ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Κ24: Προγραμματισμός Συστήματος – Εαρινό Εξάμηνο 2012
4η Προγραμματιστική Εργασία
Ημερομηνία Ανακοίνωσης: 30/5/12
Ημερομηνία Υποβολής: 15/7/12

Εισαγωγή στην Εργασία:

Η εργασία αυτή θα σας βοηθήσει να εξοικειωθείτε με τον προγραμματισμό με threads και με την χρήση sockets για την επικοινωνία μεταξύ διεργασιών. Θα υλοποιήσετε μία πολυνηματική (*multithreaded*) δικτυακή εφαρμογή (*server*) που επιτρέπει σε χρήστες με την βοήθεια ενός κοινού client προγράμματος να ανεβάζουν αρχεία εικόνας στον server, όπου θα αποθηκεύονται σε μια συμπτυγμένη (*compact*) μορφή. Οι χρήστες θα μπορούν όχι μόνο να κατεβάζουν αρχεία εικόνας που ανέβασαν αλλά και να προχωρούν στη διαγραφή όποιων από αυτά επιθυμούν.

Διαδικαστικά:

Το αποτέλεσμα της εργασίας σας θα πρέπει να τρέχει στα μηχανήματα Linux/Unix της σχολής.

- Υπεύθυνοι για την άσκηση αυτή (ερωτήσεις, αξιολόγηση, βαθμολόγηση, κτλ) είναι: ο κ. Παναγιώτης Αδαμόπουλος (grad1127 at di domain), ο κ. Αλέξανδρος Αντωνιάδης (a.antoniadis at di domain), και ο κ. Γιάννης Γκερμπεσιώτης (j.gerbesiotis at di domain).
- Η άσκηση μπορεί να γίνει είτε *ατομικά* ή από ομάδες *δύο ατόμων*.
- Παρακολουθείτε την ιστοσελίδα του μαθήματος (`http://www.di.uoa.gr/~ad/k24/index.html`) για επιπρόσθετες ανακοινώσεις και την ηλεκτρονική λίστα (`mailman`).

Βασική περιγραφή της εργασίας:

Ο βασικός στόχος της άσκησης είναι η δημιουργία μιας διαδικτυακής υπηρεσίας που επιτρέπει σε χρήστες την αποθήκευση, ανάκτηση ή και διαγραφή αρχείων εικόνας χρησιμοποιώντας ένα απλό εργαλείο, όπως για παράδειγμα το `wget` (`man wget` για περισσότερες πληροφορίες).

Οι διαμοιραζόμενες (shared) εικόνες θα φιλοξενούνται σε ένα δικτυακό διακομιστή (server) με τον οποίο μπορούν να επικοινωνήσουν οι χρήστες μέσω ενός client προγράμματος. Ο χρήστης όποτε το επιθυμεί, μπορεί να αποστείλει μια αίτηση στον διακομιστή μέσω του client με το αρχείο εικόνας που θέλει να ανεβάσει. Ο διακομιστής αφού παραλάβει την αίτηση, αποθηκεύει το αρχείο και αποστέλλει στον χρήστη μια απάντηση επιβεβαίωσης μαζί με το μοναδικό id που δημιούργησε για αυτό. Ένας χρήστης για να παραλάβει τα περιεχόμενα ενός αρχείου εικόνας από τον server πρέπει να αποστείλει ένα μήνυμα στον διακομιστή μαζί με το id που αναζητά και να λάβει ως απάντηση τα περιεχόμενα του αρχείου εικόνας που ζήτησε. Εναλλακτικά, μπορεί να αποστείλει ένα αίτημα διαγραφής συνοδευόμενο από το αντίστοιχο id για να διαγραφεί η εικόνα από τον διακομιστή.

Σενάριο Λειτουργίας Πρωτοκόλλου Εφαρμογής:

Αρχικά ας υποθέσουμε ότι ένας client θέλει να ανεβάσει μία εικόνα στον server και γνωρίζει εκ των προτέρων την ip διεύθυνση και port που ακούει για νέες συνδέσεις. Τότε, ο client κάνοντας χρήση του προγράμματος wget και με την εντολή:

```
wget --post-file meme.jpg 50.18.252.53:9646
```

στέλνει μία αίτηση στον (απομακρυσμένο) server όπως παρακάτω (δηλ. ο server ‘βλέπει’ την παρακάτω αίτηση):

```
POST / HTTP/1.0\r\n
User-Agent: Wget/1.12 (linux-gnu)\r\n
Accept: */*\r\n
Host: 50.18.252.53:9646\r\n
Connection: Keep-Alive\r\n
Content-Type: application/x-www-form-urlencoded\r\n
Content-Length: <CONTENT LENGTH>\r\n
\r\n
<BINARY DATA>
```

όπου `<CONTENT LENGTH>` το μέγεθος του αρχείου που γίνεται upload σε bytes και `<BINARY DATA>` τα δεδομένα του αρχείου.

Η ακολουθία χαρακτήρων `\r\n` υποδηλώνει αλλαγή γραμμής και είναι ένας τρόπος να ξεχωρίζει το πρωτόκολλο επικοινωνίας τα πεδία μιας αίτησης ή μιας απάντησης που θα χρησιμοποιηθούν Ο server που επιθυμεί να επικοινωνήσει χρησιμοποιώντας αυτό το πρωτόκολλο, θα πρέπει επίσης να τερματίζει τις γραμμές του με την προαναφερθείσα ακολουθία χαρακτήρων και μόνο με αυτή. Ο καθορισμός μίας συγκεκριμένης ακολουθίας αλλαγής γραμμής έχει γίνει επειδή διαφορετικά λειτουργικά συστήματα χρησιμοποιούν διαφορετικές ακολουθίες αλλαγής γραμμής, για παράδειγμα τα Unix συστήματα χρησιμοποιούν μόνο έναν χαρακτήρα, τον `\n`. Ας σημειωθεί ότι η αίτηση που περιγράφηκε παραπάνω αποτελεί μία συνεχή ακολουθία χαρακτήρων, αλλά έχει τυπωθεί σε διαφορετικές γραμμές για λόγους αναγνωσιμότητας. Αποφύγετε λοιπόν να προσθέτετε επιπλέον χαρακτήρα αλλαγής γραμμής μετά την ακολουθία `\r\n`.

Συνεχίζοντας στα πλαίσια του παραπάνω παραδείγματος, αφού ο εξυπηρετητής server λάβει την αίτηση που του έστειλε ο client, θα απαντήσει με ένα μήνυμα ‘επιβεβαίωσης’. Κάτι τέτοιο φαίνεται παρακάτω:

```
HTTP/1.0 200 OK\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
Content-Type: text\r\n
Content-Length: <CONTENT LENGTH>\r\n
\r\n
File successfully uploaded with id : <ID>
```

Γενικά, τα μηνύματα του πρωτοκόλλου που χρησιμοποιήται για την επικοινωνία πελάτη και εξυπερετητή αποτελούνται από μία επικεφαλίδα (header) και προαιρετικά ένα σώμα (body). Η τελευταία γραμμή της επικεφαλίδας, όπως φαίνεται, είναι μία κενή γραμμή, όπου η ακολουθία `\r\n` υποδηλώνει το τέλος της επικεφαλίδας του μηνύματος. Με αυτό τον τρόπο καταλαβαίνουν οι συμμετέχοντες στην επικοινωνία ότι η αίτηση έφτασε στο τέλος της και δεν χρειάζεται να διαβάσουν επιπλέον πληροφορία. Ορισμένα μηνύματα απάντησης περιέχουν σώμα, όπως για παράδειγμα ένα μήνυμα αποστολής αρχείου εικόνας στο οποίο το σώμα του μηνύματος περιέχει τα δεδομένα του αρχείου εικόνας. Αλλά, όπως θα δούμε και παρακάτω, το σώμα σε ένα μήνυμα είναι προαιρετικό.

Για να ανακτήσουμε μία εικόνα εκτελούμε τη παρακάτω εντολή:

```
wget 50.18.252.53:9646/?id=ID
```

και ο τύπος μηνύματος που αποστέλλεται από τον πελάτη στον εξυπερετητή έχει ως εξής:

```
GET /?id=ID HTTP/1.0\r\n
User-Agent: Wget/1.12 (linux-gnu)\r\n
Accept: */*\r\n
Host: 50.18.252.53:9646\r\n
Connection: Keep-Alive\r\n
\r\n
```

όπου το <ID> είναι αυτό που επέστρεψε ο server μετά την επιτυχή αποστολή της εικόνας. Στη περίπτωση που διαθέτει το συγκεκριμένο αρχείο, ο server αποκρίνεται με ένα μήνυμα όπως το παρακάτω:

```
HTTP/1.0 200 OK\r\n
Connection: close\r\n
Content-Type: image/jpeg\r\n
Content-Length: <CONTENT LENGTH>\r\n
\r\n
<BINARY DATA>
```

Το παραπάνω μήνυμα απάντησης, όπως φαίνεται, αποτελείται από μία επικεφαλίδα, η πρώτη γραμμή της οποίας πληροφορεί τον client ότι ο απομακρυσμένος server διαθέτει το αρχείο που του ζητήθηκε. Στις επόμενες γραμμές περιλαμβάνονται πληροφορίες (metadata) σχετικές με το αρχείο. Μετά την κενή γραμμή αρχίζει το σώμα του μηνύματος, το οποίο περιέχει τα δεδομένα που ζητήθηκαν. Ο client ξέρει πόσα bytes να διαβάσει από το πεδίο `Content-Length` της επικεφαλίδας. Το πεδίο αυτό χρησιμοποιείται για να γνωρίζει ο τοπικός client σε ποιο σημείο τελειώνουν τα δεδομένα του σώματος του μηνύματος.

Μία ακόμα λειτουργία, είναι αυτή της διαγραφής των αρχείων του server από τον client, κατά την οποία ο πρώτος στέλνει ένα μήνυμα με το εργαλείο `wget`:

```
wget 50.18.252.53:9646/?d_id=ID
```

το μήνυμα της εντολής φαίνεται παρακάτω:

```
GET /d_id=2 HTTP/1.0\r\n
User-Agent: Wget/1.12 (linux-gnu)\r\n
Accept: */*\r\n
Host: 50.18.252.53:9646\r\n
Connection: Keep-Alive\r\n
\r\n
```

Όταν ο server λάβει ένα αρχείο από τον client, ενημερώνει κατάλληλα τις εσωτερικές δομές του, όπως θα εξηγήσουμε στη παρακάτω ενότητα. Τέλος, αποστέλλει ένα μήνυμα που έχει την παρακάτω μορφή:

```
HTTP/1.0 200 OK\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
Content-Type: text\r\n
Content-Length: <CONTENT LENGTH>\r\n
\r\n
File successfully uploaded with id : <ID>
```

Ενδέχεται κάποιος client να ζητήσει να κάνει κάτι ταυτόχρονα τη στιγμή που κάποιος άλλος client κάνει κάτι σχετικό. Η αίτηση αυτή πρέπει να μπλοκαριστεί έως ότου τελειώσει το νήμα που εξυπερετεί την πρώτη απαίτηση. Το μπλοκάρισμα θα γίνει με thread locks.

Σε περίπτωση που υπήρξε κάποιο πρόβλημα με κάποια αίτηση το οποίο αφορά λανθασμένη σύνταξη ή ασυνέπεια δεδομένων, θα υπάρξει μία απάντηση με ένα μήνυμα της μορφής:

```
HTTP/1.0 400 Bad request\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
r\n
```

Σε περίπτωση που ένας client ζητήσει ένα αρχείο από τον server αλλά δεν υπάρχει το αρχείο αυτό:

```
HTTP/1.0 404 Not Found\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
r\n
```

Επιπλέον, για οποιοδήποτε λόγο συμβεί κάποιο πρόβλημα στο server και δε μπορέσει να εξυπηρετήσει (π.χ. δεν έχει προλάβει να αρχικοποιηθεί ο server), ο client Θα πρέπει να ενημερωθεί με ένα μήνυμα:

```
HTTP/1.0 500 Internal Server Error\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
\r\n
```

Λειτουργία haystack server:

Ο haystack server αναλαμβάνει την εξυπηρέτηση των clients μέσω των εισερχόμενων αιτήσεων. Οι αιτήσεις αυτές αφορούν την αποθήκευση, ανάκτηση και επεξεργασία (διαγραφή) αρχείων.

Ο haystack server αποθηκεύει όλα τα αρχεία που έχουν κάνει upload οι clients (τα αρχεία αυτά θα τα αποκαλούμε needles) σε ένα μεγάλο αρχείο (το οποίο θα αποκαλούμε haystack). Κάθε νέα εισαγωγή ενός needle θα γίνεται στο τέλος του haystack (διαδικασία append).

Ένα αρχείο needle περιέχει τις εξής πληροφορίες σχετικά με το needle αρχείο που αποθηκεύεται:

<ID , κατάσταση αρχείου, μέγεθος αρχείου και δεδομένα αρχείου>

Οι πληροφορίες αυτές είναι αρκετές προκειμένου ο server να είναι ικανός να διαβάσει όλα τα needles από ένα haystack file. Η κατάσταση αρχείου υποδηλώνει αν το αρχείο είναι διεγραμμένο ή όχι και εξηγείται παρακάτω. Το ID είναι μοναδικό για κάθε needle αρχείο και παράγεται ακολουθιακά (sequential).

Η μορφή ενός haystack file είναι η εξής:

<SuperBlock, Needle1, Needle2, ..., NeedleN>

όπου SuperBlock είναι το block επικεφαλίδας του haystack file και μπορεί να περιέχει μεταδεδομένα για το αρχείο haystack τα οποία καθορίζουν ότι το αρχείο είναι τύπου haystack , πχ ένα magic number. Αυτό είναι απαραίτητο καθώς ο haystack server όταν ξεκινάει (bootstrapping) δέχεται σαν παράμετρο ένα αρχείο το οποίο είτε το διαβάζει στην περίπτωση που ήδη υπάρχει (αφού εξακριβώσει πρώτα ότι είναι τύπου haystack), είτε το δημιουργεί.

Για λόγους απόδοσης ο haystack server πρέπει να διατηρεί μια εσωτερική δομή όπου περιλαμβάνει πληροφορίες σχετικά με τη θέση των needles μέσα στο haystack file. Η δομή αυτή διατηρεί για κάθε αρχείο το ID του και το offset του μέσω στο haystack file, ενώ παίζει τον ρόλο του ευρετηρίου. Για τον λόγο αυτό πρέπει να είναι πλήρως ενημερωμένο και συνεπές ως προς το haystack αρχείο. Προφανώς, όταν ο server ξεκινάει με ένα ήδη δημιουργημένο haystack file η δομή αυτή πρέπει να ενημερώνεται προτού ο server ξεκινήσει να εξυπηρετεί αιτήσεις.

Η δομή αυτή απλοποιεί την διαδικασία εξυπηρέτησης αιτήσεων από μεριάς server. Αν ένας πελάτης ζητήσει την απόκτηση ή την διαγραφή ενός αρχείου (needle) αρκεί μια αναζήτηση σε αυτή τη δομή για να εξακριβωθεί αν το αρχείο (needle) αυτό υπάρχει ή όχι και αντίστοιχα, να συνεχίσει την εξυπηρέτηση στο haystack file ή να επιστρέψει το αντίστοιχο μήνυμα στον client.

Στην λειτουργία της διαγραφής ο server για λόγους απόδοσης απλά ενημερώνει την δομή διαγράφοντας την εγγραφή του αρχείου και αλλάζει την κατάσταση του αρχείου (needle) στο haystack file. Αυτό έχει σαν αποτέλεσμα την δημιουργία αχρησιμοποίητου (unused) χώρου μέσα στο haystack file. Για να λυθεί το πρόβλημα αυτό ο server στο ξεκίνημα του (bootstrapping) εκτελεί μια διαδικασία σύμπτυξης η οποία περιγράφεται παρακάτω, στην περίπτωση που το αρχείο που του δίνεται είναι ήδη δημιουργημένο.

Ο σκοπός στην διαδικασία σύμπτυξης (compaction) είναι η απαλοιφή του αχρησιμοποίητου χώρου που προέκυψαν στο haystack αρχείο από πιθανές διαγραφές. Το compaction γίνεται διαβάζοντας όλα τα needleαρχεία από ένα haystack fileκαι γράφοντας μόνο όσα δεν έχουν κατάσταση διαγραμμένου σε ένα νέο haystack file. Στην συνέχεια το παλιό haystack file διαγράφεται και την θέση του παίρνει το νέο.

Ο server πρέπει να μπορεί να τερματίσει από τον χρήστη όταν δεχτεί το σήμα (`CTRL+C`). Τη στιγμή που δεχθεί το σήμα τερματισμού πρέπει να σταματήσει να δέχεται νέες αιτήσεις και αφήνοντας ενεργές μόνο τις αιτήσεις που είχαν ήδη ξεκινήσει. Ο ομαλός αυτός τερματισμός γίνεται προκειμένου να αποφευχθούν πιθανά σφάλματα από ημιτελή γραψίματα στο haystack αρχείο.

Προδιαγραφές της άσκησης:

Όπως αναφέρθηκε και παραπάνω ο server σας θα πρέπει να υποστηρίζει τις παρακάτω λειτουργίες:

1. Ανέβασμα ενός αρχείου φωτογραφίας (jpeg) από έναν client
2. Κατέβασμα ενός αρχείου φωτογραφίας (jpeg) από έναν client
3. Διαγραφή ενός αρχείου φωτογραφίας (jpeg) από έναν client
4. Δημιουργία δομής ευρετηρίου των needle αρχείων κατά τη διαδικασία bootstrapping
5. Compaction του haystack αρχείου κατά τη διαδικασία bootstrapping
6. Ομαλός τερματισμός του server

Στις 3 πρώτες λειτουργίες είναι απαραίτητο να χειριστείτε τις πιθανές καταστάσεις (race condition) με thread locks για όλα τα πιθανά shared resources (haystack file , εσωτερική δομή κλπ.).

Κατευθύνσεις υλοποίησης:

Θα γράψετε κώδικα σε *C/C++* (χωρίς *STL*) που θα υλοποιεί τη συμπεριφορά του server . Το πρόγραμμα του server θα δέχεται στη γραμμή εντολής 2 ορίσματα που θα είναι τα ακόλουθα:

1. Το μονοπάτι (path), απόλυτο ή σχετικό προς το αρχείο haystack από το οποίο θα γίνει η αρχικοποίηση του server.
2. Η port όπου ακούει για TCP συνδέσεις ο server.

Ένα παράδειγμα έναρξης λειτουργίας του server, είναι το ακόλουθο:

```
haystack -p 9646 -f moo.hfs
```

Για τη λειτουργία εξυπηρέτησης του server, θα πρέπει να υιοθετηθεί το μοντέλο του multi-threaded server: Ο server κάθε φορά που δέχεται μία σύνδεση από κάποιον απομακρυσμένο client, θα αναθέτει την εξυπηρέτηση του κομματιού το οποίο αιτείται ο δεύτερος σε ένα νέο thread, το οποίο μπορεί να δημιουργηθεί εκ νέου ως detached thread.

Για να κατεβάσετε κάποιο αρχείο εικόνας θα πρέπει να κρατήσετε το ID το οποίο σας επέστρεψε ο server κατά το ανέβασμα της εικόνας.

Συνολικά, τα threads που θα χρειαστεί να διατηρήσετε δίνονται παρακάτω:

- Το main thread, το οποίο λειτουργεί ως manager για τον multi-threaded server.
- Multi-threaded server (detached threads) που λειτουργεί ως εξυπηρετητής για τις αιτήσεις που δέχεται.

Η εφαρμογή σας θα πρέπει να αγνοεί τυχόν corrupted files στη περίπτωση που το ανέβασμα της εικόνας δεν ολοκληρώθηκε επιτυχώς. Αν κάτι τέτοιο συμβεί, η εφαρμογή θα πρέπει να χειρίζεται κατάλληλα την περίπτωση αυτή έτσι ώστε τα corrupted files να μην γραφτούν σαν έγγυρο needle αρχείο αλλά και να μην γίνει corrupt το haystack αρχείο και τέλος, να ενημερώνει τον χρήστη με κατάλληλο μήνυμα λάθους.

Η εφαρμογή σας θα πρέπει να τυπώνει όλες τις πληροφορίες στο standard output υπό μορφή logs όπως θα δείτε και στο Appendix.

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).
2. Ένα tar file με όλη σας τη δουλειά σε έναν κατάλογο που πιθανώς να φέρει το όνομά σας και θα περιέχει όλη σας τη δουλειά.

Άλλες Σημαντικές Παρατηρήσεις:

1. Οι εργασίες είναι είτε **ατομικές** ή μπορούν να γραφτούν από ομάδες των **δυο ατόμων**.
2. Όποιος υποβάλλει/δείχνει κώδικα που δεν έχει γραφτεί από την ίδια/ίδιο **μηδενίζεται** στο μάθημα.
3. Αν και αναμένεται να συζητήσετε με φίλους και συνεργάτες το πώς θα επιχειρήσετε να δώσετε λύση στο πρόβλημα, αντιγραφή κώδικα (οποιαδήποτε 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 πρόγραμμα σας θα πρέπει να τρέχει σε Ubuntu-Linux ή Solaris αλλιώς **δεν θα βαθμολογηθεί**.
5. Σε καμιά περίπτωση τα MS-Windows **δεν είναι επιλογή** πλατφόρμας για την παρουσίαση αυτής της άσκησης.

Appendix: Σύνοψη του πρωτοκόλλου επικοινωνίας:

Όπως αναφέρθηκε, όλα τα μηνύματα διαθέτουν επικεφαλίδα της οποίας οι γραμμές διαχωρίζονται με την ακολουθία `\r\n` και η οποία τελειώνει με μία κενή γραμμή (η οποία αποτελείται μόνο από την ακολουθία `\r\n`). Τα μηνύματα που διαθέτουν σώμα, το μέγεθος του οποίου καθορίζεται από το πεδίο επικεφαλίδας `Content-Length`.

Παρακάτω παρουσιάζεται η γενική μορφή όλων των μηνυμάτων:

Upload

1. `OK`

```
HTTP/1.0 200 OK\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
Content-Type: text\r\n
Content-Length: <CONTENT LENGTH>\r\n
\r\n
File successfully uploaded with id : <ID>
```

2. `Request error`

```
HTTP/1.0 400 Bad request\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
\r\n
```

3. `Server Error`

```
HTTP/1.0 500 Internal Server Error\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
\r\n
```

Download

1. `OK`

```
HTTP/1.0 200 OK\r\n
Content-Type: image/jpeg\r\n
Content-Length: <CONTENT LENGTH>\r\n
Connection: close\r\n
\r\n
<BINARY DATA>
```

2. `Not Found`

```
HTTP/1.0 404 Not Found\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
\r\n
```

3. `Request error`

```
HTTP/1.0 400 Bad request\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
\r\n
```

4. `Server Error`

```
HTTP/1.0 500 Internal Server Error\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
\r\n
```

Delete

1. OK

```
HTTP/1.0 200 OK\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
Content-Type: text\r\n
Content-Length: <CONTENT LENGTH>\r\n
\r\n
File with id : ID was successfully deleted
```

2. Not Found

```
HTTP/1.0 404 Not Found\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
\r\n
```

3. Request error

```
HTTP/1.0 400 Bad request\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
\r\n
```

4. Server Error

```
HTTP/1.0 500 Internal Server Error\r\n
Server: haystack_server v1.0\r\n
Connection: close\r\n
\r\n
```

Log μηνύματα του server

Παρακάτω παρουσιάζονται παραδείγματα όλων των μηνυμάτων logs του server :

1. Bootstrapping

```
[DateTime] Server is starting. Working directory : WORKING_DIR
[DateTime] N haystack files found in WORKING_DIR
[DateTime] Creating in-memory datastructures...
[DateTime] X files-needles found in the haystack files
[DateTime] Server is ready for new clients!
```

2. Shutting down

```
[DateTime] Server is shutting down. Server will not accept any other clients
[DateTime] Waiting for running clients to be served...
[DateTime] Cleaning up...
[DateTime] Bye!
```

3. General Request Error

```
[DateTime] Client [XXX.ZZZ.YYY.ZZZ] request errror.
```

4. Upload

```
[DateTime] Client [XXX.ZZZ.YYY.ZZZ] requested a file upload with LENGTH content length
[DateTime] Client [XXX.ZZZ.YYY.ZZZ] file upload finished successfully
[DateTime] Client [XXX.ZZZ.YYY.ZZZ] file upload failed due to some internal server error
```

5. Get

```
[DateTime] Client [XXX.ZZZ.YYY.ZZZ] requested downloading file with id : ID
[DateTime] Client [XXX.ZZZ.YYY.ZZZ] successfully finished downloading file with
id : ID. Total bytes tranfered : BYTES.
[DateTime] Client [XXX.ZZZ.YYY.ZZZ] request errror.
[DateTime] Client [XXX.ZZZ.YYY.ZZZ] File with id : ID was not found
[DateTime] Client [XXX.ZZZ.YYY.ZZZ] File with id : ID could not be downloaded
due to some internal server error. DETAILS
```

6. Delete

```
[DateTime] Client [XXX.ZZZ.YYY.ZZZ] requested deletion of file with id : ID
[DateTime] Client [XXX.ZZZ.YYY.ZZZ] File with id : ID was successfully deleted
[DateTime] Client [XXX.ZZZ.YYY.ZZZ] request errror.
[DateTime] Client [XXX.ZZZ.YYY.ZZZ] File with id : ID was not found
[DateTime] Client [XXX.ZZZ.YYY.ZZZ] File with id : ID could not be deleted due
to some internal server error. DETAILS
```