Άσκηση 3
Προγραμματισμός Συστήματος

Προθεσμία: 24 Μαΐου 2015

Ο στόχος της εργασίας είναι να εξοικειωθείτε με τη χρήση νημάτων και τη δικτυακή επικοινωνία. Η άσκηση είναι παραλλαγή της Άσκησης 1, με χρήση μόνο bloom filter (όχι δένδρου), αλλά με παραλληλισμό σε επίπεδο νημάτων για πιο γρήγορη αναζήτηση, καθώς και την ικανότητα να κλωνοποιηθεί η κατάσταση της αναζήτησης σε νέο κόμβο.

Όπως στην Άσκηση 1, καλείστε να αποσπάσετε μια «κρυφή λέξη» από έναν «μάντη». Η επικοινωνία σας με το μάντη περιορίζεται στην ανταλλαγή λέξεων (οι οποίες αναπαρίστανται ως συμβολοσειρές). Όταν μια λέξη δοκιμαστεί και δεν είναι η κρυφή, ο μάντης επιστρέφει μία λίστα από λέξεις για να δοκιμαστούν στη συνέχεια.

Το πρόγραμμα σας, θα πρέπει να δοκιμάζει αναζητήσεις που ξεκινούν από πολλές αρχικές λέξεις ταυτόχρονα (παράλληλα). Οι παράλληλες αναζητήσεις θα υλοποιηθούν με χρήση ξεχωριστών νημάτων (threads). Το κάθε νήμα πραγματοποιεί ανεξάρτητη αναζήτηση, χωρίς όμως να δοκιμάσει λέξεις που έχει τύχει να δοκιμάσουν ήδη άλλα νήματα. Δηλαδή αν ένα νήμα δοκιμάσει μια λέξη και ο μάντης επιστρέψει μια σειρά πιθανές επόμενες, το ίδιο νήμα θα δοκιμάσει και αυτές μία-μία, θα επιστρέψουν πιθανόν άλλες επόμενες τις οποίες το ίδιο νήμα θα δοκιμάσει κ.ο.κ.

Το παιχνίδι τελειώνει όταν κάποια από τις παράλληλες αναζητήσεις βρει τη σωστή λέξη, οπότε και το πρόγραμμά σας θα πρέπει να τερματίζει, με κάθε νήμα να παράγει μια αναφορά στατιστικών από την ιδιωτική του αναζήτηση.

Η υλοποίηση του μάντη θα σας δοθεί σε binary μορφή, ενώ θα έχει το εξής interface:

```
const char **oracle(const char *word);
```

Συγκεκριμένα, το όρισμα `word` είναι η λέξη που επιλέγετε να δοκιμάσετε, και το αποτέλεσμα είναι είτε `NULL` (που σημαίνει ότι το word ήταν η κρυφή λέξη) είτε ένας πίνακας από λέξεις/συμβολοσειρές, αγνώστου μεγέθους, ο οποίος θα είναι πάντα τερματισμένος με `NULL`.

Σε αντίθεση με την Άσκηση 1, η συνάρτηση `oracle()` είναι πλέον `thread-safe` (δεν επιστρέφει στατική μνήμη) και δεν χρειάζεται να αντιγράψετε τις δομές που επιστρέφει. Η μνήμη που επιστρέφει η `oracle()` δεσμεύεται με allocator που θα της περάσετε εσείς μέσω της παρακάτω ρουτίνας:

```
void initAlloc(void *(*allocationFunction)(size_t));
```

Δηλαδή εσείς θα περάσετε στον μάντη τη ρουτίνα `malloc` που χρησιμοποιεί το πρόγραμμά σας, έτσι ώστε η μνήμη που σας επιστρέφει η `oracle()` να μπορεί να ελευθερωθεί από το δικό σας πρόγραμμα με απλό `free`.

Όπως και στην πρώτη άσκηση, προκειμένου να μην εξετάζετε συνεχώς τις ίδιες λέξεις, θα χρησιμοποιήσετε την τεχνική ανίχνευσης στοιχείων σε σύνολο που ονομάζεται bloom filter. Το bloom filter κρατάει μια υπερ-εκτίμηση των λέξεων που έχετε συναντήσει μέχρι στιγμής (σε οποιοδήποτε νήμα). Ωστόσο, θα πρέπει να εξασφαλίσετε ότι κάθε νήμα αποκτά αποκλειστική πρόσβαση στο bloom filter, ώστε κάθε φορά να αλλάζει τα δεδομένα με ασφαλή τρόπο. Περισσότερες λεπτομέρειες για την πολυνηματική υλοποίηση του προγράμματός σας θα ακολουθήσουν σε επόμενη ενότητα.

Τέλος, καλείστε να υλοποιήσετε και μία επιπλέον λειτουργία αντιγραφής του bloom filter. Συγκεκριμένα, ένα ξεχωριστό νήμα θα λειτουργεί ως εξυπηρετητής (server) και θα περιμένει για εξωτερικές συνδέσεις. Σε κάθε πελάτη (client) που συνδέεται, θα αποστέλλει όλες τις παραμέτρους εκτέλεσης, καθώς και το ίδιο το bloom filter (πλήρη περιεχόμενα). Ο πελάτης θα πρέπει στη συνέχεια να ξεκινάει αναζήτηση γνωρίζοντας ποιες λέξεις έχουν ήδη δοκιμαστεί από το αρχικό πρόγραμμα.

Bloom Filter – Περιγραφή Δομής. Ένα bloom filter (βλ. `http://en.wikipedia.org/wiki/Bloom_filter`) είναι μία συμπαγής δομή η οποία χρησιμοποιείται για τον έλεγχο της ύπαρξης ενός στοιχείου σε ένα σύνολο. Αναλυτικότερα, χρησιμοποιώντας το bloom filter για ένα δεδομένο στοιχείο, μπορούμε να εξάγουμε ένα από τα παρακάτω συμπεράσματα. Είτε ότι (i) το στοιχείο ***σίγουρα δεν ανήκει*** στο σύνολο, ή ότι (ii) το στοιχείο είναι ***πιθανόν να ανήκει*** στο σύνολο.

Δηλαδή το bloom filter μπορεί να απαντήσει θετικά ότι ένα στοιχείο ανήκει στο σύνολο, ακόμα κι όταν δεν ανήκει (false positive). Αντίθετα, η αρνητική απάντηση είναι πάντα έγκυρη. Με αυτόν το τρόπο, το bloom filter κάνει μία υπερεκτίμηση των στοιχείων του συνόλου. Η πιθανότητα λάθους (σε περίπτωση θετικής απάντησης) αυξάνεται όσο προστίθενται στοιχεία στο σύνολο.

Ένα bloom filter αναπαρίσταται ως ένας πίνακας από bits M θέσεων (αρχικοποιημένος με 0), και συνοδεύεται από K hash functions. Ακολουθούν οι βασικές πράξεις που υποστηρίζει και το πως υλοποιούνται:

1. Για να εισάγουμε ένα στοιχείο e στο σύνολο, εφαρμόζουμε κάθε μία από αυτές τις K hash functions στο εν λόγω στοιχείο. Αυτές θα επιστρέψουν K θέσεις στον πίνακα (που κάποιες πιθανόν να συμπίπτουν). Έπειτα, σε κάθε μία από αυτές τις θέσεις θέτουμε το αντίστοιχο bit σε 1.

2. Για να ελέγξουμε αν ένα στοιχείο e ανήκει στο σύνολο, εφαρμόζουμε κάθε μία από αυτές τις K hash functions στο εν λόγω στοιχείο. Αν σε κάθε μία από τις K θέσεις που μας επιστρέφουν υπάρχουν 1, τότε το στοιχείο e (πιθανόν) να ανήκει στο σύνολο. Διαφορετικά (αν υπάρχει έστω κι ένα 0), σίγουρα το στοιχείο δεν ανήκει στο σύνολο.

Παράδειγμα Έστω $M = 16, K = 3$, και $hash_1(), hash_2(), hash_3()$ οι τρεις hash functions που σας έχουν δοθεί. Ο πίνακας των bits αρχικά θα είναι: $\underbrace{00\,00\,00\,00\;00\,00\,00\,00}_{16}$.

Έστω ότι εισάγουμε διαδοχικά τα στοιχεία s_1, s_2, s_3, για τα οποία έχουμε:

$hash_1(s_1) = 3 \bmod 16$	$hash_1(s_2) = 10 \bmod 16$	$hash_1(s_3) = 2 \bmod 16$
$hash_2(s_1) = 7 \bmod 16$	$hash_2(s_2) = 1 \bmod 16$	$hash_2(s_3) = 4 \bmod 16$
$hash_3(s_1) = 11 \bmod 16$	$hash_3(s_2) = 7 \bmod 16$	$hash_3(s_3) = 2 \bmod 16$
Bit Array $\Rightarrow \underbrace{00\,01\,00\,01\;00\,01\,00\,00}_{16}$	$\Rightarrow \underbrace{01\,01\,00\,01\;00\,11\,00\,00}_{16}$	$\Rightarrow \underbrace{01\,11\,10\,01\;00\,11\,00\,00}_{16}$

Έπειτα, ελέγχουμε την ύπαρξη των στοιχείων s_4, s_5, s_3, για τα οποία έχουμε:

$hash_1(s_4) = 5 \bmod 16$	$hash_1(s_5) = 2 \bmod 16$	$hash_1(s_3) = 2 \bmod 16$
$hash_2(s_4) = 6 \bmod 16$	$hash_2(s_5) = 10 \bmod 16$	$hash_2(s_3) = 4 \bmod 16$
$hash_3(s_4) = 1 \bmod 16$	$hash_3(s_5) = 11 \bmod 16$	$hash_3(s_3) = 2 \bmod 16$
Bit Array $\Rightarrow \underbrace{0\underline{1}\,11\,1\underline{0}\,\underline{0}1\;00\,11\,00\,00}_{16}$	$\Rightarrow \underbrace{01\,\underline{11}\,10\,01\;00\,\underline{11}\,00\,00}_{16}$	$\Rightarrow \underbrace{01\,\underline{11}\,\underline{1}0\,01\;00\,11\,00\,00}_{16}$
Answer : *no*	*yes*	*yes*

Στη περίπτωση του s_5 έχουμε ένα false positive αφού το στοιχείο φαίνεται πως ανήκει στο σύνολο, ενώ στην πραγματικότητα δεν ανήκει. Οι υπόλοιπες δύο απαντήσεις είναι έγκυρες.

Bloom Filter – Πολυνηματική (Multi-Threaded) Έκδοση. Στην άσκηση αυτή καλείστε να υλοποιήσετε ένα bloom filter που θα διαμοιράζεται ανάμεσα σε πολλά νήματα. Κάθε νήμα θα ακολουθεί δικό του μονοπάτι αναζήτησης, προσπαθώντας να καταλήξει στην κρυφή λέξη, ξεκινώντας από μία αρχική τυχαία συμβολοσειρά. Σε περίπτωση αποτυχίας (δηλαδή αν το νήμα έχει εξαντλήσει όλες τις λέξεις που παράγονται μεταβατικά από την αρχική που δοκίμασε), το νήμα ξεκινάει νέα αναζήτηση, χρησιμοποιώντας μία νέα αρχική τυχαία συμβολοσειρά, μέχρι κάποιο αριθμό επαναλήψεων. Ο μέγιστος αριθμός επαναλήψεων για κάθε νήμα καθορίζεται από την παράμετρο εκτέλεσης L. Αν η κρυφή λέξη δεν έχει βρεθεί μετά και το πέρας όλων των επαναλήψεων, τότε το νήμα, πριν τερματίσει τη λειτουργία του, γράφει σε ένα log-file το αναγνωριστικό του, το πλήθος λέξεων που δοκίμασε και το ποσοστό των λέξεων που δοκίμασε και βρίσκονταν ήδη στο bloom filter.

Σε περίπτωση επιτυχίας, το νήμα που βρήκε την κρυφή λέξη θα πρέπει να ειδοποιεί τα άλλα νήματα (π.χ. με μια δημόσια μεταβλητή την οποία ελέγχουν τα άλλα νήματα κάθε t κλήσεις της oracle που κάνουν - το $t > 20$ ώστε να μη γίνει bottleneck αυτή η μεταβλητή, αν κλειδώνει κανείς πριν τη διαβάσει). Κάθε νήμα, πριν τερματίσει τη λειτουργία του, γράφει σε ένα log-file το αναγνωριστικό του, το πλήθος λέξεων που δοκίμασε και το ποσοστό των λέξεων που δοκίμασε και βρίσκονταν ήδη στο bloom filter. Το όνομα του log-file θα δίνεται υποχρεωτικά σαν παράμετρος εκτέλεσης.

Η λειτουργικότητα του bloom filter παραμένει ακριβώς η ίδια όπως και στην περίπτωση της απλής αναζήτησης. Ωστόσο, το πρόγραμμά σας θα πρέπει να εξασφαλίζει ατομική πρόσβαση σε κάθε νήμα. Στην

προσέγγιση που θα υλοποιήσετε, το bloom filter χωρίζεται σε τμήματα και κάθε τμήμα προστατεύεται από ένα mutex. Ένα ενδεικτικό παράδειγμα ενός bloom filter μεγέθους 16 bit, που έχει χωριστεί σε 4 τμήματα (sections), φαίνεται παρακάτω:

$$\text{Bit Array} \Rightarrow \underbrace{0111}_{Section0}\ \underbrace{1001}_{Section1}\ \underbrace{0011}_{Section2}\ \underbrace{0000}_{Section3}$$

Στην πραγματική υλοποίησή σας, υποθέστε ότι ο αριθμός των τμημάτων είναι τέτοιος ώστε το μέγεθος του κάθε τμήματος να είναι πολλαπλάσιο των 64 bytes. Δηλαδή κάθε mutex προστατεύει τουλάχιστον $64 * 8 = 512$ bits του bloom filter.

Κάθε νήμα, πριν αλλάξει την τιμή οποιουδήποτε bit, θα πρέπει πρώτα να έχει κλειδώσει **όλα** τα τμήματα που θα χρειαστεί να προσπελάσει. Αυτό θα πρέπει να γίνει με τρόπο ώστε να αποφεύγονται περιπτώσεις αποκλεισμού (*deadlock*), μια και άλλα νήματα θα προσπαθούν ταυτόχρονα να κλειδώσουν τμήματα του bloom filter .

Είστε υπεύθυνοι για να ελέγξετε ότι (α) το πρόγραμμά σας δεν καταναλώνει όλο και περισσότερη μνήμη όσο τρέχει, (β) επιταχύνεται με τη χρήση πολλών νημάτων. Το (α) σημαίνει ότι η κατανάλωση μνήμης μετά από κάποιο σημείο δεν αυξάνει ή αυξάνει πολύ αργά. Για να εξασφαλίσετε αυτές τις δύο ιδιότητες πρέπει να κάνετε μετρήσεις χρονισμού (π.χ. με time) και να παρακολουθήσετε τη συμπεριφορά μνήμης του προγράμματός σας (π.χ. με top). Τις διαπιστώσεις σας από αυτές τις μετρήσεις θα τις αναφέρετε (με περίληψη των μετρήσεων) στην αναφορά που θα παραδώσετε μαζί με την άσκηση σας.

Εξυπηρετητής (Server) – Λειτουργία Αντιγραφής. Το πρόγραμμα σας θα περιμένει για εξωτερικές συνδέσεις σε μία προκαθορισμένη θύρα (port). Για κάθε νέα σύνδεση που δέχεται, ο server θα δημιουργεί ένα νέο νήμα (thread), το οποίο θα είναι υπεύθυνο αρχικά, να στείλει όλες τις παραμέτρους εκτέλεσης και έπειτα, να στείλει το ίδιο το bloom filter. Το συγκεκριμένο νήμα, πριν ξεκινήσει την αποστολή δεδομένων, θα πρέπει πρώτα να εξασφαλίσει αποκλειστική πρόσβαση στο bloom filter, κλειδώνοντας **όλα** τα τμήματα. Μόλις ολοκληρωθεί η αποστολή των δεδομένων, το εν λόγω νήμα ξεκλειδώνει το bloom filter και τερματίζει τη λειτουργία του.

Το πρόγραμμα που ξεκινάει τη διαδικασία αντιγραφής του bloom filter θα πρέπει να ελέγξει τις εισερχόμενες παραμέτρους εκτέλεσης. Εφόσον οι παράμετροι εκτέλεσης του τοπικού προγράμματος είναι **ακριβώς ίδιες** με τις εισερχόμενες, τότε και μόνο τότε ξεκινάει η διαδικασία αναζήτησης. Διαφορετικά, εκτυπώνεται ένα κατάλληλο μήνυμα λάθους και το πρόγραμμά σας τερματίζει τη λειτουργία του.

Εκτέλεση Προγράμματος Το πρόγραμμα θα καλείται ως εξής, από τη γραμμή εντολών:

```
./invoke-oracle-multithreaded SIZE N L PORT LOGFILE [-k NUM] [-h ADDRESS]
```

1. Η παράμετρος `SIZE` καθορίζει το μέγεθος του bloom filter σε *bytes*. (Χρησιμοποιείστε τη σταθερά `CHAR_BIT` του `limits.h` για να μετατρέψετε σε bits.) Ενδεικτικό μέγεθος του bloom filter, για τα δεδομένα της άσκησης, θα είναι της τάξης των 2^{20} bits.
2. Η παράμετρος `N` καθορίζει τον αριθμό των νημάτων που θα αναζητούν την κρυφή λέξη.
3. Η παράμετρος `L` καθορίζει τον αριθμό των επανεκκινήσεων που μπορεί να κάνει ένα νήμα μέχρι να βρει την κρυφή λέξη.
4. Η παράμετρος `PORT` καθορίζει τη θύρα στην οποία το πρόγραμμά σας περιμένει για εξωτερικές συνδέσεις.
5. Η παράμετρος `LOGFILE` καθορίζει το όνομα αρχείου στο οποίο γράφονται τα στατιστικά εκτέλεσης κάθε νήματος.
6. Η προαιρετική επιλογή '`-k NUM`' καθορίζει το πλήθος των hash functions που θα χρησιμοποιηθούν για το bloom filter . Θεωρείστε ότι η default τιμή είναι 3, αν παραλειφθεί.
7. Η προαιρετική επιλογή '`-h ADDRESS`' καθορίζει τη διεύθυνση από την οποία θα αντιγραφούν μέσω δικτύου τα περιεχόμενα (δηλαδή το bloom filter) άλλης ταυτόχρονης εκτέλεσης του προγράμματος, αφού ελεγθεί ότι οι παράμετροι εκτέλεσης είναι ίδιες.

Λεπτομέρειες Υλοποίησης

– Η αρχική κατάσταση παιχνιδιού (δηλαδή η κρυφή λέξη) αλλάζει με χρήση της ρουτίνας `void initSeed(int seed)`.

– Στην περίπτωση που το πρόγραμμά σας έχει κληθεί με την ειδική παράμετρο '`-h ADDRESS`', θα πρέπει να αρχικοποιήσετε το bloom filter πριν ξεκινήσει η διαδικασία αναζήτησης.

Hash Functions Εκτός της `oracle()` θα σας παρέχεται επίσης και ένα σύνολο από hash functions (για την υλοποίηση του bloom filter), οι οποίες θα είναι προσβάσιμες μέσω της ακόλουθης συνάρτησης:

```
uint64_t hash_by(int i, const char *word);
```

Η συνάρτηση αυτή, δεδομένη μίας συμβολοσειράς `word` και ενός δείκτη `i`, θα καλεί την i-οστή hash function και θα σας επιστρέφει το αποτέλεσμα (τύπου `uint64_t`, δηλαδή 64-bit μη-προσημασμένου ακεραίου).

Διαδικαστικά:
– Το πρόγραμμά σας θα πρέπει να τρέχει στα μηχανήματα Linux/Unix της σχολής.
– Για επιπρόσθετες ανακοινώσεις, παρακολουθείτε το forum του μαθήματος στο piazza.com για να δείτε ερωτήσεις/απαντήσεις/διευκρινήσεις που δίνονται σχετικά με την άσκηση. Η παρακολούθηση του φόρουμ στο piazza.com είναι *υποχρεωτική*.

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 τροποποιήσετε τα δικαιώματα αυτού του αρχείου πριν την υποβολή, π.χ. με την εντολή `chmod 755 OnomaEponymoProject3.tar` (περισσότερα στη σελίδα του μαθήματος).
2. Ένα README (απλό text αρχείο) με κάποια παραδείγματα μεταγλώττισης και εκτέλεσης του προγράμματός σας. Επίσης, μπορείτε να συμπεριλάβετε άλλες διευκρινίσεις που κρίνετε απαραίτητες (για τυχόν παραδοχές που έχετε κάνει, κτλ).
3. Οποιαδήποτε πηγή πληροφορίας, συμπεριλαμβανομένου και κώδικα που μπορεί να βρήκατε στο Διαδίκτυο θα πρέπει να αναφερθεί και στον πηγαίο κώδικά σας αλλά και στο παραπάνω README.

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).

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).
5. Η υλοποίηση της `oracle()` ενδέχεται να αλλάξει. Το πρόγραμμά σας θα πρέπει να λειτουργεί σωστά για οποιαδήποτε έκδοσή της.