Τμήμα Μηχ. Η/Υ & Πληροφορικής **Εαρινό εξάμηνο 2024**
Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων **Διδάσκων: Σ. Αναστασιάδης**

ΜΥΥ601 Λειτουργικά Συστήματα

Ανακοίνωση: Τετάρτη 13 Μαρτίου, Παράδοση: Παρασκευή, 5 Απριλίου στις 21:00

Εργαστήριο 1:Υλοποίηση πολυνηματικής λειτουργίας σε μηχανή αποθήκευσης δεδομένων

1. Εισαγωγή

Σας δίνεται μια μηχανή αποθήκευσης (storage engine) υλοποιημένη στη γλώσσα C. Σας ζητείται να υλοποιήσετε την πολυνηματική λειτουργία των εντολών add και get που παρέχει η μηχανή αποθήκευσης. Η υλοποίησή σας θα επιτρέπει πολλαπλά νήματα να καλούν τις εντολές add και get ταυτόχρονα. Η μηχανή αποθήκευσης θα πρέπει να εκτελεί τις ταυτόχρονες λειτουργίες σωστά και να διατηρεί στατιστικά του χρόνου εκτέλεσης της κάθε λειτουργίας. Η υλοποίησή σας θα πρέπει να γίνει στην γλώσσα C και να βασίζεται στην βιβλιοθήκη Pthreads του Linux.

1.1 Μηχανή αποθήκευσης βασισμένη στο δέντρο LSM

Ο πηγαίος κώδικας που σας δίνεται υλοποιεί την μηχανή αποθήκευσης **Kiwi** που βασίζεται σε δέντρο log-structured merge (LSM-tree). Οι μηχανές αποθήκευσης είναι σημαντικό μέρος των σύγχρονων υποδομών νέφους καθώς είναι υπεύθυνες για την αποθήκευση και ανάκτηση δεδομένων στις τοπικές συσκευές μιας μηχανής. Ένα κατανεμημένο σύστημα αποθήκευσης χρησιμοποιεί χιλιάδες τέτοιες μηχανές για να πετύχει κλιμακώσιμη και αξιόπιστη λειτουργία (π.χ., Facebook, Google).

Το LSM-tree είναι η δομή δεδομένων στην οποία συχνά βασίζονται οι μηχανές αποθήκευσης. Η παρεχόμενη προγραμματιστική διεπαφή (API) περιλαμβάνει λειτουργίες add και get για ζεύγη κλειδιού-τιμής. Η λειτουργία add δέχεται ως παράμετρο ένα ζεύγος κλειδιού-τιμής που πρέπει να προστεθεί στην δομή. Η λειτουργία get δέχεται ως παράμετρο ένα κλειδί και αιτείται την αντίστοιχη τιμή εφόσον υπάρχει αποθηκευμένο στην δομή ζεύγος κλειδιού-τιμής με το συγκεκριμένο κλειδί, αλλιώς επιστρέφει σφάλμα αν το κλειδί δεν βρεθεί.

Για την γρήγορη αναζήτηση, η δομή διατηρεί τα αποθηκευμένα δεδομένα ταξινομημένα στην μνήμη ή τον δίσκο. Συνήθως, τα πιο πρόσφατα δεδομένα διατηρούνται στην μνήμη σε μια ταξινομημένη δομή που λέγεται memtable και συνήθως υλοποιείται με skip list. Η skip list είναι μια δομή δεδομένων που αποτελείται από πολλαπλές λίστες οργανωμένες σε διαφορετικά επίπεδα για να επιταχύνουν την αναζήτηση κάποιου στοιχείου. Όταν το memtable γεμίσει, μετακινείται στον δίσκο προκειμένου να αδειάσει η μνήμη και να μπορέσει να δεχτεί καινούρια δεδομένα. Κατά το διάστημα που το memtable παραμένει στην μνήμη, υπάρχει ένα αρχείο log που λαμβάνει τα εισερχόμενα δεδομένα και τα αποθηκεύει προσωρινά στον δίσκο για την γρήγορη ανάκτησή τους σε περίπτωση σφάλματος.

Τα αρχεία που αποθηκεύονται στο δίσκο είναι οργανωμένα σε πολλαπλά επίπεδα των οποίων το μέγεθος αυξάνει με γεωμετρική πρόοδο από τα χαμηλότερα προς τα υψηλότερα επίπεδα (0..6). Ένα αρχείο δίσκου στην δομή αυτή ονομάζεται Sorted String Table (sst) επειδή τα δεδομένα είναι ταξινομημένα με βάση τα κλειδιά που περιέχουν συμβολοσειρές. Όταν το memtable μετακινείται στον δίσκο, *συγχωνεύεται (merging)* με τα αρχεία των οποίων τα κλειδιά επικαλύπτονται με τα κλειδιά του memtable. Επιπρόσθετα, ενεργοποιείται *σύμπτυξη (compaction)* αρχείων όταν το πλήθος των αρχείων στο επίπεδο 0 ξεπερνά ένα προκαθορισμένο κατώφλι (π.χ., 4), ή το συνολικό μέγεθος των αρχείων σε ένα επίπεδο ξεπερνάει ένα προκαθορισμένο κατώφλι για το επίπεδο αυτό. Με την σύμπτυξη, τα αρχεία ενός επίπέδου συγχωνεύονται σε ένα αρχείο που προστίθεται στο επόμενο επίπεδο.

1.2 Πολυνηματική υλοποίηση

Στην υλοποίηση που σας δίνεται, υπάρχουν ήδη δύο (2) νήματα, ένα στην εφαρμογή και ένα στην μηχανή αποθήκευσης. Το πρώτο νήμα καλεί μια ακολουθία λειτουργιών *add* ή *get* την μία μετά

την άλλη όπως ορίζεται από την γραμμή εντολών, ενώ το δεύτερο νήμα είναι υπεύθυνο για την σύμπτυξη αρχείων όταν ικανοποιούνται οι απαιτούμενες συνθήκες που αναφέρθηκαν παραπάνω.

Η λειτουργία add αλληλεπιδρά με το memtable προκειμένου να εισαγάγει ένα νέο στοιχείο στην δομή. Αν απαιτείται, το τρέχον memtable συγχωνεύεται με ένα αρχείο sst, που μπορεί να οδηγήσει σε περαιτέρω σύμπτυξη από το ένα επίπεδο στο επόμενο.

Η λειτουργία get κάνει αναζήτηση για το παρεχόμενο κλειδί πρώτα στο memtable και αν δεν βρεθεί εκεί στα αρχεία sst ξεκινώντας από το επίπεδο 0 και πηγαίνοντας σε επόμενα επίπεδα όπως απαιτείται. Είναι δυνατόν να έχει δρομολογηθεί η συγχώνευση ενός memtable, οπότε θα πρέπει και αυτό να υποστεί αναζήτηση πριν τα αρχεία sst. Τα αρχεία στα οποία θα γίνει αναζήτηση επιλέγονται με βάση το εύρος των κλειδιών που περιέχουν, υποθέτοντας ότι είναι περιττό να γίνει αναζήτηση σε ένα sst του οποίου το εύρος κλειδιών δεν περιέχει το κλειδί που αναζητείται. Η αναζήτηση σε ένα αρχείο sst γίνεται ταχύτερη με ένα Bloom filter που περιέχει τους κατακερματισμούς των κλειδιών που έχουν ήδη εισαχθεί στο συγκεκριμένο αρχείο.

Δεδομένης της πολυπλοκότητας του κώδικα, σας ζητείται να σχεδιάσετε την υλοποίηση σε βήματα. Μια τετριμμένη υλοποίηση θα χρησιμοποιούσε μία κλειδαριά για να εφαρμόσει αμοιβαίο αποκλεισμό στις λειτουργίες add και get. Μια βελτίωση θα επέτρεπε πολλαπλούς αναγνώστες ή ένα γραφέα να λειτουργεί κάθε φορά. Επιπλέον βήματα θα χρησιμοποιούσαν διαφορετικές κλειδαριές για το memtable και τα αρχεία sst, έτσι ώστε διαφορετικά νήματα να λειτουργούν ταυτόχρονα εφόσον δεν τροποποιούν το ίδιο μέρος του συστήματος το ίδιο χρονικό διάστημα. Η λύση θα πρέπει να λαμβάνει υπόψη ότι υπάρχει ήδη ένα νήμα που είναι υπεύθυνο για τις συγχωνεύσεις που τροποποιούν ασύγχρονα στο παρασκήνιο τα αρχεία sst.

Είστε υπεύθυνοι να τεκμηριώσετε τις αλλαγές που κάνετε στον κώδικα τόσο με σχόλια στον πηγαίο κώδικα που παραδίδετε, όσο και με λεπτομερή περιγραφή στην αναφορά που παραδίδετε. Κώδικας που δεν βγάζει νόημα και δεν τεκμηριώνεται επαρκώς δεν θα ληφθεί υπόψη στην βαθμολόγηση, ή μπορεί να την επηρεάσει αρνητικά.

1.3 Στατιστικά απόδοσης

Η υλοποίηση που σας δίνεται περιλαμβάνει κάποιες πολύ βασικές μετρήσεις της απόδοσης των λειτουργιών read (get) και write (add). Σας ζητείται να εξασφαλίσετε ότι λαμβάνονται σωστά οι μετρήσεις της απόδοσης κατά την πολυνηματική υλοποίηση που θα παραδώσετε. Σωστή πολυνηματική λειτουργία σημαίνει ότι οι μεταβλητές που καταγράφουν την απόδοση ενημερώνονται ατομικά από τα διαφορετικά νήματα χρησιμοποιώντας κλειδαριές για αμοιβαίο αποκλεισμό.

Η αναφορά σας θα πρέπει να περιλαμβάνει δοκιμές απόδοσης σε διαφορετικά σενάρια που κρίνετε σκόπιμα. Παραδείγματα είναι φορτία εργασίας που περιέχουν μόνο add ή μόνο get, καθώς και μίγμα add και get σύμφωνα με συγκεκριμένα ποσοστά για κάθε λειτουργία. Μπορείτε να παρουσιάσετε τα αποτελέσματα είτε απλά με screenshot ή καλύτερα με γραφικές παραστάσεις (π.χ., από το excel).

2. Προετοιμασία

Κατεβάστε την εικονική μηχανή MYY601Lab1(Kiwi).zip (1.5GB) και αποσυμπιέστε την στο τοπικό σκληρό δίσκο σας. Η εικονική μηχανή τρέχει την έκδοση 12 (“bookworm”) του Debian Linux 64-bit. Κατεβάστε και εγκαταστήστε τον VMware Player v17 στο μηχάνημά σας. Η μηχανή είναι ρυθμισμένη να χρησιμοποιεί 2 επεξεργαστικούς πυρήνες και 2GB RAM, αλλά αυτό αλλάζει εύκολα από τις ρυθμίσεις της εικονικής μηχανής. Για να εκμεταλλευτείτε το παραθυρικό περιβάλλον, εξασφαλίστε ότι τρέχετε την εικονική μηχανή σε κατάσταση πλήρους οθόνης πατώντας το κατάλληλο εικονίδιο πάνω αριστερά στον VMware Player. Μπορείτε να κάνετε login ως απλός χρήστης με το όνομα myy601 και το ίδιο ως κωδικό. Αν χρειάζεται να εγκαταστήσετε επιπλέον πακέτα στο σύστημα Linux της εικονικής μηχανής, μπορείτε να κάνετε login ως root (με κωδικό myy601) και να τρέξετε **apt install <package>**. Η εικονική μηχανή έχει εγκατεστημένη την γλώσσα προγραμματισμού C, τον εκσφαλματωτή gdb και τον φυλλομετρητή Firefox. Για διευκόλυνσή σας, μπορείτε να κάνετε copy-paste κείμενο και αρχεία μεταξύ του

λειτουργικού συστήματος της εικονικής μηχανής και του λειτουργικού συστήματος του μηχανήματός σας, π.χ., για να μεταφέρετε τον κώδικα που έχετε γράψει κατά την υποβολή με turnin.

Φρεσκάρετε τις γνώσεις σας στη γλώσσα C και εξασφαλίστε ότι κατανοείτε τις κλήσεις συναρτήσεων της βιβλιοθήκης Pthreads για την δημιουργία και τον συγχρονισμό των νημάτων. Επιπλέον, εξοικειωθείτε με τον εκσφαλματωτή **gdb** προκειμένου να κάνετε βηματική εκτέλεση και να δείτε τα τρέχοντα περιεχόμενα διαφόρων μεταβλητών (Cheatsheet & Reference). Εντολές του gdb που θα χρειαστείτε περιλαμβάνουν **file**, **list**, **start**, **run**, **break**, **info break**, **delete**, **thread**, **info thread**, **step**, **next**, **cont**, **bt**, **quit**.

Θα χρειαστείτε να χρησιμοποιήσετε βασικά εργαλεία της εικονικής μηχανής για να ανοίξετε τα αρχεία του πηγαίου κώδικα (π.χ., vim, emacs, nedit). Ανοίξτε ένα τερματικό και μετακινηθείτε στον κατάλογο **~/kiwi/kiwi-source** που περιέχει τον πηγαίο κώδικα της μηχανής αποθήκευσης και ένα απλό benchmark. Στον κατάλογο **kiwi-source** δημιουργήστε τα εκτελέσιμα εκτελώντας **make all** .

Μετά μετακινηθείτε στον κατάλογο **bench** που περιέχει το benchmark **kiwi-bench**. Μπορείτε να εισάγετε ένα πλήθος (π.χ., 10.000) στοιχείων στην μηχανή με την εντολή **./kiwi-bench write 10000** και στην συνέχεια να τα διαβάσετε με την εντολή **./kiwi-bench read 10000**. Εάν διαπιστώσετε πρόβλημα στα αποθηκευμένα δεδομένα, μπορείτε να μετακινηθείτε στον κατάλογο **bench** και να σβήσετε τον κατάλογο **testdb** με **rm -rf testdb**.

3. Εργασία

Η εργασία σας ζητά να υλοποιήσετε τις λειτουργίες add και get με δυνατότητα εκτέλεσης σε διαφορετικά νήματα.

Πρώτα χρειάζεστε μια βασική κατανόηση του μονοπατιού που ακολουθούν οι λειτουργίες add (**db_add**) και get (**db_get**). Δεν χρειάζεται να καταλάβετε όλες τις λεπτομέρειες του κώδικα αλλά να αποκτήσετε μια βασική ιδέα για τον τρόπο με τον οποίο μια λειτουργία εξυπηρετείται είτε από το memtable ή από τα αρχεία sst. Είναι χρήσιμο να πλοηγηθείτε στον κώδικα με την χρήση την τεκμηρίωσης που διατίθεται online στον επόμενο σύνδεσμο (Τεκμηρίωση κώδικα).

Στην επιλογή Data Structures της τεκμηρίωσης κώδικα θα βρείτε τον ορισμό σημαντικών δομών, όπως οι **_db**, **_memtable**, **_skiplist**, **_sst**. Ειδικότερα στην δομή **_sst** υπάρχουν ήδη μεταβλητές συγχρονισμού τύπου **pthread_mutex** ή **pthread_cond** που χρειάζεται να έχετε υπόψη σας όταν βελτιώνετε τον ταυτοχρονισμό του συστήματος.

Στην επιλογή Files μπορείτε να βρείτε σημαντικά αρχεία τύπου κεφαλίδας που ορίζουν την διεπαφή σημαντικών ενοτήτων του κώδικα. Ειδικότερα, θα πρέπει να εξοικειωθείτε με το αρχείο **db.h** και τις συναρτήσεις που θα τροποποιήσετε (π.χ., **db_add**, **db_get**), και τις αντίστοιχες συναρτήσεις στο **memtable.h**, **skiplist.h**, και **sst.h**.

Το αρχείο **config.h** περιέχει σημαντικούς ορισμούς των macros που καθορίζουν την συμπεριφορά του συστήματος, όπως την **BACKGROUND_MERGE** που ενεργοποιεί ξεχωριστό νήμα για τις συγχωνεύσεις και συμπτύξεις των αρχείων.

Εκτελέσετε το benchmark **kiwi-bench** με διαφορετικά πλήθη εισαγόμενων και αναζητούμενων στοιχείων και να παρακολουθήσετε τα μηνύματα στο τερματικό. Τρέξετε τον κώδικα στο gdb (**gdb kiwi-bench**) για να παρακολουθήσετε τις συναρτήσεις που εκτελούνται κατά τις διαφορετικές εκτελέσεις. Είναι χρήσιμο να κάνετε βηματική εκτέλεση with **step** ή **next**, να εισαγάγετε breakpoints με **break**, να δείτε την στοίβα εκτέλεσης με **bt**, να ιχνηλατήσετε την εκτέλεση κάποιου συγκεκριμένου νήματος με **thread**.

Η βασική σας εργασία είναι να σχεδιάσετε και να υλοποιήσετε την πολυνηματική εκτέλεση των λειτουργιών add και get στην μηχανή αποθήκευσης. Μια τετριμμένη προσέγγιση είναι να δημιουργήσετε πολλαπλά νήματα στην εφαρμογή benchmark (π.χ., **kiwi.c**) και να τα χρησιμοποιήσετε προκειμένου να εκτελέσετε ταυτόχρονες λειτουργίες add και get. Η προσέγγιση αυτή είναι λανθασμένη επειδή μπορεί να δημιουργήσει συνθήκες ανταγωνισμού μεταξύ των

νημάτων της εφαρμογής και του νήματος σύμπτυξης της μηχανής αποθήκευσης. Οι συνθήκες ανταγωνισμού μπορεί να προκύψουν μεταξύ λειτουργιών που εκτελούν μόνο get, μόνο add ή μίγμα από add και get.

1. Επομένως, το πρώτο σας βήμα είναι να διερευνήσετε τα υπάρχοντα νήματα που υπάρχουν στην εφαρμογή και τη μηχανή καθώς και τις λειτουργίες συγχρονισμού (αμοιβαίος αποκλεισμός, μεταβλητές συνθήκης) που υπάρχουν ήδη στην υλοποίηση που σας δίνεται. Η διερεύνηση μπορεί να γίνει με απλή αναζήτηση για κλήσεις συναρτήσεων της βιβλιοθήκης pthreads στα αρχεία πηγαίου κώδικα (π.χ., με **grep** στο τερματικό) και την ιχνηλάτηση της εκτέλεσης με χρήση του gdb. Τα σενάρια εκτέλεσης θα πρέπει να περιλαμβάνουν ακολουθίες add που εισαγάγουν νέα δεδομένα στη μηχανή και ακολουθίες get που κάνουν ερωτήσεις στα υπάρχοντα δεδομένα. Είναι σημαντικό να πειραματιστείτε με μικρά και μεγάλα σύνολα δεδομένων (μέχρι μερικές δεκάδες χιλιάδες κλειδιών) για να δείτε τη συμπεριφορά του συστήματος καθώς τα δεδομένα μετακινούνται από τη μνήμη στο δίσκο και δημιουργούνται αρχεία δεδομένων σε διάφορα επίπεδα της δομής αποθήκευσης.
2. Μια απλή προσέγγιση για την επίλυση του προβλήματος ταυτοχρονισμού είναι η εφαρμογή αμοιβαίου αποκλεισμού στην εκτέλεση των add και get λειτουργιών προκειμένου να διασφαλιστεί ότι μόνο μία λειτουργία κάθε φορά χρησιμοποιεί τη μηχανή. Όμως, θα πρέπει να φροντίσετε την αντιμετώπιση πιθανών συγκρούσεων των νημάτων της εφαρμογής με το νήμα συγχώνευσης της μηχανής που εκτελείται στο υπόβαθρο. Άρα, στην πειραματική σας επαλήθευση θα πρέπει να επιδιώξετε την ταυτόχρονη εκτέλεση συγχωνεύσεων μαζί με τις λειτουργίες add ή get και η υλοποίησή σας να τις εκτελεί σωστά.
3. Μια δεύτερη πιο προηγμένη λύση θα εφαρμόσει συγχρονισμό αναγνωστών-γραφέων στην εκτέλεση των λειτουργιών add και get. Για το σκοπό αυτό θα πρέπει να προσδιορίσετε υλοποίηση συγχρονισμού αναγνωστών-γραφέων με pthreads και θα την εισαγάγετε στις λειτουργίες db_add και db_get της μηχανής αποθήκευσης. Ξανά είναι σημαντικό στη σχεδίαση και υλοποίηση να λάβετε υπόψη την ταυτόχρονη εκτέλεση του νήματος συγχώνευσης και να δείξετε πειραματικά ότι η λύση σας δουλεύει σωστά.
4. Για διευκόλυνση της πειραματικής σας επαλήθευσης, μπορείτε να επεκτείνετε τα ορίσματα της γραμμής εντολής του αρχείου **bench.c** με επιπλέον επιλογές, όπως τη δυνατότητα να προσδιορίσετε το μίγμα λειτουργιών add και get καθώς και το ποσοστό από τον κάθε τύπο (π.χ., 50-50, 10-90, 0-100 κλπ).
5. Για να συγκρίνετε τις λύσεις σας με την αρχική υλοποίηση που σας δόθηκε, θα πρέπει να διατηρείτε στατιστικά της απόδοσης των λειτουργιών add και get (π.χ., χρόνο απόκρισης ή ρυθμοαπόδοση) με τον απαραίτητο συγχρονισμό για τον αμοιβαίο αποκλεισμό μεταξύ των διαφορετικών νημάτων που ενημερώνουν τις αντίστοιχες μεταβλητές. Στο τέλος, μπορείτε να τυπώνετε τα στατιστικά που μετρήσατε.

Η βαθμολόγηση θα γίνει με βάση τις λειτουργίες που έχετε υλοποιήσει, την ποιότητα του κώδικα που έχετε προσθέσει και την κατανοητή τεκμηρίωσή του στην αναφορά που παραδίδετε.

4. Τι θα παραδώσετε

Μπορείτε να προετοιμάσετε την λύση ατομικά ή σε ομάδες μέχρι τριών ατόμων. Η υποβολή θα γίνει από ένα μέλος της ομάδας μόνο. Υποβολή μετά την λήξη της προθεσμίας οδηγεί σε αφαίρεση 10% του βαθμού ανά ημέρα μέχρι 50%. Για παράδειγμα, αν στείλετε την λύση σας 1 ώρα μετά την προθεσμία, ο μέγιστος βαθμός που μπορείτε να λάβετε είναι 9 στα 10. Αν στείλετε την λύση μία εβδομάδα μετά την προθεσμία, ο μέγιστος βαθμός σας πέφτει σε 5 από τα 10. Υποβάλετε την λύση σας εγκαίρως με την εντολή

/usr/local/bin/turnin lab1_24@myy601 Report.pdf kiwi-source.zip

Θα βαθμολογηθείτε σύμφωνα με την περιγραφή που περιέχεται στο αρχείο **Report.pdf** και την συνοδευόμενη υλοποίηση στο αρχείο **kiwi-source.zip**. Στο αρχείο kiwi-source.zip περιλάβετε

μόνο τα αρχεία που έχετε αλλάξει με προσδιορισμό του μονοπατιού στο οποίο βρίσκονται στον πηγαίο κώδικα που σας δίνεται. Η αναφορά σας θα περιλαμβάνει τα ονόματα των μελών της ομάδας και τον αριθμό μητρώου σας. Επιπλέον περιγράφει τι έχετε υλοποιήσει και πώς το κάνατε περιγράφοντας μία-μία τις γραμμές κώδικα που προσθέσατε ή τροποποιήσατε. Γραμμές κώδικα που αλλάξατε ή προσθέσατε αλλά δεν περιλαμβάνονται στην αναφορά δεν μετράνε στον τελικό βαθμό ή μπορεί να μετρήσουν αρνητικά αν δείχνουν βασικές ελλείψεις στις προγραμματιστικές γνώσεις σας.

Ενθαρρύνεστε να προσθέσετε σχόλια στον πηγαίο κώδικα που γράψατε προκειμένου να διευκολύνετε τον εξεταστή σας να κατανοήσει την εργασία σας. Θα πρέπει επιπλέον να συμπεριλάβετε στην αναφορά σας την έξοδο της τελικής εντολής **make** καθώς και την έξοδο από την εκτέλεση με διάφορες παραμέτρους. Στην έξοδο εκτέλεσης συμπεριλάβετε μόνο την εντολή που χρησιμοποιήσατε για να τρέξετε το **kiwi-bench** και τις μετρήσεις απόδοσης χωρίς τις λεπτομερείς πληροφορίες αποσφαλμάτωσης που παράγει η εκτέλεση, εκτός και αν υπάρχει κάτι συγκεκριμένο που θέλετε να δείξετε. **Αν οι έξοδοι εκτέλεσης που περιλαμβάνονται στην αναφορά δεν αντιστοιχούν στον κώδικα που παραδίδετε, θα μηδενιστεί ολόκληρη η άσκηση για ευνόητους λόγους.**

Το αρχείο **kiwi-source.zip** που υποβάλετε είναι ένα συμπιεσμένο αρχείο με τα τροποποιημένα αρχεία. Για κάθε αρχείο που αλλάξατε θα στείλετε μόνο το τροποποιημένο και όχι το αρχικό. Θα πρέπει να εκτελέσετε **make clean** στον κατάλογο **kiwi-source** πριν δημιουργήσετε το αρχείο zip για να μην συμπεριλάβετε αρχεία εκτελέσιμου κώδικα (.o, .a, κλπ). Ο κώδικας που παραδίδετε θα πρέπει να μπορεί να μεταγλωττιστεί με τα εργαλεία που περιλαμβάνονται στην εικονική μηχανή που σας δίνεται.