ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Τεχνολογίας Υπολογιστών και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Σχεδιασμός Ενσωματωμένων Συστημάτων
9ο Εξάμηνο ΗΜΜΥ

2η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Ασκήσεις στη Βελτιστοποίηση Δυναμικών Δομών Δεδομένων (Dynamic Data Type Refinement – DDTR)

Ο σκοπός της άσκησης είναι να βελτιστοποιηθούν οι δυναμικές δομές δεδομένων δύο δικτυακών εφαρμογών: του Deficit Round Robin (**DRR**) και του αλγορίθμου **Dijkstra**, με χρήση της μεθοδολογίας «Βελτιστοποίησης Δυναμικών Δομών Δεδομένων» - Dynamic Data Type Refinement (**DDTR**). Οι δυναμικές δομές των αλγορίθμων DRR και Dijkstra θα βελτιστοποιηθούν ως προς:

- τις προσβάσεις στη μνήμη (**memory accesses**) που απαιτούνται για να προσπελαστουν τα δεδομένα τους
- και ως προς τη μέγιστη ποσότητα μνήμης που καταλαμβάνουν (**memory footprint**).

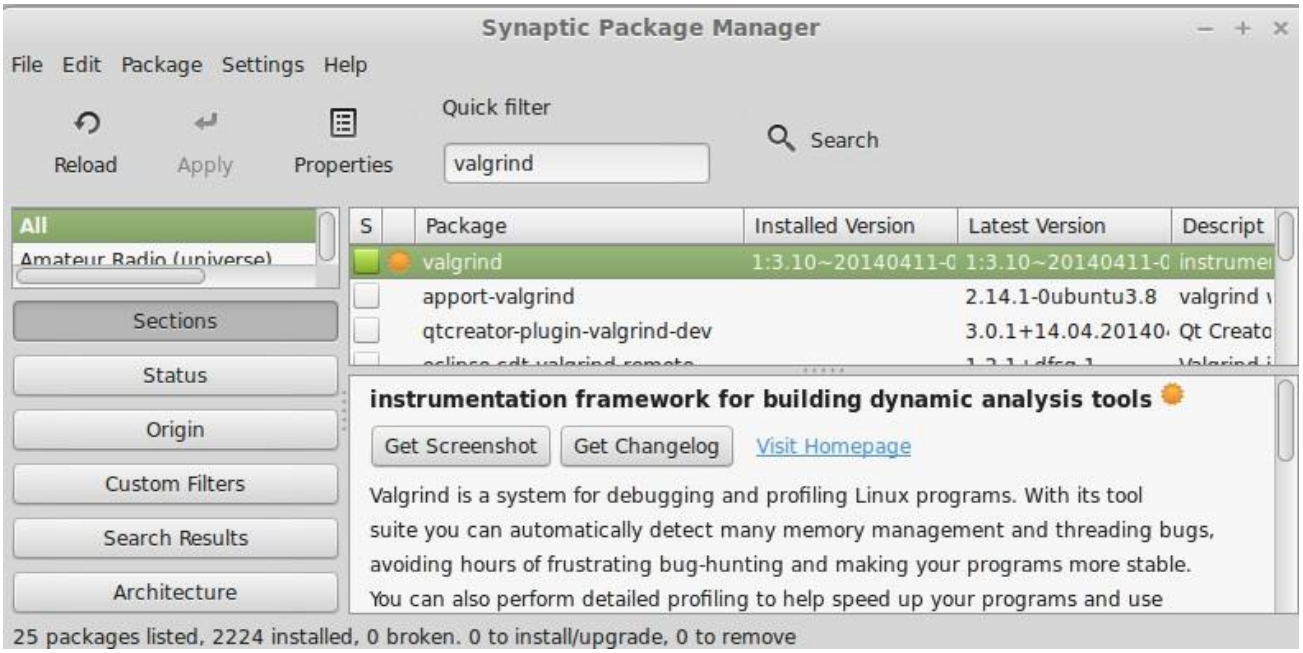
Για το evaluation των data structures θα χρησιμοποιηθούν τα εξής εργαλεία:

- Η βιβλιοθήκη DDTR.
- Το εργαλείο **Massif** της Valgrind suite.
- Το εργαλείο **Lackey** της Valgrind suite.

Α. Προετοιμασία για χρήση της βιβλιοθήκης

1. Το directory DDTR περιέχει:
   a. **synch_implementations**: Είναι η DDTR library, precompiled. Μπορεί να χρησιμοποιηθεί απευθείας και δεν χρειάζεται να ξαναγίνει compiled.
   b. **DRR**: Η εφαρμογή Deficit Round Robin.
   c. **dijkstra**: Η εφαρμογή Dijkstra.
2. Προαπαιτούμενα:
   a. **Linux – gcc** – **pthreads** (e.g. gcc 4.8)

b. **Valgind**: http://valgrind.org
Το Valgrind είναι ένα εργαλείο που παρέχει πληροφορίες για την εφαρμογή, διαβάζοντας το εκτελέσιμο αρχείο και όχι τον source code (i.e. dynamic instrumentation profiling tool). Χρησιμοποιείται για υπολογισμό της χρήσης της μνήμης (memory utilization), των προσβάσεων στη μνήμη (memory accesses), της προβληματικής χρήσης της μνήμης (memory leaks), λαθών σε πολυνηματικές εφαρμογές (threading errors) etc. **Εγκαταστήστε το Valgrind από τον package manager της έκδοσης linux που χρησιμοποιείτε**. Εναλλακτικά, μπορείτε να το κατεβάσετε και να το εγκαταστήσετε από εδώ: http://valgrind.org/downloads/current.html.



Β. Επεξήγηση της διαδικασίας βελτιστοποίησης στον αλγόριθμο Deficit Round Robin.

Σημείωση: Θα σας βοηθήσει, παράλληλα με την ανάγνωση της επεξήγησης, να βλέπετε τα αρχεία **drr.h** και **drr.c** της εφαρμογής DRR.

1. Το Deficit Round Robin είναι ένας **αλγόριθμος δρομολόγησης πακέτων (packet scheduling)** που τρέχει σε routers και χρησιμοποιείται για τη «δίκαιη» εξυπηρέτηση των πακέτων ενός αριθμού κόμβων (Nodes). Ο κώδικας προσομοιώνει την λειτουργία του αλγορίθμου και χρησιμοποιεί πραγματικά network trace files: http://crawdad.org/dartmouth/campus/.
2. Πιο συγκεκριμένα, κάθε κόμβος έχει ένα πεδίο που ονομάζεται "**deficit**", το οποίο αυξάνει σε κάθε «επίσκεψη» του δρομολογητή (scheduler), ενώ μειώνεται κάθε φορά που ο scheduler προωθεί ένα πακέτο από τον συγκεκριμένο κόμβο. Ο scheduler αποφασίζει από ποιον κόμβο θα προωθήσει ένα ή περισσότερα πακέτα συγκρίνοντας το "deficit" κάθε κόμβου με το packet size του πρώτου πακέτου του συγκεκριμένου κόμβου που περιμένει να εξυπηρετηθεί. Ο scheduler:
    a. Αυξάνει το deficit όλων των κόμβων κατά QUANTUM και στη συνέχεια, για κάθε κόμβο:
    b. Αν το deficit του κόμβου ξεπερνά το μέγεθος του πρώτου πακέτου στον συγκεκριμένο κόμβο αυτό το πακέτο προωθείται. Το deficit του κόμβου μειώνεται κατά το μέγεθος του πακέτου (ώστε οι πιθανότητες να προωθηθεί πακέτο από τον ίδιο κόμβο στην επόμενη «επισκεψη» του scheduler,

να μειωθούν).
    c. Αν το deficit του κόμβου είναι μικρότερο από το μέγεθος του πρώτου πακέτο, ο scheduler προχωρά στον επόμενο κόμβο.
3. Ο κώδικας περιέχει δύο δομές (structures): **Node**: που είναι οι ενεργοί κόμβοι, και **Packet** που είναι το πακέτο που πρόκειται να προωθηθεί.
4. Οι δομές δεδομένων του αλγορίθμου είναι οι εξής:
    a. Μια δομή δεδομένων στην οποία είναι αποθηκευμένοι οι Nodes: **clientList**.
    b. Κάθε Node έχει αποθηκευμένη σε μια data structure τα πακέτα που θα προωθήσει: **pList**.
5. Θα δοκιμάσουμε διαφορετικές υλοποιήσεις των δύο αυτών data structures: Απλά συνδεδεμένη λίστα - Single Linked List (**SLL**), Διπλά Συνδεδεμένη Λίστα - Double Linked List (**DLL**) και Δυναμικός Πίνακας - Dynamic Array (ή vector) (**DYN_ARR**). Με χρήση της βιβλιοθήκης DDTR και των εργαλείων Valgrind θα τις κάνουμε evaluation ως προς τον αριθμό των προσβάσεων στη μνήμη - **memory accesses** και το μέγιστο μέγεθος μνήμης που απαιτείται - **memory footprint**.
6. Βήμα 1: Εισαγωγή της βιβλιοθήκης DDTR στην εφαρμογή:
    a. Κάνουμε include τα αντίστοιχα header files (δείτε το drr.h):
        i. Για την Απλά Συνδεδεμένη Λίστα - **Single Linked List (SLL)**: #include "../synch_implementations/cdsl_queue.h"
        ii. Για τη Διπλά Συνδεδεμένη Λίστα - **Double Linked List (DLL)**: #include "../synch_implementations/cdsl_deque.h"
        iii. Για τον Δυναμικό Πίνακα - **Dynamic Array (DYN_ARR)**: #include "../synch_implementations/cdsl_dyn_array.h"

    Σε αυτά τα 3 αρχεία βρίσκονται οι δηλώσεις των συναρτήσεων (function declarations) που μπορούν να χρησιμοποιηθούν στον κώδικα της εφαρμογής για να αντικαταστήσουν τη δομή δεδομένων της εφαρμογής με αυτές της βιβλιοθήκης DDTR.
    (**δείτε το drr.h – γραμμή 10**)
    b. **Αντικαθιστούμε τις δηλώσεις (definitions – declarations) των δομών δεδομένων** της εφαρμογής με αυτά της βιβλιοθήκης DDTR. Παραδείγματα:
        i. cdsl_sll *pList; (Το pList είναι ένας δείκτης (pointer) σε μια απλά συνδεδεμένη λίστα – single linked list)
        ii. cdsl_dll *pList; (Το pList είναι ένας δείκτης (pointer) σε μια διπλα συνδεδεμένη λίστα - double linked list)
        iii. cdsl_dyn_array *clientList (Το clientList είναι ένας δείκτης (pointer) σε έναν δυναμικό πίνακα - dynamic array).

    (**δείτε το drr.c – γραμμή 23 ή το drr.h – γραμμή 31**)
    c. **Δημιουργούμε και κάνουμε αρχικοποίηση (initialization) τις δομές δεδομένων** καλώντας τις αντίστοιχες **cdsl_XXX_init()** συναρτήσεις (functions) από τη βιβλιοθήκη. (XXX είναι η συγκεκριμένη υλοποίηση). Παραδείγματα:
        i. clientList = cdsl_sll_init(); (Μία απλά συνδεδεμένη λίστα δημιουργείται, αρχικοποιείται και το clientList δείχνει σε αυτήν).
        ii. new_node->pList = cdsl_dll_init(); (Μία διπλά συνδεδεμένη λίστα δημιουργείται, αρχικοποιείται και το pList δείχνει σε αυτήν).

    (**δείτε το drr.c – γραμμή 45 ή 125**)

d. **Αντικαθιστούμε τις συναρτήσεις των λειτουργιών των δομών δεδομένων (data structure operation functions) της εφαρμογής με αυτά της βιβλιοθήκης**. Οι δηλώσεις των αντίστοιχων συναρτήσεων (function declarations) της βιβλιοθήκης βρίσκονται στα header files που αναφέρονται στο (a). Παραδείγματα:

i. node->pList->**enqueue**(0, node->pList, (void*)packet); (Εισαγωγή του pointer "packet" στο data structure pList).
Όρισμα (Argument) 1: Ένας unsigned int (δεν έχει σημασία σε μονονηματικές εφαρμογές, βάζετε ότι θέλετε)
Όρισμα (Argument) 2: Δείκτης (pointer) στη δομή δεδομένων που επιτελούμε τη λειτουργία.
Όρισμα (Argument) 3: Void pointer του στοιχείου που εισάγουμε.

ii. v->pList->**dequeue**(0, (v->pList));
Όρισμα (Argument) 1: Ένας unsigned int (δεν έχει σημασία σε μονονηματικές εφαρμογές, βάζετε ότι θέλετε)
Όρισμα (Argument) 2: Δείκτης (pointer) στη δομή δεδομένων που επιτελούμε τη λειτουργία.

iii. iterator_cdsl_dyn_array it, end;
Δύο **iterators** σε dynamic array γίνονται defined. Το interface τους είναι παρόμοι με την STL και βρίσκεται στα header files του (a).

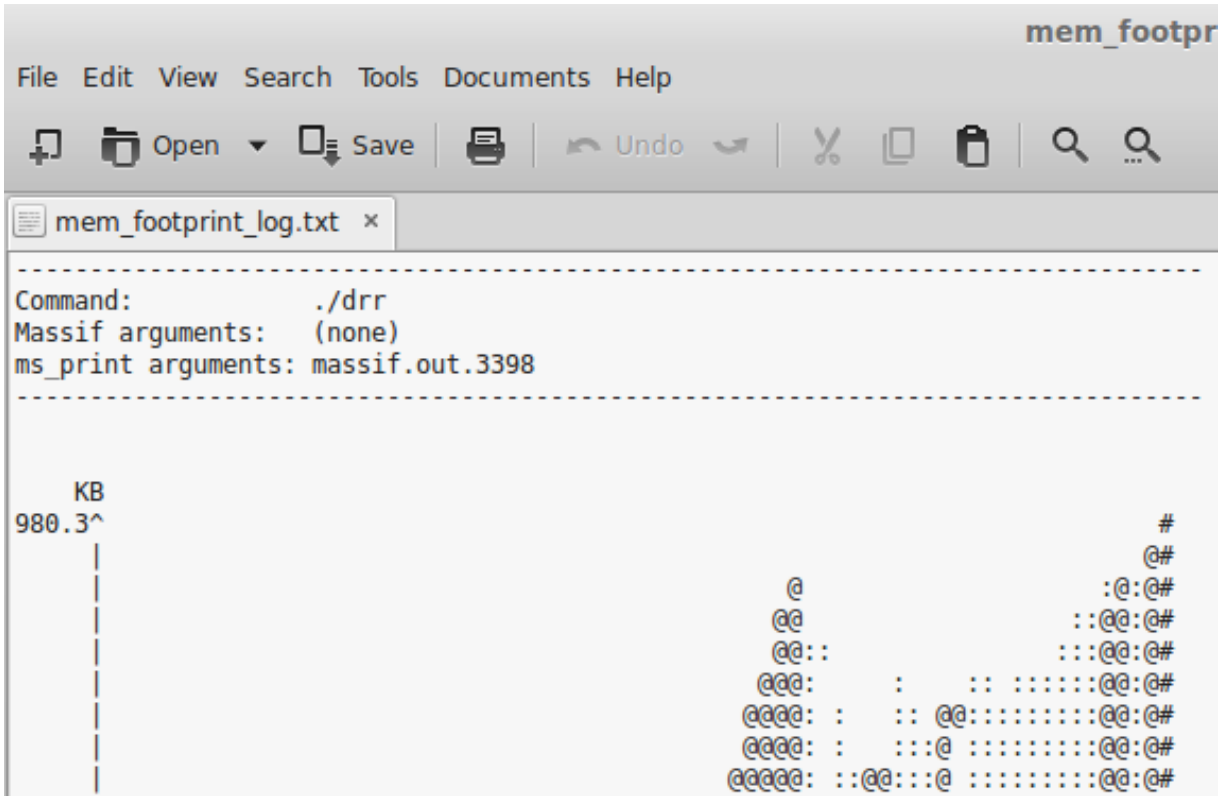
(**δείτε το drr.c – γραμμή 131 ή γραμμή 180**)

7. Compilation and Εξαγωγή αποτελεσμάτων:

a. **Compilation της εφαρμογής DRR**:
gcc drr.c -o drr -pthread -lcdsl -L./../synch_implementations -I./../synch_implementations

b. ***Αποτελέσματα αριθμού προσβάσεων στη μνήμη (Memory accesses)****:*
Δημιουργία ενός trace file των προσβάσεων στη μνήμη με χρήση του εργαλείου lackey:
Εντολή: valgrind --log-file="mem_accesses_log.txt" --tool=lackey --trace-mem=yes ./drr Καταμέτρηση του αριθμού των προσβάσεων στη μνήμη από το trace file:
Εντολή: cat mem_accesses_log.txt | grep 'I\| L' | wc -l
Σημείωση: Η διαδικασία παραγωγής του trace file συχνά κρατά αρκετό χρόνο (π.χ. 10 – 20 λεπτά) και το αρχείο που παράγεται είναι 5 – 15GB.

c. ***Αποτελέσματα μέγιστης χρήσης μνήμης (Memory footprint)****:*
Δημιουργία ενός log αρχείου και επεξεργασία του, με χρήση του εργαλείου massif:
Εντολή: valgrind --tool=massif ./drr
Εντολή: ms_print massif.out.XXXXX > mem_footprint_log.txt
(όπου XXXXX ο αριθμός που αντιστοιχεί στο αρχείο που παράχθηκε).
Στο mem_footprint_log.txt, το memory footprint είναι η μέγιστη τιμή του άξονα y στο διάγραμμα. Π.χ. στο παρακάτω πείραμα το memory footprint είναι 980.3 KB.



C. Ασκήσεις

Άσκηση 1: Βελτιστοποίηση δυναμικών δομών δεδομένων του αλγορίθμου DRR

Ο source code του DRR έχει ήδη περασμένη την βιβλιοθήκη DDTR.

a) Εκτελέστε την εφαρμογή **με όλους τους διαφορετικούς συνδυασμούς υλοποιήσεων δομών δεδομένων για τη λίστα των πακέτων και τη λίστα των κόμβων** (Σύνολο 9 συνδυασμοί). Για κάθε συνδυασμό καταγράψτε τα αποτελέσματά του σχετικά με τον αριθμό των προσβάσεων στη μνήμη (**memory accesses**) και το μέγεθος της απαιτούμενης μνήμης (**memory footprint**).

b) Βρείτε τον συνδυασμό υλοποιήσεων δομών δεδομένων με την οποία η εφαρμογή έχει τον μικρότερο αριθμό προσβάσων στη μνήμη (**minimum number of memory accesses**)**.**

c) Βρείτε τον συνδυασμό υλοποιήσεων δομών δεδομένων με την οποία η εφαρμογή έχει μικρότερες απαιτήσεις σε μνήμη (**smaller memory footprint**).

Άσκηση 2: Βελτιστοποίηση δυναμικών δομών δεδομένων του αλγορίθμου Dijkstra

Ο αλγόριθμος dijkstra βρίσκει τη συντομότερη διαδρομή (shortest path) σε έναν πίνακα μεγέθους 100x100. Οι κόμβοι που εξετάζει και αποτελούν τη συντομότερη διαδρομή (shortest path) αποθηκεύονται σε μια λίστα. Εφαρμόστε τη μεθοδολογία DDTR για την εφαρμογή αυτή:

a) **Εκτελέστε την εφαρμογή** και **καταγράψτε τα αποτελέσματα** που παράγει.

b) **Εισάγετε τη βιβλιοθήκη** στην εφαρμογή και αντικαταστήστε τη δομή δεδομένων της, με τις δομές δεδομένων της βιβλιοθήκης. Εκτελέστε την εφαρμογή και βεβαιωθείτε ότι η βιβλιοθήκη έχει εισαχθεί

σωστά, συγκρίνοντας τα αποτελέσματα που παράγονται με αυτά που καταγράψατε στο ερώτημα (a). (Πρέπει προφανώς να είναι ακριβώς τα ίδια).

c) **Εκτελέστε την εφαρμογή** για τις εξής δυναμικές δομές δεδομένων: Απλά Συνδεδεμένη Λίστα – Single Linked List (**SLL**), Διπλά Συνδεδεμένη Λίστα – Double Linked List (**DLL**) και Δυναμικό Πίνακα – Dynamic Array (**DYN_ARR**). **Καταγράψτε τα αποτελέσματα** του αριθμού προσβάσεων στη μνήμη (**memory accesses**) και του μέγιστου μεγέθους μνήμης που απαιτείται για κάθε υλοποίηση που δοκιμάζετε (**memory footprint**).

d) Βρείτε την υλοποιήση δομής δεδομένων με την οποία η εφαρμογή έχει τον μικρότερο αριθμό προσβάσων στη μνήμη (**lowest number of memory accesses**)**.**

e) Βρείτε την υλοποιήση δομής δεδομένων με την οποία η εφαρμογή έχει μικρότερες απαιτήσεις σε μνήμη (**lowest memory footprint**).

D. Επεξήγηση όρων

Argument	Όρισμα μιας συνάρτησης
Data Structure Operation Functions	Αναφέρεται στις συναρτήσεις που υλοποιούν τις λειτουργίας μιας δομής δεδομένων. Π.χ. enqueue, dequeue, insert, remove, find, etc.
DDTR (Dynamic Data Type Refinement)	Η μεθοδολογία βελτιστοποίησης δυναμικών δομών δεδομένων
Directory	Κατάλογος του Linux
DLL (Double Linked List)	Διπλά συνδεδεμένη λίστα
DRR (Deficit Round Robin)	Αλγόριθμος δρομολόγησης πακέτων
(DYN_ARR) Dynamic Array	Πίνακας που το μέγεθός του αυξάνεται κατά τη διάρκεια εκτέλεσης της εκτέλεσης της εφαρμογής, ανάλογα με τον αριθμό δεδομένων που έχει αποθηκευμένα.
Function	Συνάρτηση της γλώσσας C.
Function Definition	Ο ορισμός μιας συνάρτησης (π.χ. int myFun(int x) { return x+1; })
Function Declaration	Η δήλωση μιας συνάρτησης (π.χ. int myFun(int x);)
Initialization	Αρχικοποίηση. Εδώ, αναφέρεται στην αρχικοποίηση μιας δομής δεδομένων της βιβλιοθήκης DDTR.
Interface	Εδώ, αναφέρεται στην «επικοινωνία» ανάμεσα στις δομές δεδομένων της βιβλιοθήκης και στην εφαρμογή.
Memory Accesses	Προσβάσεις στη μνήμη
Memory Footprint	Μέγιστο μέγεθος μνήμης που απαιτείται από την εφαρμογή
SLL (Single Linked List)	Απλά συνδεδεμένη λίστα