# Κ18 - Υλοποίηση Συστημάτων Βάσεων Δεδομένων

## Χειμερινό Εξάμηνο 2019 – 2020

## Άσκηση 1

Το σύνολο συναρτήσεων που μας δοθήκαν να υλοποιήσουμε διαχειρίζονται αρχεία σωρού (Heap Files) που περιλαμβάνουν blocks τα οποία περιέχουν εγγραφές (records).

Πιο συγκεκριμένα, υλοποίησα το δικό μου heap file με την εξής δομή:

Το πρώτο μπλοκ του αρχείου περιέχει την πληροφορία εάν πρόκειται για αρχείο σωρού ή όχι χρησιμοποιώντας τον χαρακτήρα '1' (Heap File). Αυτό υλοποιείται στην συνάρτηση HP_CreateFile και έπειτα στην συνάρτηση HP_OpenFile γίνεται έλεγχος για το αν το αρχείο που πρόκειται να ανοίξουμε είναι Heap File ή όχι. Έπειτα, σε όλα τα υπόλοιπα μπλοκ εισήγαγα ένα ακέραιο πλήθος εγγραφων καθώς και κρατούσα έναν χαρακτήρα στην αρχη κάθε μπλοκ (έτσι ώστε να εχει την μορφη κεφαλιδας) που έδειχνε το πλήθος των εγγραφών την κάθε στιγμη μεσα στο μπλοκ (HP_InsertEntry). Όταν δεν χωρουσε επομενη εγγραφη στο μπλοκ γινοταν ξεκαρφιτσωμα του και αποδέσμευση μνημης και μετα δέσμευση καινούργιου μπλοκ και επαναληψη της παραπανω διαδικασίας.

## Ερώτημα Bonus

Εκτελώντας την εντολη strace –c ./build/runner έχοντας στο αρχειο hp_main.c την εντολη BF_Init(LRU) παρατηρούμε ότι στην στήλη calls εχουμε 106518 ενώ τρέχοντας την ιδια εντολή αλλα με την εντολη BF_Init(MRU) στο hp_main.c τα calls πέφτουν στα 57422.

Ο λογος των δυο MRU/LRU = 0.53908 δηλαδη στο περιπου ισουται με τον **λογο ½**.

Αυτό συμβαινει διοτι η μέθοδος αντικαταστασης LRU αντικαθιστα τα λιγοτερο προσφατα χρησιμοποιημένα μπλοκς όταν γεμίσει η ενδιάμεση μνήμη, σε αντίθεση με την MRU που αντικαθιστά τα πιο πρόσφατα χρησιμοποιημένα μπλοκς.

Εστω ότι εχουμε την εξης ενδιαμεση μνημη:

| | | | |
|---|---|---|---|
| Cell1 | Cell2 | Cell3 | Cell4 |

Και το εξης Heap File:

| | | | | | |
|---|---|---|---|---|---|
| Block1 | Block2 | Block3 | Block4 | Block5 | Block6 |

Και εστω ότι το οι επαναληπτικές σαρώσεις του αρχειου είναι 2.
Τότε θα συμβει το εξης:

Αρχικά στην πρωτη σαρωση θα γεμίσει η ενδιάμεση μνημη με τα 4 πρώτα blocks του heap file:

| | | | |
|---|---|---|---|
| Block1 | Block2 | Block3 | Block4 |

*Α)* Αν η μεθοδος αντικαταστασης είναι η LRU τοτε θα συμβει το εξης:

| | | | |
|---|---|---|---|
| Block5 | Block2 | Block3 | Block4 |
| Block5 | Block6 | Block3 | Block4 |

Και στην δευτερη σαρωση δεν θα βρεθει καποιο Block που υπηρχε από πριν στην μνημη και ουσιαστικα είναι ετοιμο. Καταλήγουμε αρα στην εξης γενικη περιπτωση:

Αν εχουμε ένα αρχειο σωρου με B πλήθος μπλοκς και ενδιάμεση μνημη με χωρητικότητα M πληθος κελιών τοτε ισχυει το εξης:

Για S σαρωσεις θα εχουμε **B*S read calls**.

*B)* Αν η μεθοδος αντικαταστασης είναι η MRU τοτε θα συμβει το εξης:

| | | | |
|---|---|---|---|
| Block1 | Block2 | Block3 | Block5 |
| Block1 | Block2 | Block3 | Block6 |

Και στην δεύτερη σάρωση θα βρεθουν 3 Blocks που υπηρχαν από πριν στην μνημη και είναι ετοιμα. Καταληγουμε αρα στην εξης γενική περιπτωση:

Αν έχουμε ένα αρχειο σωρου με πληθος B μπλοκς και ενδιαμεση μνημη με χωρητικότητα M πληθος κελιών τοτε ισχυει το εξης:

*I)* 1η σαρωση θα εχουμε B read calls

*II)* Για ολες τις S-1 υπολοιπες σαρωσεις θα εχουμε B - (M – 1) read calls

Αρα τελικα **B + (S – 1)(B – M + 1) read calls.**

Παίρνοντας λοιπον τις δυο τιμες των read calls για LRU και MRU και υπολογίζοντας τον λόγο τους ενώ οι επαναληπτικές σαρώσεις τείνουν στο άπειρο εχουμε το εξης οριο:

$\lim_{s->oo}$ MRU/LRU = $\lim_{s->oo}$ B + (S – 1)*(B – M + 1) / B*S
= $\lim_{s->oo}$ B – M + 1 / B
που στην δικη μας περιπτωση αν αντικαταστήσουμε οπου B = 6, S = 2, M = 4 εχουμε:
$\lim_{s->oo}$ MRU/LRU = 3/6 = ½.

Καταλήγουμε αρα στο συμπέρασμα οτι στην δωθείσα κατάσταση η μεθοδος MRU είναι πιο αποδοτική από την LRU και μπορει να ριξει τα read calls στο ήμισυ.