

Κ24: Προγραμματισμός Συστήματος
3η Εργασία – Εαρινό Εξάμηνο 2017
Προθεσμία Υποβολής: Κυριακή 4 Ιουνίου 2017 Ώρα 23:59

Εισαγωγή στην Εργασία:

Ο στόχος της εργασίας αυτής είναι να εξοικειωθείτε με την χρήση sockets και νημάτων (threads) στην δημιουργία μιας διαδικτυακής εφαρμογής σε περιβάλλον LINUX. Επιπλέον στα πλαίσια της εργασίας θα πρέπει να συγχρονίσετε τα νήματα που η εφαρμογή σας θα χρησιμοποιήσει.

Η εργασία σας θα υλοποιεί ένα σύστημα κατοπτρισμού επιλεγόμενης έντασης (selective-overhead mirroring) για αντιγραφή πολλαπλών απομακρυσμένων καταλόγων σε μία επιλεγμένη συσκευή. Το εν λόγω σύστημα κατοπτρισμού αποτελείται από μια σειρά από διαδικτυακά προγράμματα που όλα μαζί επιτυγχάνουν την επιθυμητή μεταφορά αρχείων. Επιμέρους προγράμματα θα πρέπει να κάνουν χρήση νημάτων για να επιτευχθεί η «παράλληλη» μεταφορά δεδομένων.

Συνοπτική Περιγραφή:

Το Σχήμα ;; παρουσιάζει την οργάνωση και την αλληλεπίδραση εμπλεκόμενων προγραμμάτων στην υλοποίηση του συστήματος. Το διαδικτυακό πρόγραμμα πελάτη MirrorInitiator, μπορεί να επικοινωνήσει με διαδικασία κατοπτρισμού (MirrorServer) σε οποιαδήποτε μηχανή παρέχοντας όχι μόνο τις συσκευές αλλά και συγκεκριμένα στοιχεία των αντίστοιχων συστημάτων αρχείου που θα πρέπει να κατοπτριστούν.

Ο χρήστης του MirrorInitiator παραθέτει ως παραμέτρους τις δικτυακές διευθύνσεις του MirrorServer και ενός ή περισσότερων ContentServers, μαζί με μία παράμετρο ανοχής σε καθυστέρηση που θα πρέπει να επιδειχθεί από κάθε ContentServer που συμμετάσχει στον κατοπτρισμό.

Ο MirrorServer δέχεται μέσω δικτυακής σύνδεσης τις πληροφορίες για τις απομακρυσμένες συσκευές από τις οποίες θα πρέπει να αντλήσει τα αρχεία με την βοήθεια νημάτων που τα ονομάζουμε MirrorManagers. Τέλος ο MirrorServer με την βοήθεια ενός σταθερού αριθμού νημάτων τύπου worker μεταφέρει στον τοπικό του αποθηκευτικό χώρο τα αρχεία που διαθέτουν οι απομακρυσμένες συσκευές.

Οι απομακρυσμένες συσκευές θα εκτελούν το πρόγραμμα ContentServer, το οποίο είναι υπεύθυνο για την εξυπηρέτηση των αιτήσεων από τον MirrorServer όσον αφορά τη λίστα των διαθέσιμων αρχείων/καταλόγων αλλά και για τη μεταφορά των επιμέρους αρχείων (περιεχόμενο).

Λειτουργική Κλήση και Περιγραφή των Εμπλεκόμενων Προγραμμάτων:

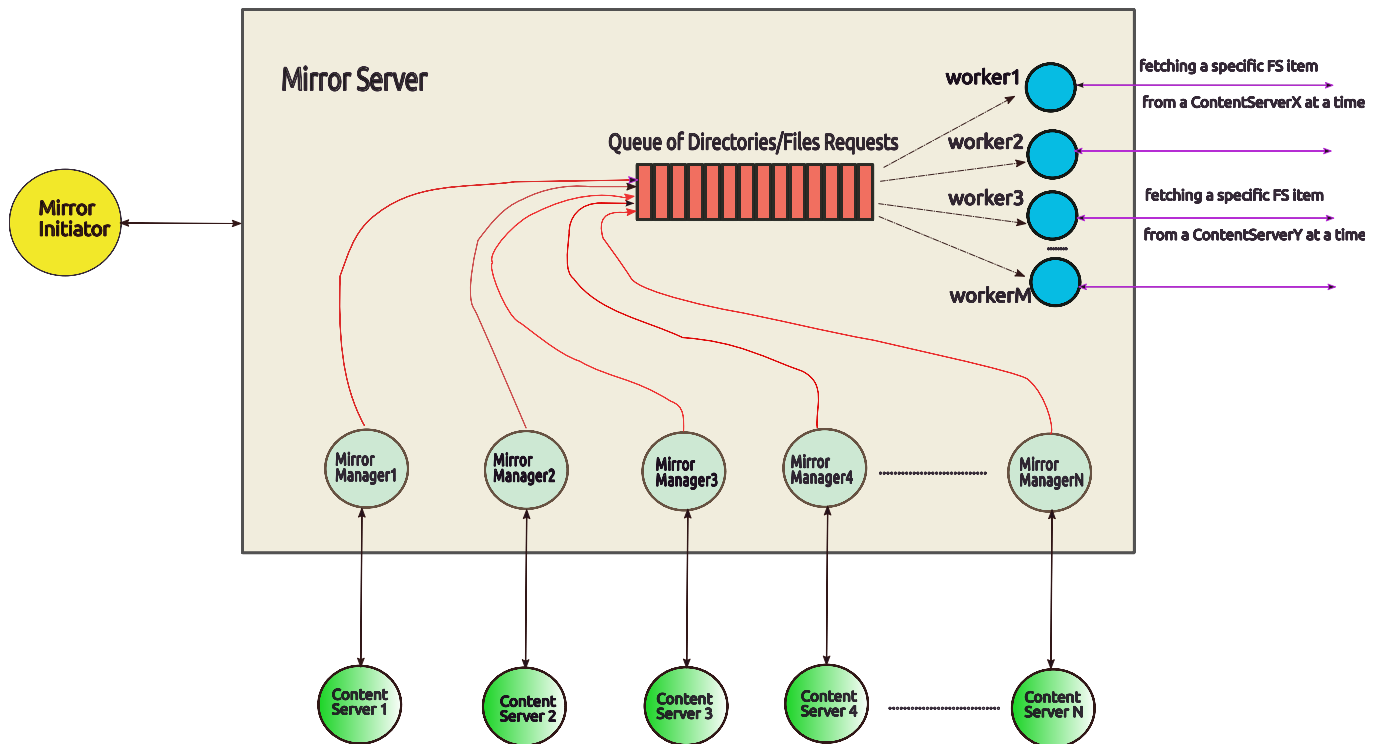
Ο MirrorInitiator συνδέεται με τον MirrorServer μέσω δικτύου και μεταφέρει τις τοποθεσίες των ContentServers των οποίων τους διαθέσιμους καταλόγους θα πρέπει να αντιγράψει ο MirrorServer.

Η κλήση του MirrorInitiator έχει ως εξής:

```
$ ./MirrorInitiator -n <MirrorServerAddress> -p <MirrorServerPort> \\  
-s <ContentServerAddress1:ContentServerPort1:dirorfile1:delay1, \\  
ContentServerAddress2:ContentServerPort2:dirorfile2:delay2, ...>
```

όπου:

1. MirrorServerAddress είναι η δικτυακή διεύθυνση της συσκευής όπου εκτελείται ο MirrorServer. Η παράμετρος μπορεί να είναι είτε όνομα είτε IP διεύθυνση, δηλαδή είτε linux23.di.uoa.gr είτε 195.134.65.24.
2. MirrorServerPort είναι ο αριθμός της θύρας στην οποία ακούει ο MirrorServer στην συσκευή MirrorServerAddress. Η παράμετρος είναι ακέραιος αριθμός.



Σχήμα 1: Επικοινωνία προγραμμάτων MirrorInitiator, MirrorServer, ContentServer(s).

3. `ContentServerAddressX` είναι η δικτυακή διεύθυνση μιας απομακρυσμένης συσκευής όπου εκτελείται ένας `ContentServer`. Η παράμετρος μπορεί να είναι είτε όνομα είτε `IP` διεύθυνση, δηλαδή είτε `linux23.di.uoa.gr` είτε `195.134.65.24`.
4. `ContentServerPortX` είναι ο αριθμός της θύρας στην οποία ακούει ο `ContentServer` στη συσκευή `ContentServerAddressX`. Η παράμετρος είναι ακέραιος αριθμός.
5. `dirorfileX` είναι το συμβολικό όνομα της οντότητας (κατάλογος ή αρχείο) που θα πρέπει αναδρομικά να ανακτηθεί και όλα της τα περιεχόμενα (και η δομή) να κατοπτριστούν στον `MirrorServer`.
6. `delayX` είναι η καθυστέρηση που θα χρησιμοποιήσει ο `ContentServer` στη συσκευή `ContentServerAddressX` κατά την αποστολή των αρχείων.

Στα παραπάνω υποθέτουμε ότι σαν χρήστες γνωρίζουμε που υπάρχουν `ContentServers` και ποιο είναι το (πιθανό) περιεχόμενο σε σχέση με το σύστημά αρχείων που ο κάθε ένας εξυπηρετεί. Σε περίπτωση που ο `ContentServer` δεν υφίσταται (δηλ. δεν τρέχει) ο χρήστης θα πρέπει να ενημερωθεί σχετικά. Επίσης όταν τερματιστεί η διαδικασία κατοπτρισμού, ο `MirrorInitiator` θα πρέπει να ενημερωθεί για τα στατιστικά της μεταφοράς π.χ. αριθμός μεταφερόμενων οντοτήτων συστήματος, συνολικό μέγεθος bytes που μεταφέρεται, μέσο μέγεθος και διασπορά μεγέθους αρχείων που έχουν μεταφερθεί καθώς και συνολικός αριθμός αρχείων/καταλόγων που έχουν μεταφερθεί.

Η κλήση του `ContentServer` έχει ως εξής:

```
$ ./ContentServer -p <port> -d <dirorfilename>
```

όπου:

1. `port` είναι ο αριθμός της θύρας στην οποία θα ακούει ο `ContentServer`.

2. `<dirorfilename>` είναι το `pathname` ενός καταλόγου ή/και αρχείου που διαθέτει η συσκευή προς αντιγραφή σε άλλες συσκευές.

Ο `ContentServer` θα δέχεται συνδέσεις και θα εξυπηρετεί δύο ειδών αιτήματα:

1. `LIST <ContentServerID> <delay>`

Είναι η αρχική επικοινωνία του `MirrorServer` με ένα συγκεκριμένο `ContentServer`. Η εντολή αυτή επιστρέφει τη λίστα των αρχείων/καταλόγων που ο `ContentServer` εξυπηρετεί και παρέχει στον `ContentServer` την `delay` στην οποία η μεταφορά περιεχομένου αρχείων και καταλόγων υπόκειται.

Μετά την αποστολή της λίστας των αρχείων, κλείνει η σύνδεση μεταξύ `ContentServer` και αντίστοιχου `MirrorManagerX`. Σημειώνεται ότι αποστέλλεται ιεραρχικά όλη η δομή των καταλόγων και των αρχείων που διατίθεται.

2. `FETCH <dirorfilename>`

Ο `ContentServer` παρέχει στο `worker` που αναζητεί το περιεχόμενο του `<dirorfilename>`. Αυτό συμβαίνει αφού ο `ContentServer` πρώτα θα καθυστερήσει την αποστολή του αρχείου/καταλόγου που ζητήθηκε κατά `delay seconds`. Σε περίπτωση που δεν υπάρχει τέτοιο αρχείο/κατάλογος, ένα μήνυμά λάθους επιστρέφεται.

Η κλήση της λειτουργίας του `MirrorServer` έχει ως εξής:

```
$ ./MirrorServer -p <port> -m <dirname> -w <threadnum>
```

όπου:

1. `port` είναι ο αριθμός της θύρας στην οποία θα ακούει ο `MirrorServer`.
2. `dirname` είναι το `pathname` του καταλόγου κάτω από τον οποίο θα αποθηκεύονται οι απομακρυσμένοι κατάλογοι.

Κάθε απομακρυσμένος κατάλογος θα αποθηκεύεται κάτω από τον κατάλογο `dirname/content_name/`, όπου το `content_name` θα προκύπτει από τη διεύθυνση και τη θύρα λειτουργίας του `ContentServer`, για παράδειγμα `linux03.di.uoa.gr_233`, όπου `linux03.di.uoa.gr` είναι η διεύθυνση και `233` είναι η θύρα λειτουργίας του συγκεκριμένου `ContentServer`.

3. `threadnum` είναι ο (σταθερός) αριθμός των νημάτων `workers` που δημιουργεί ο `MirrorServer`.

Ο `MirrorServer` δημιουργεί δυο ειδών νήματα: `MirrorManager` και `worker`. Τα `MirrorManager` νήματα δημιουργούνται με την άφιξη ενός αιτήματος κατοπτρισμού από το `MirrorInitiator` και γενικά υπάρχει ένα τέτοιο νήμα για κάθε `ContentServer`. Ο `MirrorServer` στην εκκίνηση του, δημιουργεί έναν αριθμό `workers` που λαμβάνεται ως όρισμα από τη γραμμή εντολής (`threadnum`).

Όταν ο `MirrorServer` δεχθεί ένα αίτημα κατοπτρισμού από τον πελάτη `MirrorInitiator`, δημιουργεί τόσα `MirrorManager` νήματα όσους είναι και οι `ContentServers` που αναφέρονται στο αίτημα. Τα νήματα `MirrorManager` δουλεύουν «παράλληλα» ως εξής: κάθε `MirrorManager` επικοινωνεί με ένα `ContentServer`. Θα στέλνει το αίτημα `LIST` με τις σχετικές παραμέτρους συμπεριλαμβανομένης και της επιθυμητής καθυστέρησης σε δευτερόλεπτα και θα λαμβάνει τη λίστα με τα αρχεία που εξυπηρετούνται.

Καθώς ο `MirrorManager` λαμβάνει τα αρχεία/καταλόγους, τα τοποθετεί σε ένα κοινό `buffer` που έχει πεπερασμένο μέγεθος. Τα στοιχεία (αιτήσεις) που ο εν λόγω `buffer` αποθηκεύει μπορεί να έχουν την μορφή:

`<dirorfilename, ContentServerAddress, ContentServerPort>`

Ο `buffer` διαμοιράζεται ανάμεσα σε όλα τα νήματα τύπου `MirrorManager` και `worker`. Όταν ένας `MirrorManager` τοποθετήσει και το τελευταίο `dirorfilename` στο `buffer` τερματίζει. Θα πρέπει να διαχειρίζεστε τον `buffer` σύμ-

φωνα με την λογική πολλαπλών–παραγωγών/πολλαπλών–καταναλωτών.

Τα νήματα `worker` θα δουλεύουν παράλληλα ως εξής: κάθε νήμα `worker` θα αφαιρεί ένα στοιχείο από τον `buffer` εφόσον βέβαια υπάρχει τέτοιο στοιχείο. Για κάθε στοιχείο που ανασύρει από τον `buffer`, ο `worker` συνδέεται στον κατάλληλο `ContentServer` και στέλνει ένα αίτημα `FETCH` με το `dirorfilename`. Μετά την ανάλογη καθυστέρηση που εισάγει ο `ContentServer`, λαμβάνει το αρχείο, βρίσκει το μέγεθος του και αυξάνει τις διαμοιραζόμενες μεταβλητές `bytesTransferred` και `filesTransferred`,

Ο `MirrorServer` θα χρησιμοποιεί μεταβλητές που οποιαδήποτε στιγμή μπορούν να δώσουν μια ακριβή εικόνα όσον αφορά στο πόσο έχει προχωρήσει μια μεταφορά. Για παράδειγμα η μεταβλητή `numDevicesDone` παρέχει τον τρέχοντα αριθμό συσκευών για τις οποίες έχει ολοκληρωθεί η μεταφορά. Αν η τιμή της `numDevicesDone` είναι ίση με τον αριθμό συσκευών από τους οποίους ζητά ο `MirrorInitiator` να γίνει η μεταφορά αρχείων, τότε το ‘τελευταίο ενεργό’ νήμα `worker` θα σηματοδοτήσει με την `condition variable allDone` το γεγονός ότι όλα τα αρχεία από όλες τις συσκευές που ζητήθηκαν από τον `MirrorInitiator` πελάτη έχουν μεταφερθεί και η επεξεργασία του αιτήματος έχει ολοκληρωθεί.

Ο `MirrorServer` θα επιστρέψει στον `MirrorInitiator` τα στατιστικά που υπάρχουν επαυξημένων με μέσο όρο μεγέθους μεταφορών και σχετική διασπορά.

Στα προγράμματα σας, κοινές μεταβλητές που μοιράζονται ανάμεσα σε πολλαπλά νήματα θα πρέπει να προστατεύονται με τη χρήση `mutexes`. Τονίζεται πως το *busy-waiting* δεν είναι αποδεκτή λύση για τη παραμονή πρόσβασης στον κοινό `buffer` ανάμεσα στους `MirrorManager` παραγωγούς και στους `worker` καταναλωτές.

Γενίκευση της Άσκησης:

Η παραπάνω αρχιτεκτονική μπορεί (με αλλαγές) να γενικευτεί ώστε ένα σύνολο από κόμβους να προσφέρει ταυτόχρονα όλες τις υπηρεσίες (να είναι δηλαδή και `ContentServer` και `MirrorServer`). Αν επιλέξετε και προχωρήσετε στην επιτυχή γενίκευση της υλοποίησής σας, θα λάβετε ένα επιπλέον 15% του βαθμού της εργασίας. Ωστόσο η πρώτη σας και κύρια επιμέλεια είναι να δώσετε λύση στην βασική έκδοση της προγραμματιστικής άσκησης.

Διαδικαστικά:

- Το πρόγραμμά σας θα πρέπει να γραφτεί σε *C* (ή *C++* αλλά χωρίς την χρήση έτοιμων δομών της *standard βιβλιοθήκης/STL*) και να τρέχει στα *LINUX workstations* του τμήματος.
- Το πρόγραμμα σας (*source code*) πρέπει να αποτελείται από **τουλάχιστον** δυο (και κατά προτίμηση πιο πολλά) διαφορετικά αρχεία. Η χρήση του *separate compilation* είναι *επιτακτική*!
- Η οργάνωση και η αναγνωσιμότητα (μαζί με την ύπαρξη σχολίων) του κώδικα περιλαμβάνονται στα κριτήρια βαθμολόγησης.
- Για επιπρόσθετες ανακοινώσεις, παρακολουθείτε το *forum* του μαθήματος στο *piazza.com*. Η πλήρης διεύθυνση είναι <https://piazza.com/uoa.gr/spring2017/k24/home>. Η παρακολούθηση του φόρουμ στο *Piazza* είναι υποχρεωτική.

Τι πρέπει να Παραδοθεί:

1. Μια σύντομη και περιεκτική εξήγηση για τις επιλογές που έχετε κάνει στο σχεδιασμό του προγράμματος σας (1-2 σελίδες *ASCII* κειμένου είναι αρκετές) σε αρχείο *README*.
2. Οποιαδήποτε πηγή πληροφορίας, συμπεριλαμβανομένου και κώδικα που μπορεί να βρήκατε στο Διαδίκτυο ή αλλού θα πρέπει να αναφερθεί και στον πηγαίο κώδικά σας αλλά και στο παραπάνω *README*.

3. Όλη η δουλειά σας σε ένα tar-file (OnomaEponymoProject3.tar) που να περιέχει όλα τα source files, header files, Makefile.

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 (οποιαδήποτε μορφής) είναι κάτι που δεν επιτρέπεται. Οποιοσδήποτε βρεθεί αναμεμιγμένος σε αντιγραφή κώδικά απλά παίρνει μηδέν στο μάθημα. Αυτό ισχύει για όσους εμπλέκονται ανεξάρτητα από το ποιος έδωσε/πήρε κλπ.