Εισαγωγή

Στα πλαίσια αυτής της εργασίας θα υλοποιήσετε τρεις διαφορετικές, συνεργαζόμενες εφαρμογές: α) μία εφαρμογή σε κέλυφος η οποία δημιουργεί ένα σύνολο από web sites στο δίσκο β) έναν εξυπηρετητή (web server) ο οποίος δέχεται HTTP αιτήσεις για συγκεκριμένες ιστοσελίδες από τα web sites και γ) έναν crawler που ο στόχος του είναι να κατεβάσει όλα τα web sites από τον web server και ταυτόχρονα να εξυπηρετεί εντολές αναζήτησης. Οι εφαρμογές αυτές είναι παρόμοιες με τις αντίστοιχες που χρησιμοποιούν μηχανές αναζήτησης όπως το Google.

a) Web site creator

Για το μέρος αυτό της άσκησης θα υλοποιήσετε μια εφαρμογή κελύφους η οποία θα χρησιμοποιείται ως εξής:

```
./webcreator.sh root_directory text_file w p
```

- `root_directory`: είναι ένα directory το οποίο πρέπει ήδη να υπάρχει και στο οποίο θα δημιουργηθούν τα web sites
- `text_file`: είναι ένα αρχείο κειμένου από το οποίο θα επιλεγούν τυχαίες λέξεις για τη δημιουργία των ιστοσελίδων. Ένα τέτοιο αρχείο είναι οι "20,000 Λεύγες κάτω από τη θάλασσα" του Ι. Βερν.
- `w`: ο αριθμός των web sites που θα δημιουργηθούν
- `p`: ο αριθμός των ιστοσελίδων ανά web site που θα δημιουργηθούν

Η εφαρμογή `webcreator.sh` θα λειτουργεί ως εξής: αρχικά θα πρέπει να ελέγχει πως ο κατάλογος και το αρχείο που δόθηκαν ως είσοδος υπάρχουν και πως οι παράμετροι `w` και `p` είναι ακέραιοι αριθμοί. Θα πρέπει επίσης να ελέγχει πως το αρχείο `text_file` έχει τουλάχιστον 10,000 γραμμές. Κατόπιν θα δημιουργεί μέσα στο `root_directory` `w` καταλόγους που κάθε ένας αναπαριστά ένα web site. Οι κατάλογοι αυτοί θα έχουν τη μορφή `site0, site1, ... , sitew-1`. Κατόπιν θα πρέπει να δημιουργεί `p` σελίδες μέσα σε κάθε κατάλογο που θα αναπαριστούν τις ιστοσελίδες του κάθε web site. Οι ιστοσελίδες θα έχουν το όνομα `pagei_j.html` όπου `i` είναι το web site και `j` είναι ένας τυχαίος, μοναδικός για το εκάστοτε web site, αριθμός.
Για παράδειγμα αν `w = 3` και `p = 2` μπορούν να δημιουργηθούν οι εξής κατάλογοι και σελίδες:

```
root_dir/site0/
root_dir/site0/page0_2345.html
root_dir/site0/page0_2391.html
root_dir/site1/
root_dir/site1/page1_2345.html
root_dir/site1/page1_2401.html
root_dir/site2/
root_dir/site2/page2_6530.html
root_dir/site2/page2_16672.html
```

Το περιεχόμενο κάθε ιστοσελίδας θα πρέπει να είναι HTML και θα έχει την εξής μορφή:

```
<!DOCTYPE html>
<html>
   <body>
   word1 word2 … <a href="link1">link1_text</a>
   wordn wordn+1 … <a href="link2">link2_text</a> wordm wordm+1 ...
   </body>
</html>
```

Το περιεχόμενο της κάθε ιστοσελίδας θα δημιουργείται με **τυχαίο τρόπο** χρησιμοποιώντας σαν είσοδο το αρχείο `text_file` ως εξής:

1. Επιλέγουμε έναν τυχαίο αριθμό 1 < k < #lines in text_file - 2000
2. Επιλέγουμε έναν τυχαίο αριθμό 1000 < m < 2000
3. Δημιουργούμε ένα σύνολο μεγέθους f = (p / 2) + 1 με τα ονόματα f τυχαίων ιστοσελίδων (εκτός της ίδιας ιστοσελίδας) μέσα στο ίδιο web site που ανήκει η σελίδα που δημιουργούμε. Αυτά θα είναι τα εσωτερικά links.
4. Δημιουργούμε ένα σύνολο μεγέθους q = (w / 2) + 1 με τα ονόματα q τυχαίων ιστοσελίδων προς άλλα web sites. Αυτά είναι εξωτερικά links.
5. Γράφουμε στο αρχείο τους αρχικούς headers της HTML.
6. Αρχίζοντας από τη γραμμή k του text_file, αντιγράφουμε m / (f + q) γραμμές στο αρχείο της ιστοσελίδας που δημιουργούμε και γράφουμε ένα από τα f + q links σε HTML.
7. Επαναλαμβάνουμε το 6 με τις επόμενες m / (f+q) γραμμές και το επόμενο link μέχρι να γραφούν όλες οι γραμμές και όλα τα links.
8. Γράφουμε τα τελικά HTML headers.

Το script θα πρέπει να φροντίσει να κάνει purge (και να τυπώσει το κατάλληλο μήνυμα) αν το root_directory δεν είναι άδειο. Επίσης θα πρέπει να τυπώνει κάποια μηνύματα σχετικά με το τι κάνει για κάθε web site και κάθε ιστοσελίδα. Κατά το ελάχιστο θα πρέπει να τυπώνει κάτι σαν το εξής (το φορμάτ είναι ελεύθερο, μπορείτε να το αλλάξετε και να τυπώνετε περισσότερες πληροφορίες). Δεν υπάρχει βαθμολογική ποινή για περισσότερη πληροφορία από την παρακάτω.

```
# Warning: directory is full, purging …
# Creating web site 0 …
#  Creating page root_dir/site0/page0_1234.html with 1249 lines starting at line 2094 …
#  Adding link to root_dir/site0/page0_135.html
#  Adding link to root_dir/site0/page0_485.html
# …
#  Adding link to root_dir/site1/page1_8857.html
# …
# Creating web site 1 …
# …
# All pages have at least one incoming link
# Done.
```

Στο τέλος του script θα πρέπει να τυπώσετε αν όλες οι σελίδες έχουν τουλάχιστον έναν εισερχόμενο σύνδεσμο ή όχι.

b) Web server

Σε αυτό το κομμάτι θα υλοποιήσετε έναν απλό HTTP server σε C/C++. Ο server αυτός θα τρέχει ως εξής:

```
./myhttpd -p serving_port -c command_port -t num_of_threads -d root_dir
```

- `serving_port`: είναι το port στο οποίο ακούει ο server προκειμένου να επιστρέψει ιστοσελίδες
- `command_port`: είναι το port στο οποίο ακούει ο server προκειμένου να του δώσουμε οδηγίες.
- `num_of_threads`: είναι ο αριθμός των threads που θα δημιουργήσει ο server προκειμένου να διαχειριστεί τα requests που έρχονται. Τα threads αυτά δημιουργούνται όλα μαζί στην αρχή και βρίσκονται σε ένα thread pool και από εκεί ο server τα επαναχρησιμοποιεί. Στην περίπτωση που κάποιο thread τερματίσει ο server πρέπει να ξεκινήσει ένα καινούργιο.

- `root_dir`: είναι το directory που έχει μέσα όλα τα web sites που δημιουργήθηκαν από το webcreator.sh.

Ένα παράδειγμα εκτέλεσης του server είναι το εξής:

```
./myhttpd -p 8080 -c 9090 -t 4 -d /home/users/k24/www/
```

Στο service port ο server δέχεται απλά HTTP/1.1 requests. Το πρωτόκολλο HTTP είναι αρκετά μεγάλο [RFC 2616]. Στην παρούσα άσκηση χρειάζεται να υλοποιήσετε **μόνο** ένα απλό GET. Ο server θα ακούει στο συγκεκριμένο port για requests της μορφής:

```
GET /site0/page0_1244.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
[blank line here]
```

Όταν πάρει ένα τέτοιο request, ο server το αναθέτει σε ένα από τα threads από το thread pool που έχει. Η δουλειά του thread είναι να επιστρέψει το αρχείο `root_dir/site0/page0_1244.html` πίσω σε αυτόν που το ζήτησε. Το `Connection:` header μπορείτε να το αγνοήσετε και να κλείσετε το connection ανεξάρτητα από το αν λέει keep-alive ή close. Η λειτουργία του server με τα threads γίνεται ως εξής: ο server αποδέχεται ένα connection στο socket και τοποθετεί τον αντίστοιχο file descriptor σε ένα buffer από τον οποίο διαβάζουν τα threads. Κάθε thread είναι υπεύθυνο για ένα file descriptor από το οποίο θα διαβάσει το GET request και θα επιστρέψει την απάντηση, δηλαδή το περιεχόμενο της ιστοσελίδας (ή μια διαφορετική απάντηση όπως παρακάτω στην περίπτωση που το αρχείο δεν υπάρχει ή δεν έχει το thread τα κατάλληλα permissions).

Αν το αρχείο που ζητείται υπάρχει, και ο server έχει δικαίωμα να το διαβάσει τότε επιστρέφεται μια απάντηση της μορφής (το length είναι σε bytes μόνο για το μέγεθος του περιεχόμενου, δηλ. χωρίς το header):

```
HTTP/1.1 200 OK
Date: Mon, 27 May 2018 12:28:53 GMT
Server: myhttpd/1.0.0 (Ubuntu64)
Content-Length: 8873
Content-Type: text/html
Connection: Closed
[blank line]
[content of the requested file here… e.g. <html>hello one two …</html>]
```

Αν το αρχείο που ζητείται δεν υπάρχει τότε ο server πρέπει να επιστρέψει μια απάντηση της μορφής:

```
HTTP/1.1 404 Not Found
Date: Mon, 27 May 2018 12:28:53 GMT
Server: myhttpd/1.0.0 (Ubuntu64)
Content-Length: 124
Content-Type: text/html
Connection: Closed
[blank line]
<html>Sorry dude, couldn't find this file.</html>
```

Αν ο server δεν έχει permissions για το αρχείο τότε η απάντηση είναι της μορφής:

```
HTTP/1.1 403 Forbidden
Date: Mon, 27 May 2018 12:28:53 GMT
Server: myhttpd/1.0.0 (Ubuntu64)
Content-Length: 124
Content-Type: text/html
Connection: Closed
[blank line]
<html>Trying to access this file but don't think I can make it.</html>
```

Τέλος στο command port ο server ακούει και δέχεται τις εξής απλές εντολές (1 λέξη η κάθε μία) οι οποίες εκτελούνται απευθείας από το server χωρίς να χρειάζεται να ανατεθούν σε κάποιο thread:

`STATS`: ο server απαντάει με το πόση ώρα τρέχει, πόσες σελίδες έχει επιστρέψει και το συνολικό αριθμό bytes. Π.χ.:

```
Server up for 02:03.45, served 124 pages, 345539 bytes
```

`SHUTDOWN`: ο server σταματάει να εξυπηρετεί επιπρόσθετα requests, απελευθερώνει όποια μνήμη (κοινή ή μη) έχει δεσμεύσει και σταματάει την εκτέλεσή του.

c) Web crawler

Δουλειά του crawler (σε C/C++) είναι να κατεβάσει όλα τα web sites από το web server κατεβάζοντας σελίδες, αναλύοντάς τες και βρίσκοντας links μέσα στις σελίδες που θα πρέπει να ακολουθήσει αναδρομικά. Ο crawler θα πρέπει να τρέχει ως εξής:

```
./mycrawler -h host_or_IP -p port -c command_port -t num_of_threads -d save_dir starting_URL
```

- `host_or_ip`: είναι το όνομα του μηχανήματος ή η IP στην οποία τρέχει ο server
- `port`: είναι το port στο οποίο ακούει ο server
- `command_port`: είναι το port στο οποίο ακούει ο crawler προκειμένου να του δώσουμε οδηγίες
- `num_of_threads`: είναι ο αριθμός των threads που τρέχει ο crawler. Στην περίπτωση που κάποιο thread τερματίσει ο crawler πρέπει να ξεκινήσει ένα καινούργιο.
- `save_dir`: είναι το directory στο οποίο ο crawler θα αποθηκεύσει τις σελίδες που κατεβάζει.

Ουσιαστικά μετά το πέρας της εκτέλεσης του crawler (και αν θεωρήσουμε πως όλες οι σελίδες του server είναι προσπελάσιμες με κάποιο link) το save_dir θα πρέπει να είναι ένα ακριβές αντίγραφο του root_dir.

- `starting_URL`: είναι το URL από το οποίο ξεκινάει ο crawler

Ο crawler λειτουργεί ως εξής:

1. Ξεκινώντας, δημιουργεί ένα thread pool με τα αντίστοιχα threads. Τα threads επαναχρησιμοποιούνται. Δημιουργεί επίσης μια ουρά στην οποία θα αποθηκεύει τα links που έχουν βρεθεί ως τώρα και βάζει το starting_URL σε αυτή την ουρά.
2. Κάποιο από τα threads παίρνει το URL από την ουρά και το ζητάει από το server. Το URL είναι της μορφής (για παράδειγμα) `http://linux01.di.uoa.gr:8080/site1/page0_1234.html`.
3. Αφού κατεβάσει το αρχείο το σώζει στον αντίστοιχο κατάλογο/αρχείο μέσα στο save_dir.
4. Αναλύει το αρχείο που μόλις κατέβηκε και βρίσκει και άλλα links τα οποία τα βάζει στην ουρά.

5. Επαναλαμβάνεται η διαδικασία από το 2 με όλα τα threads παράλληλα έως ότου δεν υπάρχουν άλλα links στην ουρά.

Στο command port ο crawler ακούει και δέχεται τις εξής απλές εντολές (1 λέξη η κάθε μία) οι οποίες εκτελούνται απευθείας από τον crawler χωρίς να χρειάζεται να ανατεθούν σε κάποιο thread:

`STATS`: ο crawler απαντάει με το πόση ώρα τρέχει, πόσες σελίδες έχει μαζέψει και το συνολικό αριθμό bytes. Π.χ.:

```
Crawler up for 01:06.45, downloaded 124 pages, 345539 bytes
```

`SEARCH word1 word2 word3 … word10`
Εάν ο crawler έχει ακόμα σελίδες στην ουρά τότε επιστρέφει ένα μήνυμα που υποδηλώνει πως το crawling είναι in-progress.

Εάν ο crawler έχει τελειώσει το κατέβασμα όλων των ιστοσελίδων, τότε αυτή η εντολή θα πρέπει να επιστέψει τα αποτελέσματα της Άσκησης 2 που έχετε κάνει, αλλά πάνω από το socket. Πιο συγκεκριμένα θα πρέπει να ενσωματώσετε τον κώδικα του job executor μέσα στον crawler με τις κατάλληλες παραμέτρους για τα αρχεία, να τρέξετε την εντολή search και να επιστρέψετε το αποτέλεσμα πάνω από το socket για το συγκεκριμένο query που έδωσε ο χρήστης. Οι workers θα πρέπει να ξεκινήσουν μία μόνο φορά. Επίσης κατά την αναζήτηση/δεικτοδότηση θα πρέπει να αγνοηθούν τα HTML tags.

`SHUTDOWN`: ο crawler σταματάει να ζητάει επιπρόσθετες σελίδες, απελευθερώνει όποια μνήμη (κοινή ή μη) έχει δεσμεύσει και σταματάει την εκτέλεσή του.

Παρατηρήσεις

- Η συγκεκριμένη εργασία απαιτεί αρκετή σκέψη και καλό σχεδιασμό όσον αφορά κατανεμημένους πόρους, locks στις ουρές, στα αρχεία κλπ. Η άσκηση δεν περιγράφει όλες τις λεπτομέρειες και τις δομές για τον απλό λόγο πως οι σχεδιαστικές επιλογές είναι αποκλειστικά δικές σας (βεβαιωθείτε φυσικά πως τις περιγράφετε αναλυτικά στο README). Αν έχετε διάφορες επιλογές για κάποιο σημείο της άσκησης σκεφτείτε τα υπέρ και τα κατά, τεκμηριώστε τα στο README, επιλέξτε αυτό που θεωρείτε σωστό και λογικό και περιγράψτε γιατί το επιλέξατε στο README.
- Στον creator και καθ' όλη τη διάρκεια του debugging ίσως είναι χρήσιμο να φτιάξετε πλήρεις γράφους, δηλαδή f = p και q = w.
- Στα αποτελέσματα της search για τον crawler που χρησιμοποιούν την άσκηση 2 δεν είναι απαραίτητο να βελτιώσετε την άσκηση 2. Ο βαθμός σας μπορεί να είναι 100% για τη συγκεκριμένη άσκηση αν απλά ενσωματώσετε τον κώδικα της άσκησης 2 και επιστρέφει ό,τι επιστρέφει και η άσκηση 2. Είστε ελεύθεροι να βελτιώσετε την άσκηση 2 προκειμένου να υλοποιήσετε κάτι που δεν μπορέσατε, ή που δεν δούλευε.
- Αν υλοποιήσετε σωστά το απλό HTTP του server μπορείτε να συνδεθείτε με έναν web browser για να βλέπετε τι επιστρέφει για να κάνετε debug. Μπορείτε επίσης να χρησιμοποιήσετε το telnet είτε για το HTTP είτε για τις εντολές των crawler και server.

Παραδοτέα

- Μια σύντομη και περιεκτική εξήγηση για τις επιλογές που έχετε κάνει στο σχεδιασμό του προγράμματός σας. 1-2 σελίδες ASCII κειμένου είναι αρκετές. Συμπεριλάβετε την εξήγηση και τις οδηγίες για το compilation και την εκτέλεση του προγράμματός σας σε ένα αρχείο README μαζί με τον κώδικα που θα υποβάλετε.

- Οποιαδήποτε πηγή πληροφορίας, συμπεριλαμβανομένου και κώδικα που μπορεί να βρήκατε στο Διαδίκτυο ή αλλού θα πρέπει να αναφερθεί στον πηγαίο κώδικά σας αλλά και στο παραπάνω README.
- Όλη η δουλειά σας (πηγαίος κώδικας, Makefile και README) σε ένα tar.gz file με ονομασία OnomaEponymoProject3.tar.gz. Προσοχή να υποβάλετε μόνο κώδικα, Makefile, README και όχι τα binaries.

Διαδικαστικά

- Για επιπρόσθετες ανακοινώσεις, παρακολουθείτε το forum του μαθήματος στο piazza.com. Η πλήρης διεύθυνση είναι https://piazza.com/uoa.gr/spring2018/k24/home. Η παρακολούθηση του φόρουμ στο Piazza είναι υποχρεωτική.
- Το πρόγραμμά σας θα πρέπει να γραφεί σε C (ή C++). Στην περίπτωσή που χρησιμοποιήσετε C++ δεν μπορείτε να χρησιμοποιήσετε τις έτοιμες δομές της Standard Template Library (STL). Σε κάθε περίπτωση το πρόγραμμά σας θα πρέπει να τρέχει στα Linux workstations του Τμήματος.
- Ο κώδικάς σας θα πρέπει να αποτελείται από τουλάχιστον δύο (και κατά προτίμηση περισσότερα) διαφορετικά αρχεία. Η χρήση του separate compilation είναι επιτακτική και ο κώδικάς σας θα πρέπει να έχει ένα Makefile.
- Βεβαιωθείτε πως ακολουθείτε καλές πρακτικές software engineering κατά την υλοποίηση της άσκησης. Η οργάνωση, η αναγνωσιμότητα και η ύπαρξη σχολίων στον κώδικα αποτελούν κομμάτι της βαθμολογίας σας.

Άλλες σημαντικές παρατηρήσεις

- Οι εργασίες είναι ατομικές.
- Όποιος υποβάλει / παρουσιάσει κώδικα που δεν έχει γραφτεί από την ίδια/τον ίδιο μηδενίζεται στο μάθημα.
- Αν και αναμένετε να συζητήσετε με φίλους και συνεργάτες το πώς θα επιχειρήσετε να δώσετε λύση στο πρόβλημα, αντιγραφή κώδικα (οποιασδήποτε μορφής) είναι κάτι που δεν επιτρέπεται. Οποιοσδήποτε βρεθεί αναμεμιγμένος σε αντιγραφή κώδικα απλά παίρνει μηδέν στο μάθημα. Αυτό ισχύει για όσους εμπλέκονται ανεξάρτητα από το ποιος έδωσε/πήρε κλπ.
- Οι ασκήσεις προγραμματισμού μπορούν να δοθούν με καθυστέρηση το πολύ 3 ημερών και με ποινή 5% για κάθε μέρα αργοπορίας. Πέραν των 3 αυτών ημερών, δεν μπορούν να κατατεθούν ασκήσεις.