Πληροφορική & Τηλεπικοινωνίες

Κ18 - Υλοποίηση Συστημάτων Βάσεων Δεδομένων

Χειμερινό Εξάμηνο 2019 – 2020

Καθηγητής Ι. Ιωαννίδης
Άσκηση 1 – Παράδοση 28/10/2019

Σκοπός της εργασίας αυτής είναι η κατανόηση της εσωτερικής λειτουργίας των Συστημάτων Βάσεων Δεδομένων όσον αφορά τη διαχείριση σε επίπεδο μπλοκ (block) αλλά και ως προς τη διαχείριση σε επίπεδο εγγραφών. Πιο συγκεκριμένα, στα πλαίσια της 1η εργασίας θα υλοποιήσετε ένα σύνολο συναρτήσεων που διαχειρίζονται αρχεία σωρού (Heap Files).

Οι συναρτήσεις που καλείστε να υλοποιήσετε αφορούν τη διαχείριση εγγραφών. 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.

Η διαχείριση των αρχείων σωρού γίνεται μέσω των συναρτήσεων με το πρόθεμα HP_. Τα αρχεία έχουν μέσα εγγραφές τις οποίες διαχειρίζεστε μέσω των κατάλληλων συναρτήσεων. Οι εγγραφές έχουν τη μορφή που δίνεται στη συνέχεια.

```
typedef struct{
        int id,
        char name[15],
        char surname[20],
        char city[10];
}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 όντως για αρχείο σωρού, μπορεί να περιλαμβάνει πληροφορίες για τη μορφή των δεδομένων κλπ.

Στη συνέχεια δίνεται ένα παράδειγμα των εγγραφών που θα προστίθενται στα αρχεία που θα δημιουργείται με την υλοποίησή σας. Οι εγγραφές αυτές δημιουργούνται από το πρόγραμμα hp_main.c που σας δίνεται και καλεί τις συναρτήσεις που καλείστε να υλοποιήσετε.

15, “Marianna”, “Rezkalla”, “Hong Kong”
4, “Christoforos”, “Svingos”, “Athens”
300, “Sofia”, “Karvounari”, “San Francisco”

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.

Πιο κάτω υπάρχουν τα πρωτότυπα των συναρτήσεων που σχετίζονται με το επίπεδο Block.

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 HP (Heap File)

Στη συνέχεια περιγράφονται οι συναρτήσεις που καλείστε να υλοποιήσετε στα πλαίσια της εργασίας αυτής και που αφορούν το αρχείο σωρού.

Κάθε συνάρτηση του αρχείο σωρού επιστρέφει ένα κωδικό λάθους που ορίζεται από το πιο κάτω enumeration.

```
enum HP_ErrorCode {
        HP_OK,
        HP_ERROR
}
```

Πιο κάτω περιγράφονται οι συναρτήσεις του αρχείου σωρού

HP_ErrorCode HP_Init()

Η συνάρτηση HP_Init χρησιμοποιείται για την αρχικοποίηση κάποιον δομών που μπορεί να χρειαστείτε. Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφεται HP_OK, ενώ σε διαφορετική περίπτωση HP_ERROR.

HP_ErrorCode HP_CreateFile(*const char *fileName /* όνομα αρχείου */* ***)***

Η συνάρτηση HP_CreateFile χρησιμοποιείται για τη δημιουργία και κατάλληλη αρχικοποίηση ενός άδειου αρχείου σωρού με όνομα fileName. Στην περίπτωση που το αρχείο υπάρχει ήδη, τότε επιστρέφεται ένας κωδικός λάθους. Σε περίπτωση που εκτελεστεί επιτυχώς επιστρέφεται HP_OK, ενώ σε διαφορετική περίπτωση κωδικός λάθους.

HP_ErrorCode HP_OpenFile(
*const char *fileName, /* όνομα αρχείου */*
*int *fileDesc /* αναγνωριστικός αριθμός ανοίγματος αρχείου που επιστρέφεται */* ***)***

Η συνάρτηση HP_OpenFile ανοίγει το αρχείο με όνομα filename και διαβάζει από το πρώτο μπλοκ την πληροφορία που αφορά το αρχείο σωρού. Επιστρέφει στην μεταβλητή *fileDesc* τον αναγνωριστικό αριθμό ανοίγματος αρχείου, όπως αυτός επιστράφηκε από το επίπεδο διαχείρισης μπλοκ. Σε περίπτωση που εκτελεστεί επιτυχώς επιστρέφεται HP_OK, ενώ σε διαφορετική περίπτωση κάποιος κωδικός λάθους. Αν το αρχείο που ανοίχτηκε δεν πρόκειται για αρχείο σωρού, τότε αυτό θεωρείται επίσης περίπτωση σφάλματος.

HP_ErrorCode HP_CloseFile(*int fileDesc /* αναγνωριστικός αριθμός ανοίγματος αρχείου */* ***)***

Η συνάρτηση HP_CloseFile κλείνει το αρχείο που προσδιορίζεται από τον αναγνωριστικό αριθμό ανοίγματος fileDesc. Σε περίπτωση που εκτελεστεί επιτυχώς επιστρέφεται HP_OK, ενώ σε διαφορετική περίπτωση κάποιος κωδικός λάθους.

HP_ErrorCode HP_InsertEntry (

int fileDesc, / αναγνωριστικός αριθμός ανοίγματος αρχείου */*

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

Η συνάρτηση HP_InsertEntry χρησιμοποιείται για την εισαγωγή μίας εγγραφής στο αρχείο σωρού. Ο αναγνωριστικός αριθμός ανοίγματος του αρχείου δίνεται με την fileDesc ενώ η εγγραφή προς εισαγωγή προσδιορίζεται από τη δομή record. Η εγγραφή προστίθεται στο τέλος του αρχείου, μετά την τρέχουσα τελευταία εγγραφή. Σε περίπτωση που εκτελεστεί επιτυχώς επιστρέφεται HP_OK, ενώ σε διαφορετική περίπτωση κάποιος κωδικός λάθους.

HP_ErrorCode HP_PrintAllEntries(

int fileDesc, /* αναγνωριστικός αριθμός ανοίγματος αρχείου */

char *fieldName, /* το όνομα του πεδίου στο οποίο θα γίνει η αναζήτηση */

void *value, /* η τιμή η οποία αναζητούμε*/**)**

Η συνάρτηση HP_PrintAllEntries χρησιμοποιείται για την εκτύπωση όλων των εγγραφών που το πεδίο με όνομα fieldName έχει τιμή value. Αν το value είναι NULL τότε θα εκτυπώνει όλες τις εγγραφές του αρχείου σωρού. Το fileDesc είναι ο αναγνωριστικός αριθμός ανοίγματος του αρχείου, όπως αυτός έχει επιστραφεί από το επίπεδο διαχείρισης μπλοκ. Σε περίπτωση που εκτελεστεί επιτυχώς επιστρέφεται HP_OK, ενώ σε διαφορετική περίπτωση κάποιος κωδικός λάθους.

HP_ErrorCode HP_GetEntry(

int fileDesc, /* αναγνωριστικός αριθμός ανοίγματος αρχείου */

int rowId, /* θέση εγγραφής στο αρχείο σωρού */

Record *record */* *δομή στην οποία επιστρέφεται η εγγραφή* */* **)**

Η συνάρτηση HP_GetEntry χρησιμοποιείται για την επιστροφή στην δομή record της εγγραφής εκείνης που βρίσκεται στην rowId θέση στο αρχείο σωρού. Σε περίπτωση που εκτελεστεί επιτυχώς επιστρέφεται HP_OK, ενώ σε διαφορετική περίπτωση κάποιος κωδικός λάθους.

Ερώτηση Bonus (5 Μόρια)

Η συνάρτηση TestFileScan που υπάρχει στην hp_main.c προσομοιώνει μια λειτουργία που εμφανίζεται πολύ συχνά στις βάσεις δεδομένων εκείνη της επανειλημμένης σάρωσης των δεδομένων. Χρησιμοποιήστε την εντολή strace -c η οποία παρακολουθεί την κλήση των read και write system calls του προγράμματός σας. Η hp_main.c που σας δίνεται κάνει χρήση του LRU αλγορίθμου αντικατάστασης. Παρακολουθήστε με την βοήθεια της strace τον αριθμό των read στον δίσκο. Κάντε το ίδιο χρησιμοποιώντας ως αλγόριθμο αντικατάστασης των MRU (σχολιάζοντας την γραμμή 84 και αποσχολιάζοντας την γραμμή 85).

Παρατηρείται κάποια διαφορά στην κλήση των read; Αν ναι, που νομίζετε ότι οφείλεται αυτό;

Αρχεία που σας δίνονται

Στα αρχεία που σας δίνονται θα βρείτε δύο φακέλους που περιέχουν το project που θα πρέπει να επιστρέψετε. Ο ένας περιέχει την βιβλιοθήκη η οποία 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 και heap_file.h.
- **lib**: Περιέχει όλες τις βιβλιοθήκες που θα χρειαστείτε για το project σας. Θα βρείτε το αρχείο libbf.so που είναι η βλιοθήκη για να καλείτε το επίπεδο BF.
- **src**: Τα αρχεία κώδικα (.c) τις εφαρμογής σας.
- **examples**: Σε αυτό τον φάκελο θα βρείτε ένα αρχείο main (bf_main.c) που χρησιμοποιεί το επίπεδο BF καθώς και ένα αρχείο hp_main.c το οποίο χρησιμοποιεί το επίπεδο HP.

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

Πράγματα που πρέπει να προσέξετε

- Θα πρέπει να καλείτε την συνάρτηση *BF_UnpinBlock* για κάθε BF_Block που δεν χρειάζεστε πλέον. Αν δεν τα κάνετε Unpin τότε ο buffer του επιπέδου BF γεμίζει και δεν μπορεί να “θυσιάσει” κάποιο Block γιατί όλα θα είναι ενεργά.
- Πάντα να καλείτε την συνάρτηση BF_Block_SetDirty για τα Block που αλλάζετε τα δεδομένα τους. Αν δεν τα κάνετε dirt τότε δεν γράφονται στον δίσκο.
- Ο κώδικάς που σας δίνεται στο ./examples/hp_main.c δεν θα πρέπει να αλλαχθεί!

Χρήσιμα Links

https://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html
https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_Recently_Used_.28LRU.29
https://en.wikipedia.org/wiki/Cache_replacement_policies#Most_Recently_Used_.28MRU.29
https://linux.die.net/man/1/strace

Παράδοση εργασίας

Η εργασία αυτή είναι ατομική.

Προθεσμία παράδοσης: 28/10/2019

Γλώσσα υλοποίησης: C / C++ χωρίς χρήση βιβλιοθηκών που υπάρχουν μόνο στην C++.

Περιβάλλον υλοποίησης: Linux (gcc 5.4+).

Παραδοτέα: Όλως ο φάκελος heapfile μαζί με αρχείο Makefile που θα κάνει link τον κώδικά σας με την ήδη υπάρχουσα main στο αρχείο hp_main.c. Ένα README που θα εξηγεί συνοπτικά τις σχεδιαστικές επιλογές ή τυχόν παραδοχές που έχετε κάνει και στο οποίο θα αιτιολογείτε το bonus ερώτημα, αν αυτό έχει γίνει.