Πληροφορική & Τηλεπικοινωνίες

Κ18 - Υλοποίηση Συστημάτων Βάσεων Δεδομένων

Χειμερινό Εξάμηνο 2021 – 2022

Καθηγητής Ι. Ιωαννίδης

Άσκηση 2 – Παράδοση 16/01/2022

Η 2η εργασία αφορά στην υποστήριξη δευτερευόντων ευρετηρίων σε Συστήματα Βάσεων Δεδομένων. Επίσης αφορά σε μία βασική λειτουργία που πραγματοποιείται συχνά στις βάσεις δεδομένων, που είναι η ζεύξη. Αποτελεί συνέχεια της 1ης εργασίας επειδή χρησιμοποιεί επεκτατό κατακερματισμό.
Πιο συγκεκριμένα, στα πλαίσια της 2ης εργασίας θα υλοποιήσετε ένα σύνολο συναρτήσεων που δημιουργούν και διαχειρίζονται αρχεία δευτερευόντων ευρετηρίων βάσει επεκτατού κατακερματισμού.
Στη συνέχεια θα υλοποιήσετε την λειτουργία της ζεύξης για δύο αρχεία χρησιμοποιώντας το κλειδί των δευτερευόντων αρχείων.
Οι συναρτήσεις που καλείστε να υλοποιήσετε αφορούν τη διαχείριση εγγραφών και τη διαχείριση ευρετηρίων. Η υλοποίησή τους θα γίνει πάνω από το επίπεδο διαχείρισης μπλοκ υποχρεωτικά, το οποίο δίνεται έτοιμο ως βιβλιοθήκη. Τα πρωτότυπα (definitions) των συναρτήσεων που καλείστε να υλοποιήσετε όσο και των συναρτήσεων της βιβλιοθήκης επιπέδου μπλοκ δίνονται στη συνέχεια, μαζί με επεξήγηση για τη λειτουργικότητα που θα επιτελεί η κάθε μία.
Οι εγγραφές έχουν τη μορφή που δίνεται στη συνέχεια. Το id είναι το κλειδί.

```
typedef struct{
        int id,
        char name[15];
        char surname[20];
        char city[20];
}Record;
```

Το πρώτο μπλοκ (block) κάθε αρχείου περιλαμβάνει “ειδική” πληροφορία σχετικά με το ίδιο το αρχείο.
Η πληροφορία αυτή χρησιμεύει στο να αναγνωρίσει κανείς αν πρόκειται για αρχείο πρωτεύοντος ή δευτερεύοντος ευρετηρίου και όποια άλλη πιθανή πληροφορία νομίζεται ότι χρειάζεται.
Στη συνέχεια δίνεται ένα παράδειγμα των εγγραφών που θα περιέχουν τα αρχεία.

{15, “Giorgos”, “Dimopoulos”, “Ioannina”}
{4, “Antonia”, “Papadopoulou”, “Athina”}
{300, “Yannis”, “Yannakis”, “Thessaloniki”}

Συναρτήσεις BF (Block File)

Το επίπεδο block (BF) είναι ένας διαχειριστής μνήμης (memory manager) που λειτουργεί σαν κρυφή μνήμη (cache) ανάμεσα στο επίπεδο του δίσκου και της μνήμης. Το επίπεδο block

κρατάει block δίσκου στην μνήμη. Κάθε φορά που ζητάμε ένα block δίσκου, το επίπεδο BF πρώτα εξετάζει την περίπτωση να το έχει φέρει ήδη στην μνήμη. Αν το block υπάρχει στην μνήμη τότε δεν το διαβάζει από τον δίσκο, σε αντίθετη περίπτωση το διαβάζει από τον δίσκο και το τοποθετεί στην μνήμη. Επειδή το επίπεδο BF δεν έχει άπειρη μνήμη κάποια στιγμή θα χρειαστεί να "πετάξουμε" κάποιο block από την μνήμη και να φέρουμε κάποιο άλλο στην θέση του. Οι πολιτικές που μπορούμε να πετάξουμε ένα block από την μνήμη στο επίπεδο block που σας δίνεται είναι οι LRU (Least Recently Used) και MRU (Most Recently Used). Στην LRU "θυσιάζουμε" το λιγότερο πρόσφατα χρησιμοποιημένο block ενώ στην MRU το block που χρησιμοποιήσαμε πιο πρόσφατα.
Στη συνέχεια, περιγράφονται οι συναρτήσεις που αφορούν το επίπεδο από block, πάνω στο οποίο θα βασιστείτε για την υλοποίηση των συναρτήσεων που ζητούνται. Η υλοποίηση των συναρτήσεων αυτών θα δοθεί έτοιμη με τη μορφή βιβλιοθήκης.

Στο αρχείο κεφαλίδας bf.h που σας δίνεται ορίζονται οι πιο κάτω μεταβλητές:

BF_BLOCK_SIZE 512 */* Το μέγεθος ενός block σε bytes */*
BF_BUFFER_SIZE 100 */* Ο μέγιστος αριθμός block που κρατάμε στην μνήμη */*
BF_MAX_OPEN_FILES 100 */* Ο μέγιστος αριθμός ανοικτών αρχείων */*

και enumerations:
enum BF_ErrorCode *{ ... }*

Το BF_ErrorCode είναι ένα enumeration που ορίζει κάποιους κωδικούς λάθους που μπορεί να προκύψουν κατά την διάρκεια της εκτέλεσης των συναρτήσεων του επιπέδου BF.
enum ReplacementAlgorithm { LRU, MRU }

Το ReplacementAlgorithm είναι ένα enumeration που ορίζει τους κωδικούς για τους αλγορίθμους αντικατάστασης (*LRU* ή *MRU*).
Πιο κάτω υπάρχουν τα πρωτότυπα των συναρτήσεων που σχετίζονται με την δομή BF_Block.

typedef struct BF_Block BF_Block;

Το struct BF_Block είναι η βασική δομή που δίνει οντότητα στην έννοια του Block. Το BF_Block έχει τις πιο κάτω λειτουργίες.

void BF_Block_Init(*BF_Block **block /* δομή που προσδιορίζει το Block */)*

Η συνάρτηση *BF_Block_Init* αρχικοποιεί και δεσμεύει την κατάλληλη μνήμη για την δομή *BF_BLOCK*.

void BF_Block_Destroy(*BF_Block **block /* δομή που προσδιορίζει το Block */)*

Η συνάρτηση *BF_Block_Destroy* αποδεσμεύει την μνήμη που καταλαμβάνει η δομή *BF_BLOCK*.

void BF_Block_SetDirty(BF_Block *block /* *δομή που προσδιορίζει το Block* */***)**

Η συνάρτηση BF_Block_SetDirty αλλάζει την κατάσταση του block σε dirty. Αυτό πρακτικά σημαίνει ότι τα δεδομένα του block έχουν αλλαχτεί και το επίπεδο BF, όταν χρειαστεί θα γράψει το block ξανά στον δίσκο. Σε περίπτωση που απλώς διαβάζουμε τα δεδομένα χωρίς να τα αλλάζουμε τότε δεν χρειάζεται να καλέσουμε την συνάρτηση.

char* BF_Block_GetData(const BF_Block *block /* *δομή που προσδιορίζει το Block* */ **)**

Η συνάρτηση BF_Block_GetData επιστρέφει ένα δείκτη στα δεδομένα του Block. Άμα αλλάξουμε τα δεδομένα θα πρέπει να κάνουμε το block dirty με την κλήση της συνάρτησης BF_Block_SetDirty. Σε καμία περίπτωση δεν πρέπει να αποδεσμεύσετε την θέση μνήμης που δείχνει ο δείκτης.

<u>Πιο κάτω υπάρχουν τα πρωτότυπα των συναρτήσεων που σχετίζονται με το επίπεδο Block.</u>

BF_ErrorCode BF_Init*(const ReplacementAlgorithm repl_alg /* πολιτική αντικατάστασης */)*

Με τη συνάρτηση BF_Init πραγματοποιείται η αρχικοποίηση του επιπέδου BF. Μπορούμε να επιλέξουμε ανάμεσα σε δύο πολιτικές αντικατάστασης Block εκείνης της LRU και εκείνης της MRU.

BF_ErrorCode BF_CreateFile(*const char* filename* /* όνομα αρχείου */***)***

Η συνάρτηση BF_CreateFile δημιουργεί ένα αρχείο με όνομα filename το οποίο αποτελείται από blocks. Αν το αρχείο υπάρχει ήδη τότε επιστρέφεται κωδικός λάθους. Σε περίπτωση επιτυχούς εκτέλεσης της συνάρτησης επιστρέφεται BF_OK, ενώ σε περίπτωση αποτυχίας επιστρέφεται κωδικός λάθους. Αν θέλετε να δείτε το είδος του λάθους μπορείτε να καλέσετε τη συνάρτηση BF_PrintError.

BF_ErrorCode BF_OpenFile(
const char filename, /* όνομα αρχείου */*
int file_desc /* αναγνωριστικό αρχείου block */)*

Η συνάρτηση BF_OpenFile ανοίγει ένα υπάρχον αρχείο από blocks με όνομα filename και επιστρέφει το αναγνωριστικό του αρχείου στην μεταβλητή file_desc. Σε περίπτωση επιτυχίας επιστρέφεται BF_OK ενώ σε περίπτωση αποτυχίας, επιστρέφεται ένας κωδικός λάθους. Αν θέλετε να δείτε το είδος του λάθους μπορείτε να καλέσετε τη συνάρτηση BF_PrintError.

BF_ErrorCode BF_CloseFile(*int file_desc /* αναγνωριστικό αρχείου block */* ***)***

Η συνάρτηση BF_CloseFile κλείνει το ανοιχτό αρχείο με αναγνωριστικό αριθμό file_desc. Σε περίπτωση επιτυχίας επιστρέφεται BF_OK ενώ σε περίπτωση αποτυχίας, επιστρέφεται ένας κωδικός λάθους. Αν θέλετε να δείτε το είδος του λάθους μπορείτε να καλέσετε τη συνάρτηση BF_PrintError.

BF_ErrorCode BF_GetBlockCounter(
const int file_desc, / αναγνωριστικό αρχείου block */*
*int *blocks_num /* τιμή που επιστρέφεται */)*

Η συνάρτηση Get_BlockCounter δέχεται ως όρισμα τον αναγνωριστικό αριθμό file_desc ενός ανοιχτού αρχείου από block και βρίσκει τον αριθμό των διαθέσιμων blocks του, τον οποίο και επιστρέφει στην μεταβλητή blocks_num. Σε περίπτωση επιτυχίας επιστρέφεται BF_OK ενώ σε περίπτωση αποτυχίας, επιστρέφεται ένας κωδικός λάθους. Αν θέλετε να δείτε το είδος του λάθους μπορείτε να καλέσετε τη συνάρτηση BF_PrintError.

BF_ErrorCode BF_AllocateBlock(
const int file_desc, / αναγνωριστικό αρχείου block */*
*BF_Block *block /* το block που επιστρέφεται */)*

Με τη συνάρτηση BF_AllocateBlock δεσμεύεται ένα καινούριο block για το αρχείο με αναγνωριστικό αριθμό blockFile. Το νέο block δεσμεύεται πάντα στο τέλος του αρχείου, οπότε ο αριθμός του block είναι BF_getBlockCounter(...) - 1. Το block που δεσμεύεται καρφιτσώνεται στην μνήμη (pin) και επιστρέφεται στην μεταβλητή block. Όταν δεν χρειαζόμαστε άλλο αυτό το block τότε πρέπει να ενημερώσουμε το επίπεδο block καλώντας την συνάρτηση BF_UnpinBlock. Σε περίπτωση επιτυχίας της *BF_AllocateBlock* επιστρέφεται BF_OK, ενώ σε περίπτωση αποτυχίας επιστρέφεται ένας κωδικός λάθους. Αν θέλετε να δείτε το είδος του λάθους μπορείτε να καλέσετε τη συνάρτηση BF_PrintError.

BF_ErrorCode BF_GetBlock(
const int file_desc, */* αναγνωριστικό αρχείου block */*
const int block_num, */* αναγνωριστικός αριθμός block */*
BF_Block *block /* το block που επιστρέφεται */**)**

Η συνάρτηση BF_GetBlock βρίσκει το block με αριθμό block_num του ανοιχτού αρχείου file_desc και το επιστρέφει στην μεταβλητή block. Το block που δεσμεύεται καρφιτσώνεται στην μνήμη (pin). Όταν δεν χρειαζόμαστε άλλο αυτό το block τότε πρέπει να ενημερώσουμε τον επίπεδο block καλώντας την συνάρτηση BF_UnpinBlock. Σε περίπτωση επιτυχίας της *BF_GetBlock* επιστρέφεται BF_OK, ενώ σε περίπτωση αποτυχίας επιστρέφεται ένας κωδικός λάθους. Αν θέλετε να δείτε το είδος του λάθους μπορείτε να καλέσετε τη συνάρτηση BF_PrintError.

BF_ErrorCode BF_UnpinBlock(*BF_Block *block /* δομή block που γίνεται unpin */)*

Η συνάρτηση BF_UnpinBlock ξεκαρφιτσώνει το block από το επίπεδο BF το οποίο κάποια στιγμή θα το γράψει στο δίσκο. Σε περίπτωση επιτυχίας επιστρέφεται BF_OK, ενώ σε περίπτωση αποτυχίας επιστρέφεται ένας κωδικός λάθους. Αν θέλετε να δείτε το είδος του λάθους μπορείτε να καλέσετε τη συνάρτηση BF_PrintError.

void BF_PrintError(BF_ErrorCode err /* κωδικός λάθους */ **)**

Η συνάρτηση BF_PrintError βοηθά στην εκτύπωση των σφαλμάτων που δύναται να υπάρξουν με την κλήση συναρτήσεων του επιπέδου αρχείου block. Εκτυπώνεται στο stderr μια περιγραφή του σφάλματος.

void BF_Close()

Η συνάρτηση BF_Close κλείνει το επίπεδο Block γράφοντας στον δίσκο όποια block είχε στην μνήμη.

Συναρτήσεις HT (Extendible Hash Table)

Θα χρειαστεί μια μικρή τροποποίηση στην συναρτηση HT_InsertEntry που παραδώσατε στην εργασία 1 ώστε να επιστρέφει τη θέση της εγγραφής μέσα στο block καθώς και τις εγγραφές που άλλαξαν θέση σε περίπτωση split. Θα σας χρειαστούν για την ενημερωση του ευρετηρίου και για την δημιουργία της εγγραφής που θα εισαχθεί στο δευτερεύον ευρετήριο.

HT_ErrorCode HT_InsertEntry (

int indexDesc, / θέση στον πίνακα με τα ανοιχτά αρχεία */*

Record record, / δομή που προσδιορίζει την εγγραφή */*

int tupleId, /* Η θέση της εγγραφής στο πρωτεύον ευρετήριο */

UpdateRecordArray *updateArray /* πίνακας με τις αλλαγές που προέκυψαν από την εισαγωγη της νέας εγγραφής */

)

Η συνάρτηση HT_InsertEntry χρησιμοποιείται για την εισαγωγή μιας εγγραφής στο αρχείο κατακερματισμού. Οι πληροφορίες που αφορούν το αρχείο βρίσκονται στον πίνακα ανοιχτών αρχείων, ενώ η εγγραφή προς εισαγωγή προσδιορίζεται από τη δομή record. Στον πινακα updateArray επιστρέφεται η απαιτουμενη πληροφορία για την ενημερωση του δευτερευοντος ευρετηριου λογω των αλλαγων στο πρωτεύον πχ (surname city, oldTupleId, newTupleId). Στο tupleId επιστρέφεται η θέση εισαγωγής της εγγραφής στο πρωτεύον. Τη δομή του πίνακα UpdateRecordArray την ορίζεται εσείς. Σε περίπτωση που εκτελεστεί επιτυχώς επιστρέφεται HT_OK, ενώ σε διαφορετική περίπτωση κάποιος κωδικός λάθους.

Συναρτήσεις SHT (Secondary Hash Table)

Στη συνέχεια περιγράφονται οι συναρτήσεις που καλείστε να υλοποιήσετε στα πλαίσια της εργασίας αυτής οι οποίες αφορούν στη δημιουργία δευτερεύοντος ευρετηρίου επεκτατού κατακερματισμού, στο χαρακτηριστικό **surname** ή **city**.
Υποθέτουμε ότι υπάρχει ήδη το πρωτεύον ευρετήριο επεκτατού κατακερματισμού στο πεδίο id. Δεδομένου ότι υπάρχει το πρωτεύον ευρετήριο θα σας είναι χρήσιμο στα μεταδεδομένα του δευτερεύοντος ευρετηρίου (πίνακα αρχείων) να κρατάτε την πληροφορία για το ποιο είναι το αντίστοιχο πρωτεύον ευρετήριο.
Κάθε συνάρτηση του αρχείο επεκτατού κατακερματισμού επιστρέφει ένα κωδικό λάθους που ορίζεται από το πιο κάτω enumeration.

```
enum HT_ErrorCode {
        HT_OK,
        HT_ERROR
}
```

Στη συνέχεια περιγράφονται οι συναρτήσεις του αρχείου επεκτατού κατακερματισμού

HT_ErrorCode ***SHT_Init()***

Η συνάρτηση SHT_Init χρησιμοποιείται για την πιθανή αρχικοποίηση κάποιων δομών που μπορεί να χρειαστείτε. Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφεται HT_OK, ενώ σε διαφορετική σε περίπτωση κωδικός λάθους.

HT_ErrorCode SHT_CreateSecondaryIndex(
*const char *sfileName, /* όνομα αρχείου */*
*char *attrName, /* όνομα πεδίου-κλειδιού */*
int attrLength, / μήκος πεδίου-κλειδιού */*
int depth, / το ολικό βάθος ευρετηρίου επεκτατού κατακερματισμού */*
*char *fileName /* όνομα αρχείου πρωτεύοντος ευρετηρίου*/)*

Η συνάρτηση SHT_CreateSecondaryIndex χρησιμοποιείται για τη δημιουργία και κατάλληλη αρχικοποίηση ενός αρχείου δευτερεύοντος κατακερματισμού με όνομα sfileName για το αρχείο πρωτεύοντος κατακερματισμού fileName.
Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφεται HT_OK, ενώ σε διαφορετική περίπτωση κωδικός λάθους.

HT_ErrorCode SHT_OpenSecondaryIndex(
*const char *sfileName, /* όνομα αρχείου */*
*int *indexDesc /* θέση στον πίνακα με τα ανοιχτά αρχεία που επιστρέφεται */)*

Η ρουτίνα αυτή ανοίγει το αρχείο με όνομα sfileName. Εάν το αρχείο ανοιχτεί κανονικά, η ρουτίνα επιστρέφει *HT_OK*. Σε διαφορετική περίπτωση, επιστρέφει *HT_ERROR*.

Θα πρέπει να κρατάτε στην μνήμη έναν πίνακα για όλα τα ανοιχτά αρχεία. Ο ακέραιος που επιστρέφεται στην μεταβλητή *indexDesc* είναι η θέση του πίνακα που αντιστοιχεί στο αρχείο που μόλις ανοίχτηκε. Σ' αυτόν τον πίνακα θα κρατάτε οτιδήποτε σχετικό κρίνετε ότι πρέπει να είναι άμεσα διαθέσιμο για κάθε ανοιχτό αρχείο (π.χ., γενικές πληροφορίες για το αρχείο, το αναγνωριστικό του αρχείου έτσι όπως έχει ανοιχτεί από το λειτουργικό σύστημα, κτλ. - χωρίς τα παραπάνω να θεωρούνται απαραίτητα). Το ίδιο αρχείο μπορεί να ανοιχτεί πολλές φορές και για κάθε άνοιγμα καταλαμβάνει διαφορετική θέση στον πίνακα που κρατάτε στη μνήμη. Μπορείτε να υποθέσετε ότι οποιαδήποτε στιγμή δεν θα υπάρχουν περισσότερα από MAXOPENFILES = 20 ανοιχτά αρχεία.

HT_ErrorCode SHT_CloseSecondaryIndex(*int indexDesc /* θέση στον πίνακα με τα ανοιχτά αρχεία */****)***

Η ρουτίνα αυτή κλείνει το αρχείο δευτερεύοντος ευρετηρίου επεκτατού κατακερματισμού του οποίου οι πληροφορίες βρίσκονται στην θέση indexDesc του πίνακα ανοιχτών αρχείων. Επίσης σβήνει την καταχώρηση που αντιστοιχεί στο αρχείο αυτό στον πίνακα ανοιχτών αρχείων. Η συνάρτηση επιστρέφει *EH_OK* εάν το αρχείο κλείσει επιτυχώς, ενώ σε διαφορετική σε περίπτωση κωδικός λάθους.

HT_ErrorCode SHT_SecondaryInsertEntry (
int indexDesc, / θέση στον πίνακα με τα ανοιχτά αρχεία */*
SecondaryRecord record / δομή που προσδιορίζει την εγγραφή */****)***

Η συνάρτηση SHT_SecondaryInsertEntry χρησιμοποιείται για την εισαγωγή μιας εγγραφής στο αρχείο κατακερματισμού. Οι πληροφορίες που αφορούν το αρχείο βρίσκονται στον πίνακα ανοιχτών αρχείων, ενώ η εγγραφή προς εισαγωγή προσδιορίζεται από τα index_key και tupleId.
Σε περίπτωση που εκτελεστεί επιτυχώς επιστρέφεται HT_OK, ενώ σε διαφορετική περίπτωση κάποιος κωδικός λάθους.

Η δομή SecondaryRecord η οποία χρησιμοποιείται από τη SHT_InsertEntry είναι η ακόλουθη:

```
typedef struct{
        char index_key[20];
        int tupleId;  /*Ακέραιος που προσδιορίζει το block και τη θέση μέσα στο block στην
        οποία     έγινε η εισαγωγή της εγγραφής στο πρωτεύον ευρετήριο.*/
}SecondaryRecord;
```

Το **tupleId** υπολογίζεται από τον αριθμό του block τον αριθμό των εγγραφών που χωράει το block και τη θέση της εγγραφής μέσα στο εν λόγω block, δηλαδή:

tupleId= (blockId+1) *num_of_rec_in_block)+index_of_rec_in_block

HT_ErrorCode SHT_SecondaryUpdateEntry (
int indexDesc, / θέση στον πίνακα με τα ανοιχτά αρχεία */*
***UpdateRecordArray *updateArray** /* δομή που προσδιορίζει την παλιά εγγραφή */)*

Η συνάρτηση SHT_SecondaryUpdateEntry χρησιμοποιείται για την ενημέρωση των δεικτών των εγγραφών του δευτερεύοντος ευρετηρίου σε περίπτωση split στο πρωτεύον ευρετήριο. Τη δομή του UpdateRecordArray την ορίζετε εσείς (δείτε και ***HT_InsertEntry*** *)*

HT_ErrorCode SHT_PrintAllEntries(
int sindexDesc, / θέση στον πίνακα με τα ανοιχτά αρχεία του αρχείου δευτερεύοντος ευρετηρίου */*
*char *index_key /* τιμή του πεδίου-κλειδιού προς αναζήτηση */)*

Η συνάρτηση SHT_PrintAllEntries χρησιμοποιείται για την εκτύπωση όλων των εγγραφών που υπάρχουν στο αρχείο κατακερματισμού οι οποίες έχουν τιμή στο πεδίο-κλειδί του δευτερεύοντος ευρετηρίου ίση με index_key. Η δομή δίνει πληροφορία για το αρχείο κατακερματισμού, όπως αυτή έχει επιστραφεί από την SHT_OpenIndex. Το πρωτεύον ευρετήριο θα το βρείτε από τα μεταδεδομένα που κρατάτε για το δευτερεύον ευρετήριο. Για κάθε εγγραφή που υπάρχει στο αρχείο και έχει τιμή στο πεδίο-κλειδί ίση με value, εκτυπώνονται τα περιεχόμενά της (συμπεριλαμβανομένου και του πεδίου-κλειδιού). Σε περίπτωση που εκτελεστεί επιτυχώς επιστρέφεται HT_OK, ενώ σε διαφορετική περίπτωση κάποιος κωδικός λάθους.

HT_ErrorCode SHT_HashStatistics(*char *filename /* όνομα του αρχείου που ενδιαφέρει */)*

Η συνάρτηση διαβάζει το αρχείο με όνομα filename και τυπώνει στατιστικά στοιχεία: α) Το πόσα blocks έχει ένα αρχείο, β) Το ελάχιστο, το μέσο και το μέγιστο πλήθος εγγραφών που έχει κάθε bucket ενός αρχείου. Σε περίπτωση που εκτελεστεί επιτυχώς επιστρέφεται HT_OK, ενώ σε διαφορετική περίπτωση κάποιος κωδικός λάθους.

Ζεύξη

Η ζεύξη γίνεται στο κοινό πεδίο επιλογής των δύο αρχείων. Για το πεδίο πρέπει να υπάρχει δευτερεύον ευρετήριο. Η ζεύξη γίνεται πάνω στο πεδίο που έχουν οριστεί τα δευτερεύοντα ευρετήρια. Η ζεύξη περιορίζεται στο block. (θεωρούμε ότι όλες οι εγγραφές με κοινο πεδίο κλειδί χωρούν σε ένα block) και δε μοιράζονται σε περισσότερα του ενός. Τα δύο αρχεία περιέχουν εγγραφές ίδιου τύπου. Θα εκτυπώνει σε μία γραμμή τη ζεύξη των δύο γραμμών στο πεδίο κλειδί της ζεύξης. Αν το index_key είναι NULL επιστρέφει όλες τις εγγραφές ζεύξης.

HT_ErrorCode SHT_InnerJoin(

int sindexDesc1, / θέση στον πίνακα με τα ανοιχτά αρχεία του αρχείου δευτερεύοντος ευρετηρίου για το πρώτο αρχείο εισόδου */*

int sindexDesc2, / θέση στον πίνακα με τα ανοιχτά αρχεία του αρχείου δευτερεύοντος ευρετηρίου για το δεύτερο αρχείο εισόδου*

*char *index_key /* Το κλειδί που θα γίνει η ζεύξη*/)*

Έλεγχος ορθότητας του προγράμματός σας

Η main που σας δίνεται σε αυτή την εργασία είναι ενδεικτική. Καλείστε να δημιουργήσετε δικές σας συναρτήσεις/mains που θα αποδεικνύουν την ορθή λειτουργία του προγράμματός σας.

Συνάρτηση main Ελέγχου Λειτουργικότητας

Για να ελέγξετε την ορθότητα της υλοποίησης σας πρέπει να κατασκευάσετε και να παραδώσετε τουλάχιστον μια δική σας συνάρτηση main ελέγχου η οποία θα διεκπεραιώνει τις παρακάτω διαδικασίες:

1. Δημιουργία αρχείου επεκτατού κατακερματισμού (HT)
2. Δημιουργία δευτερεύοντος ευρετηρίου στο πεδίο surname (SHT)
3. Εισαγωγή ενδεικτικών εγγραφών στο HT και το SHT από test αρχείο με πλήθος εγγραφών.
4. Αναζήτηση υποσυνόλου εγγραφών με κάποιο κριτήριο ως προς το πεδίο-κλειδί του πρωτεύοντος
5. Αναζήτηση εγγραφών τα οποία πληρούν κάποιο κριτήριο ως προς το πεδίο στο οποίο έγινε η ευρετηρίαση στο δευτερεύον ευρετήριο.
6. Εισαγωγή νέας εγγραφής στο πρωτεύον και δευτερεύων.
7. Αναζήτηση πάλι (βήματα 4 και 5) με τα στοιχεία της νέας εγγραφής.
8. Εκτύπωση των στατιστικών

Σημείωση:

1. Εισαγωγή ζεύγους (τιμής κλειδιού, δείκτη εγγραφής) στο δευτερεύον ευρετήριο (SHT) μπορεί να γίνει μόνο εφόσον έχει ολοκληρωθεί η αντίστοιχη διαδικασία εισαγωγής εγγραφής στο πρωτεύον (HT).

2. Θα πρέπει η υλοποίηση σας να είναι συμβατή με τις main συναρτήσεις οι οποίες θα δοθούν, δηλαδή θα πρέπει να ακολουθήσετε πιστά τα πρότυπα συναρτήσεων τα οποία σας δίνονται.

Αρχεία που σας δίνονται

Στα αρχεία που σας δίνονται θα βρείτε δύο φακέλους που περιέχουν το project που θα πρέπει να επιστρέψετε. Το project έχει την πιο κάτω δομή:

- **bin**: Ο κώδικας των εκτελέσιμων που δημιουργούνται
- **build**: Περιέχει όλα τα object files που δημιουργούνται κατά την μεταγλώττιση.
- **include**: Περιέχει όλα τα αρχεία κεφαλίδας που θα έχει το project σας. Θα βρείτε το αρχείο bf.h sht_file.h και hash_file.h.

- **lib**: Περιέχει όλες τις βιβλιοθήκες που θα χρειαστείτε για το project σας. Θα βρείτε το αρχείο libbf.so που είναι η βιβλιοθήκη για να καλείτε το επίπεδο BF.
- **src**: Τα αρχεία κώδικα (.c) τις εφαρμογής σας.
- **examples**: Σε αυτό τον φάκελο θα βρείτε ένα αρχείο main (ht_main.c) το οποίο περιέχει κώδικα που τρέχει τις συναρτήσεις που πρέπει να υλοποιήσετε.

Επίσης σας δίνετε και ένα αρχείο Makefile για να κάνετε αυτόματα compile των κωδικά σας.

Πράγματα που πρέπει να προσέξετε

- Η ζεύξη θα γίνεται εντός των εγγραφών του ίδιου block.

Παράδοση εργασίας

Η εργασία παραδίδεται από τις ομάδες που έχουν σχηματιστεί κατά την 1η εργασία. Θα παραδώσετε τη βιβλιοθήκη SHT και την HT. Θα αλλάξετε την main σας ώστε να κάνει εισαγωγή των εγγραφών τόσο σε πρωτεύον όσο και δευτερεύον ευρετήριο. Αν θέλετε εκτός από την αναγκαία αλλαγή στην HT_InsertEntry μπορείτε να βελτιώσετε καποια συνάρτηση που δεν είχατε υλοποιήσει σωστά στην 1η εργασία σε περίπτωση που σας δημιουργεί πρόβλημα στη δεύτερη εργασία.
Σημειώνω ότι οι αλλαγές δε θα ληφθούν υπόψη για την βαθμολόγηση της πρώτης εργασίας.
Προθεσμία παράδοσης: 16/01/2022
Γλώσσα υλοποίησης: C / C++ χωρίς χρήση βιβλιοθηκών που υπάρχουν μόνο στην C++.
Περιβάλλον υλοποίησης: Linux (gcc 5.4+).
Περιβάλλον εξέτασης: Η εργασία θα εξεταστεί στα linux του di.uoa και καλείστε να έχετε ελέγξει ότι εκτελείται ορθώς στο συγκεκριμένο περιβάλλον.

Παραδοτέα

Όλως ο φάκελος hashfile μαζί με αρχείο Makefile που θα κάνει link τον κώδικά σας με την ήδη τροποποιημένη main αλλά και με τις υπόλοιπες που θα υλοποιήσετε. Ένα README που θα εξηγεί τις σχεδιαστικές επιλογές ή τυχόν παραδοχές που έχετε κάνει. Επίσης θα αναφέρεται τυχόν δυσλειτουργίες του προγράμματος σας που έχετε παρατηρήσει. Για κάθε ομάδα παραδίδει μόνο 1 άτομο και βάζει στο README τα ον/νυμο και τους ΑΜ των υπόλοιπων μελών.