HY486 - Αρχές Κατανεμημένου Υπολογισμού
Εαρινό Εξάμηνο 2015-2016

Πρώτη Προγραμματιστική Εργασία

1. Γενική περιγραφή

Η πρώτη προγραμματιστική εργασία απαρτίζεται από δύο μέρη. Στο πρώτο μέρος καλείστε να υλοποιήσετε μια βιβλιοθήκη από διαμοιραζόμενες δομές δεδομένων (concurrent data structures) και να μετρήσετε την απόδοση των υλοποιήσεών σας με benchmarks τα οποία θα σχεδιάσετε και θα υλοποιήσετε εσείς οι ίδιοι. Το μέρος αυτό περιγράφεται αναλυτικά στην Ενότητα 2. Στο δεύτερο μέρος θα χρησιμοποήσετε κάποιες από αυτές τις δομές για να εκτελέσετε ένα πολυνηματικό υπολογισμό (multithreaded computation), ως παράδειγμα χρήσης των δομών αυτών στην πράξη. Το δεύτερο μέρος περιγράφεται αναλυτικά στην Ενότητα 3.

Η προγραμματιστική εργασία θα πρέπει να υλοποιηθεί στη γλώσσα C με τη χρήση της βιβλιοθήκης pthreads.

Είναι αξιοσημείωτο ότι κάποια μέρη της προγραμματιστικής εργασίας είναι υποχρεωτικά για τους προπτυχιακούς φοιτητές αλλά όχι για τους μεταπτυχιακούς φοιτητές, ενώ για άλλα μέρη της εργασίας ισχύει το αντίθετο, όπως υποδεικνύεται στην εκφώνηση.

2. Υλοποίηση και πειραματική ανάλυση μιας βιβλιοθήκης διαμοιραζόμενων δομών δεδομένων

Στο πρώτο μέρος της προγραμματιστικής εργασίας ζητείται να υλοποιήσετε μια βιβλιοθήκη που θα περιέχει τις παρακάτω δομές δεδομένων:

- Διαμοιραζόμενος Μετρητής (concurrent counter)
- Διαμοιραζόμενες Ταξινομημένες Συνδεδεμένες Λίστες (concurrent linked list) (1) με χρήση ενός coarse-grained κλειδώματος, (2) hand-over-hand κλειδωμάτων (fine-grained locking) και (3) με χρήση lazy synchronization (όπως διδάχθηκε στο μάθημα με κλειδώματα).
- Διαμοιραζόμενη Ουρά (concurrent queue) με χρήση δύο κλειδωμάτων

Οι ορισμοί των συναρτήσεων (function definitions) βρίσκονται σε ξεχωριστά header files, ένα για κάθε διαμοιραζόμενη δομή. Ο κώδικας για την κάθε δομή θα πρέπει να βρίσκεται στο αντίστοιχο αρχείο .c, του οποίου το όνομα πρέπει να είναι ίδιο με αυτό του header αρχείου. Για παράδειγμα, για το διαμοιραζόμενο μετρητή, οι ορισμοί των συναρτήσεων βρίσκονται στο αρχείο concurrent_counter.h και η υλοποίηση των συναρτήσεων αυτών θα πρέπει να βρίσκεται αντίστοιχα στο concurrent_counter.c.

Εφόσον οι δομές αυτές θα προσπελαζόνται από πολλά νήματα ταυτόχρονα, θα πρέπει να προστατεύονται κατάλληλα από κλειδώματα (locks) όπου αυτό χρειάζεται. Για τα κλειδώματα αυτά μπορείτε να χρησιμοποιήσετε τα mutexes που παρέχονται από την βιβλιοθήκη pthreads (βλέπε υλικό φροντιστηρίου pthread_mutex_t).

Ακολουθεί η αναλυτική περιγραφή και το API κάθε διαμοιραζομένης δομής δεδομένων.

Concurrent counter	
Stores an integer value protected by a mutex	
API	
void init_counter(counter_t *counter)	Sets val to 0 and initializes the mutex.
void increment_counter(counter_t *counter)	Increments val by one.
void decrement_counter(counter_t *counter)	Decrements val by one
int get_counter(counter_t *counter)	Returns the current value of val
int test_and_set_counter(counter_t *counter, int test_value, int new_value)	Tests the current value of val against test_value and if equal sets it to new_value.Must return 0 upon failure and 1 upon success.

Concurrent linked lists: (1) simple, (2) hand-over-hand locking, and (3) lazy synchronization	
Three different implementations (in order to measure performance differences)	
API	
void list_init(list_t *list)	Initializes the list
int list_insert(list_t *list, int key)	Inserts a node at the head of the list with data set to key
int list_delete(list_t *list, int key)	Deletes node with matching key from the list.

	Returns FALSE upon failure or TRUE upon success
node_t *list_lookup(list_t *list, int key)	Traverses the list to find the node with a matching key. Returns a pointer to the node if found, or NULL otherwise.

Concurrent queues (Two-locks implementation of Michael & Scott)	
A concurrent queue that utilizes a lock for the tail and a separate lock for the head of the queue that allows concurrent enqueue/dequeue operations.	
API	
void queue_init(queue_t *queue, int max_elements);	Initializes the queue (head/tail pointers and associated locks) and sets the maximum number of elements.
int queue_enqueue(queue_t *queue, int value);	Inserts an element in the queue. Returns -1 (overflow) if the maximum number of elements has been reached, or 0 on success.
int queue_dequeue(queue_t *queue, int *value);	Dequeues the element at the head of the queue and stores it into *value. Returns FALSE (underflow) if the queue is empty or TRUE on success.
int queue_peek(queue_t *queue, int *value);	Returns the value at the head of the queue without removing it (it stores it into *value). Returns FALSE if queue is empty, or or TRUE upon success.
int queue_isfull(queue_t *queue);	Returns TRUE if queue is full or FALSE if not.
int queue_isempty(queue_t *queue);	Returns TRUE if queue is empty or FALSE if not.

Τέλος, για το μέρος αυτό θα χρειαστεί να μετρήσετε την απόδοση των διαφόρων υλοποιήσεων της διαμοιραζόμενης λίστας που υλοποιήσατε. Για να γίνει αυτό, θα πρέπει να υλοποιήσετε τα κατάλληλα benchmarks, ένα για κάθε υλοποίηση της λίστας που πρέπει να μετρήσετε. Για κάθε μια από τις

υλοποιήσεις αυτές, θα πρέπει να εκτελούνται 500000 λειτουργίες σε έναν βρόγχο και να σημειώνεται ο χρόνος εκτέλεσης με τη χρήση της συνάρτησης gettimeofday, την οποία θα πρέπει να καλείται με την έναρξη και τη λήξη της εκτέλεσης του βρόγχου. Θα πρέπει να παρουσιάσετε δύο πειράματα για κάθε δομή:

1. Από τις 500000 λειτουργίες, 10% είναι εισαγωγές, 10% είναι διαγραφές και 80% είναι αναζητήσεις. Αυτό θα πραγματοποιηθεί με τον εξής αλγόριθμο: Αρχικά γίνονται 50000 εισαγωγές στη δομή που εισάγουν τα κλειδιά 1 έως 50000. Στη συνέχεια ακολουθούν αναζητήσεις ως εξής: το νήμα j ξεκινάει αναζητώντας το κλειδί 10j και συνεχίζει αναζητώντας το κλειδί που έχει τιμή (όσο το προηγούμενο κλειδί που αναζήτησε + 17) % 50000. Ακολουθεί η φάση διαγραφής όπου όλα τα κλειδιά διαγράφονται. Το νήμα j διαγράφει τα κλειδιά j, 2j, 3j, κ.ο.κ., μέχρι να διαγραφούν και τα 50000 κλειδιά της λίστας.
2. Από τις 500000 λειτουργίες, 40% είναι εισαγωγές, 40% είναι διαγραφές και 20% είναι αναζητήσεις.

Το πρόγραμμά σας θα πρέπει να μετράει το συνολικό χρόνο εκτέλεσης των λειτουγιών εισαγωγής, το συνολικό χρόνο εκτελέσης των λειτουργιών αναζήτησης και τον συνολικό χρόνο εκτέλεσης των λειτουργιών διαγραφής. Θα πρέπει επίσης να εμφανίζει το χρόνο που απαιτήθηκε κατά μέσο όρο για την εκτέλεση κάθε είδους λειτουργίας.

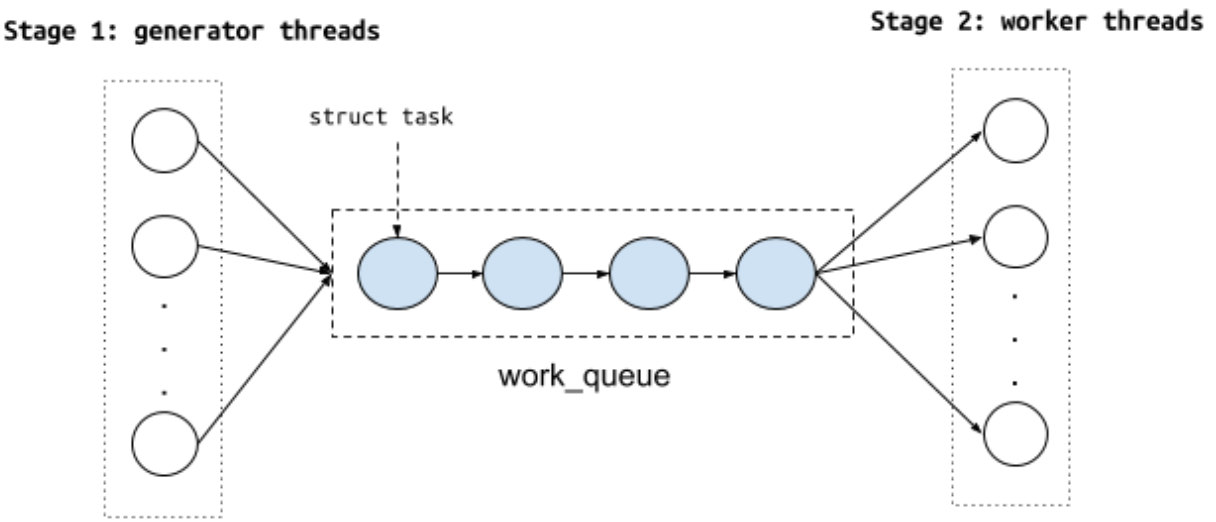
Πιο συγκεκριμένα, θα πρέπει να προσμετράται ο μέσος χρόνος εκτέλεσης ανά λειτουργία και ο συνολικός χρόνο εκτέλεσης. Τα αποτελέσματα σας θα πρέπει να παρουσιαστούν σε γραφήματα. Στα γραφήματα αυτά, ο y άξονας θα πρέπει να αναπαριστά το χρόνο (πιθανότατα σε milliseconds) και ο x άξονας θα αναπαριστά το πλήθος των νημάτων που εκτελούν το πείραμα και θα πρέπει να πειραματιστείτε με 1, 4, 8, 16 νήματα.

Τα γραφήματα αυτά, καθώς και περιγραφή που θα δικαιολογεί τα αποτελέσματα των πειραμάτων σας θα πρέπει να παραδοθούν σε ένα pdf αρχείο μαζί με την άσκηση.

3. Υλοποίηση multithreaded md5sum calculator

Σε αυτή την ενότητα απαιτείται να υλοποιήσετε ένα πρόγραμμα στο οποίο εκτελούνται πολλά νήματα, με τη χρήση της βιβλιοθήκης pthreads, το οποίο θα χρησιμοποιεί τις δομές που υλοποιήσατε στην Ενότητα 2 και θα υπολογίζει τα MD5 checksums των αρχείων ενός directory. Τα αρχεία αυτά περιέχουν randomly generated data.

Για την υλοποίηση του παραπάνω προγράμματος θα χρειαστεί να υλοποιήσετε ένα πρόγραμμα στο οποίο δύο οικογένειες νημάτων εκτελούνται ταυτόχρονα: νήματα τύπου **generator** και νήματα τύπου **worker**. Επίσης θα πρέπει να χρησιμοποιηθούν οι εξής δομές δεδομένων που έχετε υλοποιήσει στο πρώτο μέρος: διαμοιραζόμενος μετρητής (concurrent counter) και μια διαμοιραζόμενη ουρά (concurrent queue) που ονομάζεται work_queue. Τα νήματα και οι δομές παρουσιάζονται στο Σχήμα 1.



struct task:

Περιγράφει έναν κόμβο της ουράς work_queue και έχει τα εξής πεδία,

```
struct task {
        char file_path[50];
        struct task *next;
};
```

Νήματα τύπου Generator:

Ο αριθμός των νημάτων τύπου generator θα πρέπει να μπορεί να παραμετροποιείται κατά τη μεταγλώτισση (at compile time) ως ακολούθως:

```
#define GENERATOR_THREAD_NUM 16
```

Τα νήματα τύπου generator επιλέγουν ένα αρχείο από το directory με την χρήση της συνάρτησης readdir (την οποία χρησιμοποιήσαμε και στο φροντιστήριο για τα προγράμματα md5_file και md5_file_multithreaded) και δημιουργούν ένα task, τύπου **struct task, το οποίο εισάγουν στην ουρά**. Ένα απο τα πεδία αυτού του struct περιέχει το πλήρες path προς το αρχείο σε μορφή string (path_to_file field). Για παράδειγμα, αν το filename που μας επέστρεψε η readdir είναι το f1.dat και το directory είναι το test_files, τότε το path προς το αρχείο είναι το test_directory/f1.dat. Σημειώστε πως η εκτέλεση της readdir() από πολλά νήματα ταυτόχρονα απαιτεί συγχρονισμό.

Εάν η readdir επιστρέψει NULL, το οποίο σημαίνει πως δεν υπάρχουν άλλα αρχεία να επεξεργαστούμε, τα νήματα τύπου generator πρέπει να τερματίσουν την εκτέλεσή τους με τη χρήση της pthread_exit, αφού πρώτα θέσουν την τιμή μιας διαμοιραζόμενης μεταβλητής, **work_done** , σε TRUE για να σηματοδοτήσουν πως δεν θα γίνουν άλλες προσθήκες στην ουρά (η μεταβλητή αυτή είναι απαραίτητη για τον ομαλό τερματισμό των νημάτων τύπου worker).

work_queue:
Τα στοιχεία της ουράς αυτής είναι τύπου struct task και στηρίζεται στην υλοποίηση της διαμοιραζόμενης ουράς που έχετε κάνει στην Ενότητα 2. Θα πρέπει να προσαρμόσετε την προηγούμενη υλοποίηση σας σε ξεχωριστά αρχεία (work_queue.h, work_queue.c) ώστε η ουρά αυτή να δουλεύει με τον σωστό τύπο δεδομένων (αντί απλά για int όπως στην αρχική υλοποίηση την οποία πρέπει επίσης να παραδώσετε).

Νήματα τύπου Worker:

Κάθε νήμα τύπου worker εξάγει ένα struct task απο την work_queue και υπολογίζει το MD5 checksum του αρχείου που περιγράφει το πεδίο file_path του struct αυτού. Εάν το νήμα βρει την ουρά άδεια, τότε θα πρέπει να ελέγξει τη μεταβλητή work_done και να τερματίσει αν η work_done είναι ίση με TRUE. διαφορετικά, πρέπει να ελέγχει περιοδικά και αν υπάρχουν tasks στην ουρά και αν η work_done έχει γίνει TRUE και να τερματίσει όταν έχει βεβαιωθεί πως δεν υπάρχουν άλλα tasks προς εξυπηρέτηση.

Το πρόγραμμα σας πρέπει να συνοδεύεται και από το κατάλληλο makefile με εντολές που κάνουν compile και clean τα binaries.

Για τον κώδικα της readdir( ), τα scripts τα οποία δημιουργούν τα αρχεία καθώς και τον κώδικα υπολογισμού των MD5 checksums μπορείτε να χρησιμοποιήσετε το υλικό που σας έχει σταλεί από το πρώτο φροντιστήριο. Συγκεκριμένα, δείτε το αρχεία md5_file_multithreaded.c και setup.sh.

Εκτέλεση των εργασιών (tasks) με την τεχνική της κλοπής εργασίας (work stealing) - Υποχρεωτικό για τους μεταπτυχιακούς φοιτητές - 10% bonus για τους προπτυχιακούς φοιτητές

Αυτό το κομμάτι της άσκησης είναι υποχρεωτικό μόνο για τους μεταπτυχιακούς φοιτητές και είναι bonus για τους προπτυχιακούς φοιτητές. Αφορά την Ενότητα 3, όπου θα πρέπει να αποθηκεύεται με διαφορετικό τρόπο τα tasks καθώς και να τροποποιήσετε τη συμπεριφορά των worker_threads.

Η τεχνική της κλοπής εργασίας χρησιμοποιείται κατά τη δρομόλογηση (scheduling) πολυνηματικών υπολογισμών:
- Για κάθε νήμα υπάρχει μια ουρά όπου αποθηκεύονται τα tasks που το νήμα πρέπει να εκτελέσει. Επομένως, τώρα υπάρχουν τόσες working_queues όσες και τα νήματα.
- Τα νήματα τύπου generator επιλέγουν τυχαία (με πιθανότητες που είναι biased) μια ουρά για να εισάγουν κάθε task που παράγουν.
- Κάθε νήμα τύπου worker εξάγει ένα task απο την ουρά του (όσο αυτή δεν είναι άδεια) και το εκτελεί.
- Αν η ουρά ενός νήματος είναι άδεια, το νήμα επιχειρεί να "κλέψει" tasks απο τις ουρές άλλων νημάτων. Επιλέγει τυχαία μια τέτοια ουρά και εκτελεί μια dequeue() για να «κλέψει» ένα task και να το εκτελέσει. Όταν ένα worker thread «κλέψει» από την ουρά ενός άλλου thread, πρέπει να αυξάνει

έναν διαμοιραζόμενο μετρητή ο οποίος θα μετράει πόσα «κλεψίματα» έγιναν κατά την εκτέλεση του προγράμματος και θα τυπώνεται κατά την έξοδο από το πρόγραμμα.

Για να δημιουργηθούν οι κατάλληλες συνθήκες για κλοπή εργασίας θα πρέπει να υπάρχει μια ασυμμετρία μεταξύ των tasks τα οποία εισάγονται στις διάφορες ουρές, η οποία μπορεί να υλοποιηθεί είτε με διαφορετικούς ρυθμούς τοποθέτησης tasks στις διάφορες ουρές, είτε με διαφορετικά μεγέθη αρχείων ώστε η εκτέλεση διαφορετικών tasks να έχει διαφορετικές απαιτήσεις σε χρόνο εκτέλεσης. Σημειώνεται ότι αυτή η ασυμμετρία μπορεί να προκύψει αν τα νήμα τύπου generator προσθέτουν tasks σε ουρές εργασίας με τυχαίο τρόπο χωρίς να ακολουθείται η ομοιόμορφη κατανομή στην επιλογή των ουρών.

Θα πρέπει να επιλέξετε τις κατάλληλες παραμέτρους κατά την εκτέλεση του πρόγραμματός σας ώστε να παρατηρούνται διάφορα επίπεδα κλοπής εργασίας (έντονο, μη-έντονο, κλπ). Συνοπτικά, με βάση τα παραπάνω θα πρέπει να πειραματιστείτε με τις εξής παραμέτρους:

- δίκαιης έναντι μη-δίκαιης δρομολόγησης των tasks στις ουρές
- ομοιόμορφα έναντι μη-ομοιόμορφων μεγεθών αρχείων

Καταγράψτε τα ευρήματα σας για τους διάφορους συνδυασμούς των παραπάνω μεταβλητών και εξηγήστε γιατί καταλήξατε στο συνδυασμό που φαίνεται να παράγει τις πιο κατάλληλες συνθήκες, στο pdf αρχείο που θα παρδώσετε.

4. Παράδοση άσκησης

Η παράδοση των προγραμματιστικών ασκήσεων γίνεται μέσω του προγράμματος turnin. Το όνομα του παραδοτέου για την άσκηση είναι **project1**. Η εντολή turnin που πρέπει να χρησιμοποιηθεί είναι η εξής:

turnin project1@hy486 <dir>

όπου <dir> είναι ο φάκελος όπου βρίσκονται τα αρχεία που πρέπει να παραδοθούν. Επειδή το πρόγραμμα turnin δεν καταχωρεί συμπιεσμένα αρχεία, τα αρχεία προς παράδοση δεν πρέπει να είναι binary/compressed/gzipped. Θα πρέπει να παραδώσετε τα απαραίτητα αρχεία απο την Ενότητα 1 (τα αρχεία .c και .h των διαμοιραζόμενων δομών, τα benchmarks καθώς και το .pdf αρχείο με τα γραφήματα και τις εξηγήσεις σας) όπως και τα αρχεία απο την Ενότητα 2 (το πολυνηματικό πρόγραμμα για τον υπολογισμό των MD5 checksums και τις τροποποιημένες διαμοιραζόμενες δομές όπου αυτές απαιτούνται).