**ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ**

**Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών**


*Θεμελιώδη Θέματα Επιστήμης Υπολογιστών*

*3η Σειρά Ασκήσεων (2018-19)*

Ονοματεπώνυμο : Χρήστος Τσούφης

Α.Μ. : 03117176

Στο πλαίσιο της 3ης σειράς ασκήσεων, χρησιμοποιούμε την τεχνική Dynamic Data Type Refinement με σκοπό τη βελτιστοποίηση ως προς τη δεσμευμένη ποσότητα μνήμης καθώς και το πλήθος προσβάσεων σε εκείνη των εφαρμογών Deficit Round Robin και A*.

Οι μετρήσεις λήφθηκαν σε περιβάλλον Arch Linux x64 με τα εργαλεία Massif και Lackey της σουίτας Valgrind.

## **Άσκηση 1η (**Deficit Round Robin)

Η εφαρμογή έγινε compiled για τους εννέα διαφορετικούς συνδυασμούς υλοποιήσεων για τις δομές δεδομένων των πακέτων (*pList) και των κόμβων (*clientList). Οι συνδυασμοί αυτοί επιλέγονται μέσω των definitions στην αρχή του drr.h.

Αρχικά καταγράφηκαν οι προσβάσεις στη μνήμη των εκτελέσιμων αυτών με το Lackey, και στη συνέχεια το Massif έκανε profile των προγραμμάτων ως προς τη χρήση μνήμης.

Η εκτέλεση των εργαλείων και η λήψη των μετρήσεων αυτοματοποιήθηκε με το παρακάτω script:

```
#!/bin/sh

# Checksums for housekeeping
md5sum ./bin/* > ./binary_hashes.txt

for file in ./bin/*
do
   echo "Current binary: $file" &&
   valgrind --log-file="./cur_log.txt" --tool=lackey --trace-mem=yes $file &&
   sleep 2 && echo "$file" >> drr_results_mem_acc.txt &&
   cat ./cur_log.txt | grep 'I\| L' | wc -l >> drr_results_mem_acc.txt && rm -
   f ./cur_log.txt &&
   valgrind --tool=massif $file > /dev/null && sleep 1 &&
   ms_print ./massif.out.* > ./massif_results/drr_massif_$(echo $file | cut -d / -f 3).txt && sleep 1 && rm
   -f ./massif.out.* && sleep 2
done
```

Τα αποτελέσματα των μετρήσεων του Lackey φαίνονται στον παρακάτω πίνακα:

| | | **Υλοποίηση clientList** | | |
|---|---|---|---|---|
| | | *Queue* | *Dequeue* | *Vector* |
| **Υλοποίηση pList** | *Queue* | 9682906 | 9684304 | 9783106 |
| | *Dequeue* | 9760358 | 9762165 | 9860967 |
| | *Vector* | 73039436 | 73041243 | 73129507 |

Η δομή δεδομένων της απλά συνδεδεμένης λίστας (Queue, SLL) υλοποιείται από το cdsl_queue.h, της διπλά συνδεδεμένης λίστας (Deque, DLL) από το cdsl_deque.h και του δυναμικού πίνακα (Vector, Dynamic Array) από το cdsl_dyn_array.h.

Ο συνδυασμός ώστε η εφαρμογή να βελτιστοποιείται ως προς τις προσβάσεις στη μνήμη είναι η υλοποίηση και των δύο structures με απλά συνδεδεμένη λίστα (απαιτήθηκαν 9682906 προσβάσεις).

Παρομοίως, τα αποτελέσματα των μετρήσεων του Massif δίνονται παρακάτω:

**Υλοποίηση clientList**

| **Υλοποίηση pList** | *Queue* | *Dequeue* | *Vector* |
|---|---|---|---|
| *Queue* | 166.3 kB | 168.5 kB | 162.0 kB |
| *Dequeue* | 213.2 Kb | 214.8 Kb | 208.4 kB |
| *Vector* | 166.3 kB | 168.3 kB | 162.0 kB |

Προκύπτει από τα δεδομένα λοιπόν ότι οι βέλτιστοι συνδυασμοί δομών δεδομένων ως προς την ποσότητα της μνήμης που καταλαμβάνεται κατά το runtime της εφαρμογής είναι είτε δυναμικός πίνακας για τους clients και τα πακέτα είτε δυναμικός πίνακας για τους clients και απλά συνδεδεμένη λίστα για τα πακέτα (162 kB).

Βέβαια, το γεγονός ότι και οι δύο υλοποιήσεις χρησιμοποιούν σχεδόν την ίδια ποσότητα μνήμης οφείλεται σε πολλούς παράγοντες, και κυρίως στο testcase που χρησιμοποιείται στην συγκεκριμένη εκτέλεση του Round Robin. Αν για παράδειγμα είχαμε λίγα περισσότερα πακέτα στην pList (υλοποιημένη με vector) και ο δεσμευμένος χώρος είχε γεμίσει, το vector θα διπλασιαστεί στη μνήμη, συγκριτικά με την απλά συνδεδεμένη λίστα η οποία θα αυξηθεί γραμμικά με το πλήθος των πακέτων.

## **Άσκηση 2η (**Dijkstra)

- Όμοια με την άσκηση 1, χρησιμοποιώντας το αρχείο input.dat προκύπτουν τα αποτελέσματα:

```
Shortest path is 3 in cost. Path is:  16 41 45 51 68 2 71 47 79 23 77 1 58 33 32 66
Shortest path is 0 in cost. Path is:  17 65 73 36 46 10 58 33 13 19 79 23 91 67
Shortest path is 1 in cost. Path is:  18 15 41 45 51 68
Shortest path is 2 in cost. Path is:  19 69
christost@christost -VirtualBox:~/Desktop/DDTR_exercise/dijsktra$ gcc dijkstra.c -o dijkstra -pthread -lcdsl -L./../synch_implementatio
ns -I./../synch_implementations
christost@christost -VirtualBox:~/Desktop/DDTR_exercise/dijsktra$ ./dijkstra input.dat
Shortest path is 1 in cost. Path is:  0 41 45 51 50
Shortest path is 0 in cost. Path is:  1 58 57 20 40 17 65 73 36 46 10 38 41 45 51
Shortest path is 1 in cost. Path is:  2 71 47 79 23 77 1 58 57 20 40 17 52
Shortest path is 2 in cost. Path is:  3 53
Shortest path is 1 in cost. Path is:  4 85 83 58 33 13 19 79 23 77 1 54
Shortest path is 3 in cost. Path is:  5 26 23 77 1 58 99 3 21 70 55
Shortest path is 3 in cost. Path is:  6 42 80 77 1 58 99 3 21 70 55 56
Shortest path is 0 in cost. Path is:  7 17 65 73 36 46 10 58 57
Shortest path is 0 in cost. Path is:  8 37 63 72 46 10 58
Shortest path is 1 in cost. Path is:  9 33 13 19 79 23 77 1 59
Shortest path is 0 in cost. Path is:  10 60
Shortest path is 5 in cost. Path is:  11 22 20 40 17 65 73 36 46 10 29 61
Shortest path is 0 in cost. Path is:  12 37 63 72 46 10 58 99 3 21 70 62
Shortest path is 0 in cost. Path is:  13 19 79 23 77 1 58 99 3 21 70 55 12 37 63
Shortest path is 1 in cost. Path is:  14 38 41 45 51 68 2 71 47 79 23 77 1 58 33 13 92 64
Shortest path is 1 in cost. Path is:  15 13 92 94 11 22 20 40 17 65
Shortest path is 3 in cost. Path is:  16 41 45 51 68 2 71 47 79 23 77 1 58 33 32 66
Shortest path is 0 in cost. Path is:  17 65 73 36 46 10 58 33 13 19 79 23 91 67
Shortest path is 1 in cost. Path is:  18 15 41 45 51 68
Shortest path is 2 in cost. Path is:  19 69
```

- Στον κώδικα του αρχείο dijkstra.c πραγματοποιήθηκαν οι παρακάτω αλλαγές ώστε να εκτελεστεί για κάθε μία από τις 3 δομές δεδομένων (SLL, DLL, DYNAMIC ARRAY) και να προκύψουν τα ίδια αποτελέσματα με πριν.

```
#define SLL
//#define DLL
//#define DYN_ARR
#if defined(SLL)
#include "../synch_implementations/cdsl_queue.h"
#endif
#if defined(DLL)
#include "../synch_implementations/cdsl_deque.h"
#endif
#if defined(DYN_ARR)
#include "../synch_implementations/cdsl_dyn_array.h"
#endif
```

```
#if defined(SLL)
cdsl_sll *qHead;
iterator_cdsl_sll it;
#elif defined(DLL)
cdsl_dll *qHead;
iterator_cdsl_dll it;
#else
cdsl_dyn_array *qHead;
iterator_cdsl_dyn_array it;
#endif
```

```
void enqueue (int iNode, int iDist, int iPrev)
{
  QITEM *qNew = (QITEM *)  malloc(sizeof(QITEM));
     it = qHead -> iter_begin(qHead);
  if  (!qNew)
     {
       fprintf(stderr, "Out of memory.\n");
       exit(1);
     }
  qNew->iNode = iNode;
  qNew->iDist =  iDist;
  qNew->iPrev = iPrev;
  qHead->enqueue(0, qHead, (void*)qNew);
  g_qCount++;
}
```

```
void dequeue (int *piNode, int *piDist, int *piPrev)
{
  it = qHead -> iter_begin(qHead);
  QITEM *temp = (QITEM*)(qHead->iter_deref(qHead,it));
  if  (g_qCount != 0) {
    *piNode = temp -> iNode;
    *piDist = temp -> iDist;
    *piPrev = temp -> iPrev;
    g_qCount--;
  }
 qHead->dequeue(0, qHead);
}
```

- Τα αποτελέσματα που προέκυψαν για τον αριθμό προσβάσεων στη μνήμη (memory access) και το μέγεθος της απαιτούμενης μνήμης (memory footprint) μετά την εκτέλεση του αλγορίθμου για τις 3 δομές δεδομένων είναι τα εξής:

*memory access*

| | |
|---|---|
| SLL | 94494743 |
| DLL | 94672055 |
| DYN_ARR | 141252124 |

*memory footprint*

| | |
|---|---|
| SLL | 356.2KB<br>Command: ./dijkstra input.dat<br>Massif arguments: (none)<br>ms_print arguments: massif.out.3334<br>--------------------------------------<br><br>KB<br>356.2^ |
| DLL | 470.9KB<br>--------------------------------------<br>Command: ./dijkstra input.dat<br>Massif arguments: (none)<br>ms_print arguments: massif.out.3513<br>--------------------------------------<br><br>KB<br>470.9^ |
| DYN_ARR | 360.7KB<br>--------------------------------------<br>Command: ./dijkstra input.dat<br>Massif arguments: (none)<br>ms_print arguments: massif.out.3129<br>--------------------------------------<br><br>KB<br>360.7^ |

- Από τον 1ο πίνακα συμπεραίνει κανείς ότι τον μικρότερο αριθμό προσβάσεων στη μνήμη έχει η υλοποίηση με **SLL** (94494743 προσβάσεις στη μνήμη).
- Από τον 2ο πίνακα συμπεραίνει κανείς ότι τις μικρότερες απαιτήσεις σε μνήμη έχει η υλοποίηση με **SLL** (356.2KB).