Πληροφορική & Τηλεπικοινωνίες

Υλοποίηση Συστημάτων Βάσεων Δεδομένων Χειμερινό Εξάμηνο 2024 – 2025

Διδάσκων Θ. Μαΐλης

Άσκηση 1 - Προθεσμία: 25/10/2023

Σκοπός της Εργασίας

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. Επίσης, μέσω της εργασίας και των επόμενων εργασιών θα γίνει αντιληπτό το κατά πόσο μπορεί να βελτιώσει την απόδοση ενός Συστήματος Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ) η ύπαρξη ευρετηρίων πάνω στις εγγραφές. Πιο συγκεκριμένα, στα πλαίσια της εργασίας θα υλοποιήσετε ένα σύνολο συναρτήσεων που διαχειρίζονται αρχεία που δημιουργήθηκαν βάσει οργάνωσης αρχείου 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** (heap files) γίνεται μέσω των συναρτήσεων με το πρόθεμα HP_. Τα αρχεία σωρού 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[20], 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 σωρού ή αρχείο κατακερματισμού, σε πληροφορία που αφορά το πεδίο κατακερματισμού για τα αντίστοιχα αρχεία κ.λπ. Στη συνέχεια δίνεται ένα παράδειγμα των εγγραφών που θα προστίθενται στα αρχεία που θα δημιουργείται με την υλοποίησή σας. Εγγραφές με τις οποίες μπορείτε να ελέγξετε το πρόγραμμά σας θα δοθούν υπό μορφή αρχείων κειμένου μαζί με κατάλληλες συναρτήσεις main στο eclass του μαθήματος.

```
{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.

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.

Άσκηση 1

Συναρτήσεις HP (σωρού (heap file))

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

```
int HP_CreateFile(char *fileName     /*όνομα αρχείου*/);
```

Η συνάρτηση HP_CreateFile χρησιμοποιείται για τη δημιουργία και κατάλληλη αρχικοποίηση ενός άδειου αρχείου σωρού με όνομα fileName. Στα πλαίσια της δημιουργίας, αποθηκεύεται στο πρώτο block του αρχείου σωρού ένα στιγμιότυπο του HP_Info που κρατάει τα μεταδεδομένα της δομής. Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφεται 0, ενώ σε διαφορετική περίπτωση -1.

```
HP_info* HP_OpenFile(char *fileName,   /* όνομα αρχείου */
    int *file_desc
    /* προσδιοριστικό ανοίγματος αρχείου όπως δίνεται από την BF_OpenFile */);
```

Η συνάρτηση HP_OpenFile ανοίγει το αρχείο με όνομα filename και διαβάζει από το πρώτο μπλοκ την πληροφορία που αφορά το αρχείο σωρού. Κατόπιν, ενημερώνεται μια δομή που κρατάει όσες πληροφορίες κρίνονται αναγκαίες για το αρχείο αυτό προκειμένου να είναι δυνατή η επεξεργασία των εγγραφών του. Η μεταβλητή *file_desc αναφέρεται στο αναγνωριστικό ανοίγματος του συγκεκριμένου αρχείου όπως προκύπτει από την BF_OpenFile.

```
typedef struct {
…
} HP_info;
```

Η δομή **HP_Info** μπορεί να κρατάει πληροφορίες όπως: το id του τελευταίου block του αρχείου σωρού, τον αριθμό των εγγραφών που χωράνε σε κάθε block του αρχείου σωρού.

```
typedef struct {
    …
} HP_block_info;
```

Επιπλέον στο τέλος κάθε block μπορεί ναυπάρχει μία δομή στην οποία θα αποθηκεύετε πληροφορίες σε σχέση με το block όπως: τον αριθμό των εγγραφών στο συγκεκριμένο block, έναν δείκτη στο επόμενο block δεδομένων.

```
int HP_CloseFile(int file_desc,
    /* προσδιοριστικό ανοίγματος αρχείου όπως δίνεται από την BF_OpenFile */
    HP_info* header_info   /* η επικεφαλίδα του αρχείου */);
```

Η συνάρτηση HP_CloseFile κλείνει το αρχείο που προσδιορίζεται από το αναγνωριστικό file_desc. Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφεται 0, ενώ σε διαφορετική περίπτωση -1. Η συνάρτηση είναι υπεύθυνη και για την αποδέσμευση της μνήμης που καταλαμβάνει η δομή που περάστηκε ως παράμετρος, στην περίπτωση που το κλείσιμο πραγματοποιήθηκε επιτυχώς.

```
int HP_InsertEntry( int file_desc,
    /* προσδιοριστικό ανοίγματος αρχείου όπως δίνεται από την BF_OpenFile */
    HP_info* header_info,   /* επικεφαλίδα του αρχείου*/
    Record record           /* δομή που προσδιορίζει την εγγραφή */
);
```

Η συνάρτηση HP_InsertEntry χρησιμοποιείται για την εισαγωγή μιας εγγραφής στο αρχείο σωρού. Το αναγνωριστικό για το αρχείο είναι file_desc, τα μεταδεδομένα του αρχείου βρίσκονται στη δομή header_info, ενώ η εγγραφή προς εισαγωγή προσδιορίζεται από τη δομή record. Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφετε τον αριθμό του block στο οποίο έγινε η εισαγωγή (blockId) , ενώ σε διαφορετική περίπτωση -1.

```
int HP_GetAllEntries(int file_desc,
    /* προσδιοριστικό ανοίγματος αρχείου όπως δίνεται από την BF_OpenFile */
    int id /* η τιμή id της εγγραφής στην οποία πραγματοποιείται η αναζήτηση*/
);
```

Η συνάρτηση αυτή χρησιμοποιείται για την εκτύπωση όλων των εγγραφών που υπάρχουν στο αρχείο σωρού οι οποίες έχουν τιμή στο πεδίο-κλειδί ίση με value. Έχει σαν είσοδο το προσδιοριστικό ανοίγματος του αρχείου

σωρού, έναν δείκτη στα μεταδεδομένα του. Για κάθε εγγραφή που υπάρχει στο αρχείο και έχει τιμή στο πεδίο id ίση με value, εκτυπώνονται τα περιεχόμενά της (συμπεριλαμβανομένου και του πεδίου-κλειδιού). Να επιστρέφεται επίσης το πλήθος των blocks που διαβάστηκαν μέχρι να βρεθούν όλες οι εγγραφές. Σε περίπτωση επιτυχίας επιστρέφει το πλήθος των blocks που διαβάστηκαν, ενώ σε περίπτωση λάθους επιστρέφει -1.

Σημειώσεις 1ης εργασίας

- Το **πρώτο block** των αρχείων σωρού κρατάει τα μεταδεδομένα τα οποία σχετίζονται με την δομή (HP_info). Σε περίπτωση που έχουμε κάποια πράξη όπως η εισαγωγή, τότε θα πρέπει να ανανεώνονται οι τιμές των HP_info και οι αλλαγές αυτές θα πρέπει να γραφτούν κάποια στιγμή στον δίσκο. Ιδανικά θέλουμε το block αυτό να παραμένει στην ενδιάμεση μνήμη και να γίνει unpin με το κλείσιμο του αρχείου (αφού έχει γίνει dirty εφόσον περιέχει αλλαγές).
- Αποφύγετε την χρήση των συναρτήσεων για allocate στην μνήμη. Αντί αυτού, να έχετε δείκτες σε συγκεκριμένες περιοχές της ενδιάμεση μνήμη. Στο παράδειγμα bf_main δείχνουμε πως μπορεί να γίνει αυτό.
- Για την ολοκλήρωση της άσκησης θα χρειαστεί να χρησιμοποιήσετε την συνάρτηση **memcpy** για την αντιγραφή πληροφορίας από ένα Record στην ενδιάμεση μνήμη.
- Για το αρχείο σωρού θεωρείστε ότι **δεν έχουμε πρωτεύον κλειδί**, οπότε μην κάνετε κάποιον έλεγχο.
- Θα πρέπει να ακολουθήσετε πιστά τα πρότυπα συναρτήσεων τα οποία σας δίνονται.

Παράδοση εργασίας

- Η εργασία είναι ομαδική **2 ή 3 ατόμων.**
- Γλώσσα υλοποίησης: **C**

Παραδοτέα

- Για την πρώτη άσκηση περιμένουμε να παραδώσετε το αρχείο πηγαίου κώδικα hp_file.c
- Θέλουμε το αρχείο αυτό να ανταποκρίνεται στην βιβλιοθήκη hp_file.h που σας δίνουμε και να μην έχει γίνει κάποια αλλαγή στην αντίστοιχη βιβλιοθήκη.
- Όλη η λειτουργικότητα του κώδικά σας θα πρέπει να βρίσκεται στο αρχείο hp_file.c.

Βαθμολόγηση

Η βαθμολόγηση της εργασίας θα γίνει με γνώμονα την **ορθότητα**, την **πληρότητα** των αποτελεσμάτων, καθώς και την **ελαχιστοποίηση των διαβασμάτων και γραψιμάτων** στον σκληρό δίσκο. Από τη στιγμή που το αρχείο σωρού αντιστοιχεί σε πίνακα χωρίς κάποιο πρωτεύον κλειδί, θεωρήστε ότι μπορεί να έχουμε περισσότερες από μία εγγραφές με το ίδιο ID, επομένως θα πρέπει να επιστρέφονται όλες από την GetAllEntries. Ο κώδικάς σας θα πρέπει να εξεταστεί με διαφορετικούς αριθμούς εισαγωγών, ενώ θα πρέπει να διασφαλίζετε ότι κάθε φορά που κλείνετε και ανοίγετε το αρχείο σωρού, μπορείτε να το ξανανοίξετε χωρίς να υπάρχει απώλεια πληροφορία.