίστοιχο στον server.

Η σύνταξη θα είναι:

filesynd <*port*> <*thread_pool_size*> <*queue_size*> <*tree_height*> <*tree_fanout*>

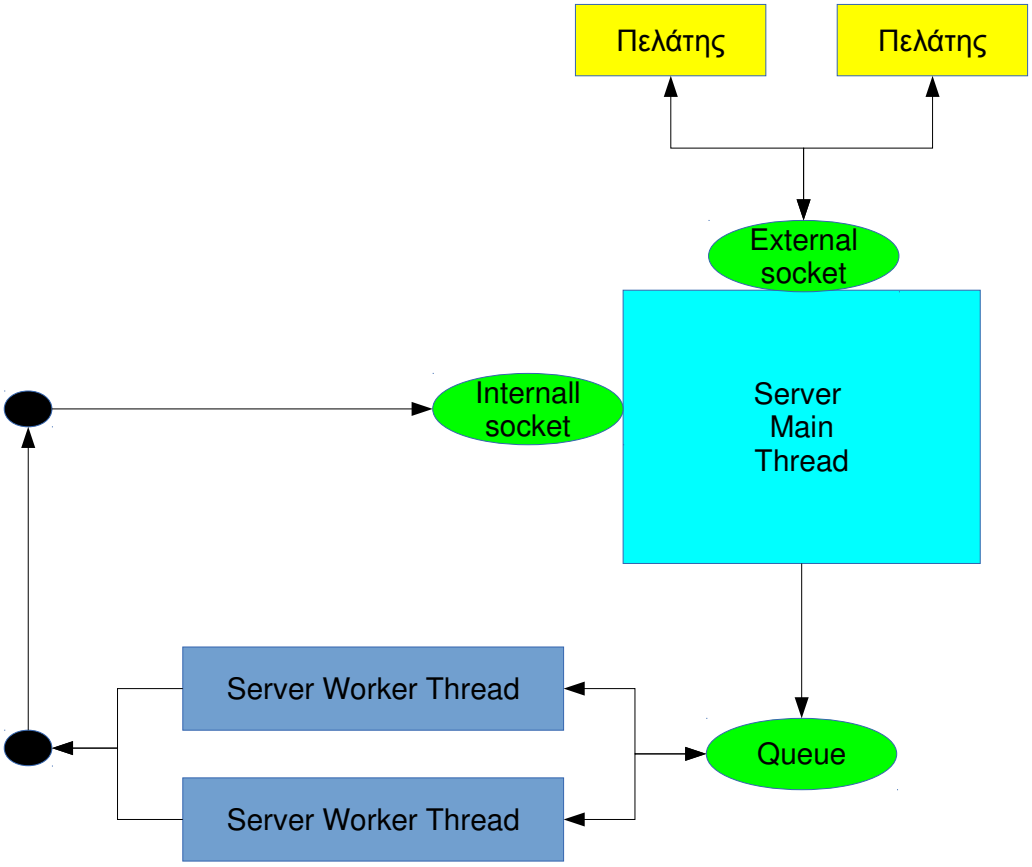
Όπου:

port: Το port στο οποίο θα ακούει για εξωτερικές συνδέσεις
thread_pool_size: Ο αριθμός των worker threads στο thread pool
queue_size: Το μέγεθος της ουράς ενδοεπικοινωνίας
tree_height: το (σταθερό) ύψος του hash-tree
tree_fanout: το (σταθερό) fan-out του hash-tree

Η εφαρμογή θα εξυπηρετεί πολλαπλούς πελάτες μέσω multiplexing σε ένα thread (μέσω select/poll/epoll) και θα μοιράζει την πραγματική εργασία σε ένα thread pool. Όλη η δικτυακή επικοινωνία με τους πελάτες θα γίνεται από ένα και μόνο thread (απαγορεύονται λύσεις thread-per-socket).
Όταν λάβει οποιοδήποτε μήνυμα από πελάτη του δικτύου, θα το προωθήσει σε κάποιο από τα worker threads για να το υλοποιήσει.
Όταν το worker-thread τελειώσει, θα στείλει το αποτέλεσμα πίσω στο κυρίως thread, το οποίο με τη σειρά του θα το προωθήσει στον "πελάτη".

Για την προώθηση μηνυμάτων προς το thread pool, θα χρησιμοποιήσετε μια δομή τύπου ουράς (queue) , υλοποιώντας μια σχέση producer/consumer.
Για την προώθηση απαντήσεων από το thread pool προς το main thread, μπορείτε να χρησιμοποιήσετε είτε sockets τύπου AF_INET (όπου το listening side θα "ακούει" μόνο στο loopback interface), είτε sockets τύπου AF_UNIX.

Ένα διάγραμμα που δείχνει την επιθυμητή διάταξη του server είναι το ακόλουθο:



Τα μηνύματα θα είναι τα ακόλουθα:

Μήνυμα	**Κατεύθυνση** C=Client S=Server W=Worker thread	**Περιγραφή**
GETFILES <directory>	C->S, S->W	Αιτείται τα στοιχεία των αρχείων του καταλόγου <directory>
EXPECT <num>	W->S, S->C	Ενημερώνει τον πελάτη πόσα αρχεία να περιμένει (δηλαδή μηνύματα FILEINFO)
PROCESSFILE <directory> <filename>	S->W	Αιτείται την ανάγνωση του αρχείου και τη δημιουργία του hash-tree
FILEINFO <filename> <filesize> <hash-tree>	W->S, S->C	Ενημερώνει τον πελάτη με τα απαραίτητα στοιχεία για τον συγχρονισμό ενός αρχείου
GETDATA <directory> <filename> <pageno>	C->S, S->W	Αιτείται μια συγκεκριμένη σελίδα ενός αρχείου από τον εξυπηρετητή
DATA <filename> <pageno> <pagedata>	W->S, S->C	Μεταφέρει στον πελάτη τα δεδομένα μιας σελίδας ενός αρχείου
ERROR <message>	W->S, S->C	Μήνυμα λάθους το οποίο αποστέλλει ένα thread σε οποιαδήποτε περίπτωση λάθους, με το ανάλογο μήνυμα.

Έτσι, όταν ληφθεί ένα μήνυμα GETFILES, προωθείται άμεσα στο queue του thread pool. Ένα από τα threads θα το αναλάβει, και θα ανατρέξει στον δοθέντα κατάλογο για να βρει όλα τα αρχεία που περιλαμβάνει. Για κάθε ένα που θα βρει, θα προωθήσει στο queue του thread pool ένα μήνυμα <PROCESSFILE> για να γίνει η ανάγνωσή του. Τέλος, θα στείλει ένα μήνυμα EXPECT στο πελάτη (μέσω του main thread πάντα) έτσι ώστε να τον ενημερώσει πόσα FILEINFO μηνύματα να περιμένει.
Σημειώστε ότι το μέγεθος σελίδας για αυτή την άσκηση είναι σταθερό και ίσο με 4 Kb (4096 bytes).
Σημειώστε επίσης ότι, για να στείλει ένα μήνυμα το κυρίως thread του server σε έναν πελάτη, θα πρέπει να ξέρει για ποιον πελάτη πρόκειται! Άρα, το main thread θα πρέπει να εμπλουτίζει ένα μήνυμα με αυτήν την πληροφορία πριν το τοποθετήσει στο queue για το thread pool.

Τα worker threads θα εκτελούν τελικά μια από τις παρακάτω λειτουργίες:

Μήνυμα	**Λειτουργία**
GETFILES <directory>	Ψάχνει στον συγκεκριμένο κατάλογο για όλα τα αρχεία. Προωθεί από ένα PROCESSFILE μήνυμα για το καθένα στο queue του thread pool.. Τέλος στέλνει ένα μήνυμα "EXPECT" στο main thread για να το προωθήσει στον πελάτη
PROCESSFILE <directory> <filename>	Βρίσκει το μέγεθος του αρχείου και στη συνέχεια φτιάχνει το hash-tree που το συνοψίζει. Στέλνει ένα μήνυμα FILEINFO με αυτά τα στοιχεία στο main thread για να το προωθήσει στον πελάτη.
GETDATA <directory> <filename> <pageno>	Διαβάζει την συγκεκριμένη σελίδα του αρχείου και στέλνει ένα μήνυμα DATA με αυτά τα στοιχεία στο main thread για να το προωθήσει στον πελάτη.

Θέμα 2:Client, Μονάδες 40 ***

Η δεύτερη εφαρμογή που καλείστε να αναπτύξετε είναι το αντίστοιχο πρόγραμμα πελάτη (*filesync*) το οποίο θα επικοινωνεί με τον εξυπηρετητή με σκοπό να συγχρονίσει τα περιεχόμενα ενός τοπικού του καταλόγου.
Η κλήση του προγράμματος από την γραμμή εντολών θα έχει ως εξής:

./filesync <server_ip> <server_port> <thread_pool_size> <queue_size> <tree height> <tree fan-out> <directory-name>

Όπου:

server_ip: Η διεύθυνση IP που χρησιμοποιεί ο εξυπηρετητής
server_port: Η πόρτα που χρησιμοποιεί ο εξυπηρετητής
thread_pool_size: Ο αριθμός των worker threads στο thread pool
queue_size: Το μέγεθος της ουράς ενδοεπικοινωνίας για τα worker threads
tree_height: το (σταθερό) ύψος του hash-tree
tree_fanout: το (σταθερό) fan-out του hash-tree
directory-name: το όνομα του καταλόγου τον οποίο θα συγχρονίσει το filesync

Το ζητούμενο εδώ είναι να κερδίσουμε χρόνο μέσω παραλληλίας σε δύο επίπεδα:

1. Ο client θα πρέπει να χρησιμοποιήσει πολλά νήματα για να διαβάσει τα στοιχεία των (τοπικών) αρχείων.
2. Ο client θα πρέπει να ζητήσει από τον server να του δώσει τα στοιχεία των (απομακρυσμένων) αρχείων χωρίς να περιμένει πρώτα να τελειώσει η δική του επεξεργασία.

Το πρόγραμμα πελάτης λοιπόν εκκινεί την εκτέλεση του διαβάζοντας και επαληθεύοντας τις τιμές των παραμέτρων που έλαβε από την γραμμή εντολών. Στη συνέχεια αρχικοποιεί μια ουρά μηνυμάτων (message-queue) την οποία και θα χρησιμοποιήσει για την επικοινωνία μεταξύ των threads. Τέλος, δημιουργεί τόσα νήματα όσα ορίζει η παράμετρος <*thread_pool_size*>. Το αρχικό νήμα, δηλαδή αυτό που δημιούργησε όλα τα υπόλοιπα, θα το ονομάσουμε το κύριο νήμα (main thread), ενώ τα υπόλοιπα νήματα εργάτες (worker threads).
Τα νήματα εργάτες θα περιμένουν μηνύματα από το queue (σχέση παραγωγού-καταναλωτή, producer-consumer), όπου τον ρόλο του παραγωγού τον αναλαμβάνει το κύριο νήμα και τον ρόλο του καταναλωτή τον αναλαμβάνουν τα νήματα εργάτες. Συνεπώς, σε αυτό το στάδιο του προγράμματος, η διεργασία filesync αποτελείται από <*thread_pool_size*>*+1* νήματα συνολικά.
Έπειτα, το κύριο νήμα, ανοίγει και διαβάζει τα περιεχόμενα του καταλόγου που έλαβε σαν όρισμα από την γραμμή εντολών. Οι κλήσεις συστήματος που θα χρησιμοποιήσετε για αυτήν την λειτουργία ανήκουν στην οικογένεια που ορίζεται στο αρχείο-επικεφαλίδα "dirent.h" και σας είναι ήδη γνωστές από τις διαφάνειες του μαθήματος. Ο σκοπός είναι για κάθε καταχώρηση του καταλόγου (όνομα αρχείου) να δημιουργηθεί ένα δέντρο κατακερματισμού από τα περιεχόμενα του αρχείου, ένα έργο το οποίο θα διεκπεραιωθεί από τα νήματα εργάτες. Για το σκοπό αυτό, το κύριο νήμα δημιουργεί μία αίτηση "*CALCULATE_HASH <filename>"* την οποία την τοποθετεί στην ουρά αιτήσεων. Αυτό το γεγονός έχει σαν αποτέλεσμα να ξυπνήσει (ξεμπλοκάρει) ένα νήμα εργάτη το οποίο θα αναλάβει να την εξυπηρετήσει (δες παρακάτω για το τι θα κάνει). Σε αυτό το σημείο τα νήματα-εργάτες είναι ελεύθερα και αυτόνομα για να επεξεργαστούν τα δεδομένα των ζητούμενων αρχείων (παραλληλία).

Όταν ένα νήμα εργάτης παραλάβει μία αίτηση *CALCULATE_HASH*, εκκινεί τον υπολογισμό ενός δέντρου κατακερματισμού από τα περιεχόμενα του αρχείου που του υποδεικνύει. Κάθε νήμα εργάτης, όταν ολοκληρώσει τον υπολογισμό ενός δέντρου κατακερματισμού, το τοποθετεί σε μία διαμοιραζόμενη λίστα η οποία συγκεντρώνει τα αποτελέσματα όλων των νημάτων εργατών. Τα περιεχόμενα της λίστας αυτής πρέπει να είναι ταξινομημένα βάση του ονόματος του αρχείου, κάτι το οποίο μπορείτε να φροντίσετε κατά την εισαγωγή στη λίστα

Επιστρέφοντας στην λειτουργία του κύριου νήματος, ενώ αυτό διαβάζει τις καταχωρήσεις του καταλόγου, υπάρχει περίπτωση, εκείνη την χρονική στιγμή, όλα τα νήματα εργάτες να είναι ήδη απασχολημένα με τον υπολογισμό δέντρων κατακερματισμού. Σε αυτήν την περίπτωση, το κύριο νήμα δεν θα αναστείλει την λειτουργία του περιμένοντας τα νήματα εργάτες να τελειώσουν με το έργο τους, αλλά θα τοποθετήσει την αίτηση στην διαμοιραζόμενη ουρά και θα συνεχίσει με την επόμενη (αν υπάρχει) καταχώρηση του καταλόγου κανονικά. Συνεπώς, μετά το πέρας της ανάγνωσης των περιεχομένων του καταλόγου από το κύριο νήμα, κατά πάσα πιθανότητα θα εκκρεμούν ακόμη αρκετές αιτήσεις

CALCULATE_HASH.

Χωρίς να περιμένει αυτές να τελειώσουν, το κύριο νήμα επιχειρεί να ανοίξει μία TCP σύνδεση με τον εξυπηρετητή. Όταν συνδεθεί με επιτυχία με τον εξυπηρετητή του αποστέλλει μήνυμα "*GETFILES <directory-name>*", δηλώνοντας έτσι στον εξυπηρετητή πως επιθυμεί να συγχρονίσει τα περιεχόμενα του τοπικού του καταλόγου *<directory-name>* με αυτά που βρίσκονται απομακρυσμένα στον εξυπηρετητή.
Το πρόγραμμα του εξυπηρετητή, με την παραλαβή ενός τέτοιου μηνύματος, θα απαντήσει με τα εξής:

EXPECT <n>
FILEINFO <name$_1$,size$_1$,hash_tree$_1$>
FILEINFO <name$_2$,size$_2$,hash_tree$_2$>
.
.
.
FILEINFO <name$_n$,size$_n$,hash_tree$_n$>

Το μήνυμα "*EXPECT <n>*" δηλώνει στον πελάτη, ουσιαστικά, πόσα αρχεία περιέχει ο κατάλογος. Για κάθε αρχείο του καταλόγου, ο εξυπηρετητής αποστέλλει ένα μήνυμα τύπου *FILEINFO* το οποίο περιέχει μία τριπλέτα που περιλαμβάνει το όνομα, το μέγεθος και το σειριοποιημένο (serialized) δέντρο κατακερματισμού για ένα αρχείο του καταλόγου.
Σημειώστε σε αυτό το σημείο ότι το μήνυμα EXPECT μπορεί να μην έρθει πρώτο, λόγω της παράλληλης επεξεργασίας στην πλευρά του server. Σε ακραία περίπτωση, θα μπορούσαμε να έχουμε ακόμη και το εξής:

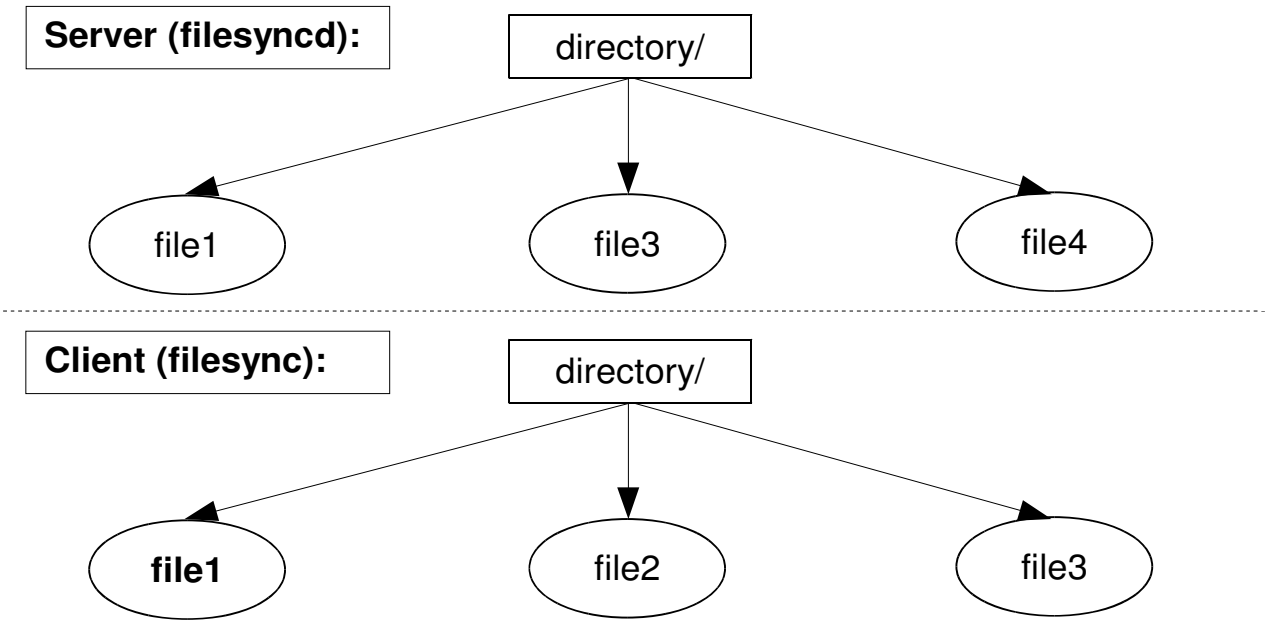
FILEINFO <name$_1$,size$_1$,hash_tree$_1$>
FILEINFO <name$_n$,size$_n$,hash_tree$_n$>
.
.
.
FILEINFO <name$_2$,size$_2$,hash_tree$_2$>
EXPECT <n>

Συνεπώς, το πρόγραμμα πελάτης, μετά την επιτυχή αποστολή ενός *GETFILES* μηνύματος, εισέρχεται σε έναν βρόγχο στον οποίο περιμένει να διαβάσει την απάντηση από τον εξυπηρετητή με στόχο να χτίσει την λίστα-αρχείων-εξυπηρετητή. Αν υποθέσουμε ότι όλα βαίνουν καλώς και ο πελάτης λάβει πρώτα το μήνυμα *EXPECT*, τότε είναι σε θέση να γνωρίζει ακριβώς πόσα μηνύματα *FILEINFO* θα διαβάσει στην συνέχεια. Όμως, επειδή, όπως προείπαμε, υπάρχει περίπτωση τα μηνύματα να ληφθούν ανακατεμένα, ο πελάτης θα πρέπει να είναι σε θέση να χειριστεί αυτήν την περίπτωση. Συνεπώς, ενώ λαμβάνει *FILEINFO* μηνύματα και δεν έχει λάβει το μήνυμα *EXPECT,* τα αποθηκεύει ούτως ή άλλως στην λίστα-αρχείων-εξυπηρετητή και συνεχίζει με την επόμενη ανάγνωση. Όταν τελικά λάβει το μήνυμα *EXPECT* είναι σε θέση πλέον να γνωρίζει πόσα και αν θα ακολουθήσουν άλλα *FILEINFO* μηνύματα. Όταν δεν υπάρχουν άλλα διαθέσιμα μηνύματα, ο πελάτης προχωρά στο επόμενο βήμα της εκτέλεσής του. Σημειώστε ότι και αυτή η λίστα θα πρέπει να είναι ταξινομημένη, κάτι το οποίο μπορείτε και πάλι να φροντίσετε κατά την εισαγωγή.

Σε αυτό το σημείο, το κύριο νήμα ελέγχει αν όλα τα νήματα εργάτες έχουν ολοκληρώσει το

έργο τους. Σε περίπτωση που κάποιο/α είναι ακόμα απασχολημένα, τώρα και μόνον τώρα θα περιμένει να τερματίσουν. Με αυτόν τον τρόπο έχουμε πετύχει την επικάλυψη του τοπικού υπολογισμού των δέντρων κατακερματισμού με την δικτυακή επικοινωνία με τον εξυπηρετητή μέσω της παραλληλίας.

Όταν όλα τα νήματα εργάτες τερματίσουν την λειτουργία τους, τότε το κύριο νήμα εισέρχεται στο στάδιο συγχρονισμού των περιεχομένων του φακέλου. Στα πλαίσια αυτής της περιγραφής θα θεωρήσουμε ότι η κατάσταση των φακέλων στον πελάτη και στον εξυπηρετητή είναι αυτή του παρακάτω σχήματος.



Σχήμα 1: Περιεχόμενα του υπό συγχρονισμού καταλόγου με όνομα “directory” στην πλευρά του πελάτη και του εξυπηρετητή.

Το αρχείο “file3” υπάρχει και είναι το ίδιο και απαράλλαχτο, το “file1” υπάρχει και στις δύο πλευρές αλλά, τόσο τα περιεχόμενά του, όσο και το μέγεθός του ή και τα δύο, διαφέρουν. Το “file2” υπάρχει μόνο στην πλευρά του πελάτη και αντιστοίχως, το “file4”, μόνο στην πλευρά του εξυπηρετητή.

Σε αυτό το στάδιο της εκτέλεσής του, το κύριο νήμα έχει στην διάθεσή του δύο λίστες. Η πρώτη περιέχει καταχωρήσεις που αφορούν τον τοπικό του κατάλογο ενώ η δεύτερη περιέχει τις καταχωρήσεις στην πλευρά του εξυπηρετητή.

Για να συγχρονιστούν τα περιεχόμενα των φακέλων πρέπει:

1. Να δημιουργηθούν όλα τα αρχεία που υπάρχουν στην δεύτερη λίστα και δεν υπάρχουν στην πρώτη. Σε αυτήν την περίπτωση το πρόγραμμα πελάτης θα εκτυπώνει στην οθόνη μήνυμα: “Creating file <*filename*> with size <*size*>”.
2. Να διαγραφούν όλα τα αρχεία που υπάρχουν στην πρώτη και δεν υπάρχουν στην δεύτερη. Σε αυτήν την περίπτωση το πρόγραμμα πελάτης θα εκτυπώνει στην οθόνη μήνυμα: “Deleting file <*filename*>”.
3. Να συγκριθούν τα αρχεία που εμφανίζονται και στις δύο λίστες χρησιμοποιώντας τον αλγόριθμο σύγκρισης των δέντρων Merkle που μάθατε στην $1^{η}$ εργασία. Σε αυτήν την περίπτωση το πρόγραμμα πελάτης θα εκτυπώνει στην οθόνη μήνυμα: “File <*filename*> client and server versions differ in page <*page_number*>”, για κάθε

σελίδα στην οποία τα περιεχόμενα του αρχείου <*filename*> διαφέρουν.

Στο υπόβαθρο σας έχει δοθεί ένας αλγόριθμος που υλοποιεί τα παραπάνω σε ένα πέρασμα.

Χρησιμοποιώντας την παραπάνω μεθοδολογία καθίσταται εφικτό το πρόγραμμα πελάτης να ζητήσει από τον εξυπηρετητή μόνο τις σελίδες δεδομένων που χρειάζεται, περιορίζοντας έτσι το δικτυακό κόστος επικοινωνίας.

Το πρόγραμμα πελάτης λοιπόν θα ζητήσει μία-μία όλες τις σελίδες δεδομένων από τον εξυπηρετητή με την αποστολή *GETDATA* μηνυμάτων.

Σημειώστε ότι θα πρέπει να χρησιμοποιήσετε το πεδίο "μέγεθος αρχείου" έτσι ώστε, και να ζητήσετε σελίδες μόνο στο πραγματικό εύρος δεδομένων του αρχείου, αλλά και να θέσετε το σωστό μέγεθος αρχείου μετά τον συγχρονισμό (δες ftruncate).

Μετά το πέρας αυτής της διαδικασίας, τα περιεχόμενα του καταλόγου έχουν συγχρονιστεί με επιτυχία και το πρόγραμμα πελάτης τερματίζει την λειτουργία του εκτυπώνοντας μήνυμα επιτυχίας.

Θέμα 3 (Extra credit), Μονάδες 20 ***

Προσέξτε ότι το τελικό στάδιο του προγράμματος client είναι απολύτως σειριακό.

Η ροή δεδομένων είναι η εξής:

Αποστολή αίτησης για σελίδα δεδομένων στον server (write to socket)
<Αναμονή απάντησης: κόστος δικτυακής επικοινωνίας>
Αίτηση εγγραφής δεδομένων στο δίσκο (write to file)
<Αναμονή ολοκλήρωσης: κόστος εγγραφής σε δίσκο>

Και αυτό επαναλαμβάνεται για κάθε σελίδα κάθε αρχείου που πρέπει να έρθει από τον server.

Είναι προφανές ότι αυτές οι αναμονές θα μπορούσαν να επικαλυφθούν με επεξεργασία, αν:

1. Υπήρχε σταθερή ροή αιτήσεων σελίδων προς το δίκτυο, χωρίς αναμονή για πρότερη απάντηση (δες πρωτόκολλα επανεκπομπής, στόχος η αποφυγή του Stop-And-Wait και της τραγικά μικρής δικτυακής απόδοσης που επιφέρει).
2. Η ροή αιτήσεων εγγραφής δεδομένων στον δίσκο ήταν σταθερή και επικάλυπτε την αναμονή απάντησης από το δίκτυο.

Σχεδιάστε και εφαρμόστε μια λύση στο παραπάνω πρόβλημα.

Hint: A socket can safely be read from one thread and written to by another thread (that is, two threads need no further synchronization if one is only reading from a socket and the other is only writing to it).

Διαδικαστικά

Επισημάνσεις/Παραδοχές:

- Υποθέστε ότι ο κατάλογος για συγχρονισμό μπορεί να είναι μόνο άμεσος υπό-κατάλογος του τρέχοντος καταλόγου, τόσο για τον client όσο και για τον server.
- Υποθέστε ότι δεν υπάρχουν hard/soft links, όπως και υποκατάλογοι στους καταλόγους για συγχρονισμό.
- Υποθέστε ότι τα αρχεία στον server δεν αλλάζουν για όλη τη διάρκεια ζωής του προγράμματος server
- Θα πρέπει να δώσετε μια καλή λύση στην οριοθέτηση μηνυμάτων στο κανάλι (see: message framing for TCP socket). Θα πρέπει να γνωρίζετε που αρχίζει και που τελειώνει ένα μήνυμα, και να μπορείτε να μεταφέρετε οποιαδήποτε πληροφορία

μεταβλητού μεγέθους χωρίς περιορισμούς όσον αφορά τους χαρακτήρες που περιλαμβάνονται.

- Θα πρέπει να διαχειριστείτε κατάλληλα την περίπτωση που η μια πλευρά κλείνει την σύνδεση (SIGPIPE/EPIPE)
- Θα εκτιμηθεί περισσότερο μια λύση που σέβεται το γεγονός ότι ο client και ο server μπορεί να τρέχουν σε πλατφόρμες διαφορετικής αρχιτεκτονικής (χρήση htonX, ntohX).
- Συνεχίζοντας με το παραπάνω, η ενδοεπικοινωνία με sockets όμως δεν έχει τέτοιους περιορισμούς. Μάλιστα, επειδή εγγυημένα στην περίπτωσή μας τα δύο άκρα είναι στην ίδια διεργασία, μπορείτε να μεταφέρετε ακόμη και δείκτες μέσω αυτής.
- Η σύγκριση των hash-trees θα πρέπει να γίνει από τη ρίζα προς τα κάτω και όχι φύλλο-φύλλο.
- Οποιαδήποτε καλύτερη λύση για την ταξινόμηση των λιστών με τα ονόματα αρχείων είναι ευπρόσδεκτη.
- Εννοείτε ότι περιμένουμε αμυντικό προγραμματισμό (έλεγχο λαθών) σε όλα τα επίπεδα (από τις παραμέτρους των προγραμμάτων μέχρι την ορθότητα των δικτυακών μηνυμάτων).

Τι θα βαθμολογηθεί:

1. Η συμμόρφωση του κώδικά σας με τις προδιαγραφές της άσκησης
2. Η οργάνωση και η αναγνωσιμότητα (μαζί με την ύπαρξη σχολίων) του κώδικα
3. Η χρήση Makefile και η κομματιαστή σύμβολο-μετάφραση (separate compilation).
4. Η αναφορά που θα γράψετε και θα υποβάλετε μαζί με τον πηγαίο κώδικα σε μορφή PDF.

Άλλες σημαντικές παρατηρήσεις:

1. Οι εργασίες είναι **ατομικές.**
2. Όποιος υποβάλλει / δείχνει κώδικα που δεν έχει γραφτεί από την ίδια/τον ίδιο **μηδενίζεται** στο μάθημα.
3. Αν και αναμένεται να συζητήσετε με φίλους και συνεργάτες το πως θα επιχειρήσετε να δώσετε λύση στο πρόβλημα, αντιγραφή κώδικα (οποιασδήποτε μορφής) είναι κάτι που **δεν επιτρέπεται** και δεν πρέπει να γίνει. Οποιοσδήποτε βρεθεί αναμεμειγμένος σε αντιγραφή κώδικα **απλά παίρνει μηδέν** στο μάθημα. Αυτό ισχύει για **όλους όσους εμπλέκονται**, ανεξάρτητα από το ποιος έδωσε/πήρε κλπ.
4. Το πρόγραμμά σας θα πρέπει να γραφτεί σε C (ή C++ **χωρίς όμως** STL extensions) και θα πρέπει να τρέχει σε Ubuntu-Linux, αλλιώς **δεν θα βαθμολογηθεί**.
5. Σε καμία περίπτωση τα MS-Windows **δεν είναι επιλογή** πλατφόρμας για την παρουσίαση αυτής της άσκησης.