Εργασία 1 - Υλοποίηση Συστημάτων Βάσεων Δεδομένων - Σωρός - Πρωτεύον ευρετήριο

Παναγιώτης Δρίβας, Α.Μ.: 1115201900055

Παναγιώτα Γύφτου , Α.Μ.: 1115201900318

Δεκέμβριος 2022

Θέμα εργασίας

Υλοποίηση σημαντικών συναρτήσεων που αφορούν τον τύπο οργάνωσης αρχείο σωρού (*heapfile*) και τον τύπο οργάνωσης στατικό κατακερματισμένο αρχείο (*hashfile*).

Παρατήρηση: Το πλήθος των εγγραφών που θα εισαχθούν στο αρχείο είναι μεγέθους 300 *#define RECORDS_NUM* 300.

Επιλέχθηκε το 300 για όλα τα είδη οργάνωσης αρχείου, το οποίο κάλλιστα μπορεί να αλλαχθεί.

Τροποποιημένα Αρχεία

- Αρχείο σωρού:
  1. *./examples/hp_main.c*
  2. *./include/hp_file.h*
  3. *./src/hp_file.c*

- Αρχείο στατικού κατακερματισμού:
  1. *./examples/ht_main.c*
  2. *./include/ht_table.h*
  3. *./src/ht_table.c*

Μεταγλώττιση & Εκτέλεση

Οι ακόλουθες εντολές λειτουργούν στο αρχικό *directory*, δηλαδή μέσα στον κατάλογο που περιέχει το αρχείο *Makefile*.

- Για να μεταγλωττίσετε και να εκτελέστε το εκτελέσιμο αρχείο,
  - για την υλοποίηση του αρχείου σωρού, πληκτρολογήστε:

    make hp; **./build/hp_main**

  - για την υλοποίηση του αρχείου στατικού κατακερματισμού, πληκτρολογήστε:

    make ht; **./build/ht_main**

- Για να διαγράψετε το αρχείο *data.db*, πληκτρολογήστε:

  rm data.db

(A) Συναρτήσεις HeapFile

- **HP_CreateFile**

Η συνάρτηση $HP\_CreateFile()$ αρχικά δημιουργεί με την βοήθεια της συνάρτησης $BF\_CreateFile()$ (της βιβλιοθήκης $BF\ block$ $(bf.h)$), ένα αρχείο στον σκληρό δίσκο με όνομα $data.db$. Στην συνέχεια με την κλήση της $BF\_OpenFile()$, επιτυγχάνεται το άνοιγμα του υπάρχοντος πλέον αρχείου με όνομα $data.db$ και αναχτάται ο περιγραφέας αρχείου, ο οποίος αντιγράφεται στην μεταβλητή fd.Έπειτα δεσμεύεται και αρχικοποιείται στην υπάρχουσα ενδιάμεση μνήμη ένα μπλοκ, μέσω της συνάρτησης $BF\_Block\_Init()$, και κατόπιν με την κλήση της $BF\_AllocateBlock()$, δεσμεύεται στο τέλος του αρχείου σωρού ένα καινούριο μπλοκ, το οποίο είναι το αντίστοιχο μπλοκ, αυτού του μπλοκ που είχαμε πριν δεσμεύσει στην ενδιάμεση μνήμη.Έχοντας τελειώσει τη τοποθέτηση, με τη βοήθεια της $BF\_Block\_GetData()$ γίνεται η λήψη ενός δείκτη που αντιγράφεται στη μεταβλητή $data$, με τον οποίο γίνεται εφικτή η πρόσβαση στο μπλοκ (στη τρέχουσα κατάσταση είναι το πρώτο) της ενδιάμεσης μνήμης. Το αρχικό μπλοκ θα χρησιμοποιηθεί για την αποθήκευση χρήσιμων δεδομένων, που θα βοηθήσουν στην διαχείριση των εγγραφών. Έτσι στη συνέχεια, έχοντας δημιουργήσει μια δομή $struct$ με όνομα HP_info, ορίζουμε έναν δείκτη σε αυτή την δομή ($heapFile\_info$), τον αρχικοποιούμε να δείχνει στην αρχή του αρχικού μπλοκ της ενδιάμεσης μνήμης και θέτουμε τιμές στα μέλη της δομής. Τέλος ορίζουμε το αρχικό μπλοκ ως $dirty$ (μέσω της $BF\_Block\_SetDirty()$), ώστε όταν διαγραφεί το μπλοκ από την ενδιάμεση μνήμη και γραφεί στο δίσκο, αφού πρώτα ξεκαρφιτσωθεί ($BF\_UnpinBlock()$), να ενημερωθεί το αρχείο στο δίσκο με τις αλλαγές που έγιναν στο μπλοκ, δηλαδή ενημερώνεται το αντίστοιχο μπλοκ στο δίσκο, και μετά να γίνει η αποδέσμευση της μνήμης που καταλαμβάνει η δομή BF_BLOCK με την χρήση της συνάρτησης $BF\_Block\_Destroy()$. Το τελευταίο βήμα πριν την επιτυχημένη ολοκλήρωση της λειτουργίας, της $HP\_CreateFile()$, είναι να διαγράψουμε τον περιγραφέα αρχείου, κάνοντας τον, διαθέσιμο για νέα χρήση, κλείνοντας έτσι το αρχείο ($BF\_CloseFile()$).

- **HP_OpenFile**

Σκοπός της συνάρτησης $HP\_OpenFile()$, είναι να επιστρέψει ένα δείκτη που δείχνει στο μπλοκ με τα μεταδεδομένα του αρχείου σωρού.Το πρώτο πράγμα που κάνει η συνάρτηση είναι να ανοίξει εκ νέου το υπάρχον αρχείο με όνομα $data.db$ και μέσω του περιγραφέα αρχείου (με την βοήθεια της $BF\_GetBlock()$) , βρίσκει το πρώτο μπλοκ, το οποίο έχουμε ορίσει ότι θα περιέχει τα μεταδεδομένα.Πριν την επιστροφή του δείκτη η συνάρτηση ελέγχει αν το αρχείο που ανοίχτηκε είναι τύπου σωρός.Στην περίπτωση που το αρχείο είναι $heap\ file$ ανανεώνονται από το $struct\ HP\_info$ το μέλος με το $file\ descriptor$, διότι υπάρχει περίπτωση να έχει δοθεί καινούριος περιγραφέας από αυτόν που είχαμε λάβει κατά την λειτουργία της $HP\_CreateFile()$ (για τον λόγο ότι τον κλείσαμε στο τέλος της $HP\_CreateFile()$) και την διεύθυνση του μπλοκ κεφαλής διότι μπορεί το μπλοκ να δεσμευτεί σε άλλη διεύθυνση. Τελειώνοντας επιστρέφεται ο δείκτης στην συνάρτηση που κάλεσε την $HP\_OpenFile()$.Διαφορετικά αν το εξετάσιμο αρχείο δεν είναι αρχείο σωρού η συνάρτηση σταματάει ακαριαία επιστρέφοντας "$NULL$".

- **HP_CloseFile**

Η συνάρτηση $HP\_CloseFile()$ αποδεσμεύει($BF\_UnpinBlock()$ και $BF\_Block\_Destroy()$) από το σύστημα το ενεργό μπλοκ με τα μεταδεδομένα αλλά και διάφορους δείκτες. Συγκεκριμένα, κάνουμε $free$ το $string\ type[\ ]$ που είναι αποθηκευμένο στο $struct$. Τέλος με την βοήθεια της μεθόδου $BF\_Close\_File()$ γίνεται το κλείσιμο του ανοιχτού αρχείου, διαγράφοντας έτσι τον περιγραφέα αρχείου.

- **HP_InsertEntry**

Η συνάρτηση $HP\_InsertEntry()$ εισάγει μία εγγραφή, που της έχει δοθεί ως όρισμα, σε ένα διαθέσιμο μπλοκ, για χάρη της συνάρτησης που την κάλεσε.Αρχικά εξετάζει πόσα μπλοκ υπάρχουν στο αρχείο σωρού με την χρήση της μεθόδου $BF\_GetBlockCounter()$.

Ανάλογα με το πλήθος των μπλοκ υλοποιούνται οι εξής συνθήκες:

◦ Η κατάσταση στην οποία βρίσκεται το αρχείο σωρού, στην πρώτη συνθήκη, είναι ότι υπάρχουν περισσότερα από ένα μπλοκ. Σε αυτή την περίπτωση εξετάζεται αν το τελευταίο μπλοκ που δεσμεύτηκε στο σωρό έχει διαθέσιμο χώρο για

την εισαγωγή της εγγραφής.Έτσι αρχικά φορτώνουμε το τελευταίο μπλοκ στην ενδιάμεση μνήμη, (το αναγνωριστικο του οποίου είναι αποθηκευμένο στη κεφαλή) και εξετάζουμε το πληθος των εγγραφών του.

Υπάρχουν δύο περιπτώσεις:

1. Να υπάρχει ελεύθερος χώρος για τοποθέτηση εγγραφής. Σε αυτή την περίπτωση εισάγουμε το *record*, ενημερώνουμε τα *metadata* του μπλοκ και τέλος θέτουμε το τρέχον μπλοκ ως *dirty*, το ξεκαρφιτσώνουμε και αποδεσμεύουμε την μνήμη που καταλαμβάνει η δομή αυτού του μπλοκ.

2. Το μπλοκ να είναι γεμάτο. Σε αυτή την περίπτωση, ενεργοποιείται στο τρέχον μπλοκ, η σημαία *unpin* και μετά αποδεσμεύεται η μνήμη που καταλαμβάνει η δομή BF_BLOCK. Τέλος ενεργοποιείται η σημαία δέσμευσης καινούργιου μπλοκ (*allocate_new_block*), ώστε η επόμενη συνθήκη να ικανοποιείται και να δεσμεύσει νέο μπλοκ για να αποθηκευτεί η εγγραφή.

○ Στην δεύτερη συνθήκη ελέγχεται αν το πλήθος των μπλοκ είναι ίσο με 1 ή έχει ενεργοποιηθεί η σημαία της δέσμευσης καινούργιου μπλοκ από την προηγούμενη συνθήκη, δηλαδη αν είμαστε στην κατάσταση όπου είτε δεν υπάρχουν μπλοκ αποθήκευσης εγγραφών, είτε υπάρχουν μόνο γεμάτα μπλοκ και πρέπει να δεσμεύσουμε καινούριο μπλοκ για την προσθήκη της εγγραφής. Για τον λόγο αυτό, μέσα στο σώμα της συνθήκης, αρχικά δεσμεύεται ένα μπλοκ στην ενδιάμεση μνήμη και κατόπιν δεσμεύεται και το αντίστοιχο μπλοκ στο αρχείο σωρού.Στην συνέχεια γίνεται η εισαγωγή του *record* και οι απαραίτητες ενημερώσεις τόσο των μεταδεδομένων του τρέχοντος μπλοκ όσο και των μεταδεδομένων του μπλοκ κεφαλής.Τέλος θέτουμε την σημαία *dirty* στο μπλοκ κεφαλής, διότι ενημερώθηκε το μέλος με το αναγνωριστικό του τελευταίου μπλοκ, στο μπλοκ που μόλις έγινε η εγγραφή, του ορίζεται και σε αυτό η σημαία *dirty* μέσω της μεθόδου $BF\_Block\_SetDirty()$, ώστε να γίνει η αντιγραφή του πλέον τροποποιημένου μπλοκ, στο αντίστοιχό του, το οποίο βρίσκεται στο αρχείο σωρού.Ολοκληρώνοντας το ξεκαρφιτσώνουμε και αποδεσμεύουμε την μνήμη που καταλαμβάνει.

- **HP_GetAllEntries**

Σκοπός της συνάρτησης είναι να βρει αν υπάρχει εγγραφή με κλειδί ίσο με τιμή *value*, στο αρχείο σωρού, διαφορετικά επιστρέφει -1.Αρχικά η συνάρτηση δεσμεύει ένα βοηθητικό μπλοκ στην ενδιάμεση μνήμη όπου θα της είναι χρήσιμο για την ανάγνωση των περιεχομένων των μπλοκ, ώστε να βρει την εγγραφή που αντιστοιχεί στο $id = value$.Έπειτα λαμβάνει μέσω της μεθόδου $BF\_GetBlockCounter$ τον αριθμό των μπλοκ του αρχείου σωρού για να ξέρει ποιο είναι το μέγιστο πλήθος μπλοκ που θα εξετάσει.Στη συνέχεια μέσα σε ένα *while* βρόχο ελέγχει ένα-ένα τα μπλοκ του σωρού. Τέλος όταν βρεθεί η ζητούμενη εγγραφή η εξέταση σταματάει και επιστρέφεται στην συνάρτηση που την κάλεσε, το πλήθος μπλοκ που διαβάστηκαν μέχρι να βρεθεί το μπλοκ.

- **Βοηθητικές Συναρτήσεις**

1. $print\_Metadata\_HeapFile$:

   $Print$ τα μετα-δεδομένα. Χρησιμοποιείται στην συνάρτηση $HP\_OpenFile$.

2. $print\_HeapFile$:

   Εκτυπώνει το αρχείο σωρού. Χρησιμοποιείται στην $main()$, του αρχείο *hp_main.c*, αλλά έχει σχολιαστεί, το χρησιμοποιούμε αποκλειστικά για *debugging*.

- Βοηθητικά structs με τα μέτα-δεδομένα -

1. $struct\ HP\_info$:

Σε αυτή τη δομή κρατάμε τις πληροφορίες, οι οποίες είναι σημαντικές για την διαχείριση του αρχείου σωρού.Το *struct* αυτό είναι αποθηκευμένο στο πρώτο μπλοκ, τόσο στο αρχείο σωρού όσο και στο αντίστοιχο μπλοκ της ενδιάμεσης μνήμης, ώστε να είναι άμεσα προσπελάσιμο.Το μπλοκ 0 είναι καρφιτσωμένο καθόλη τη διάρκεια της διαχείρισης του αρχείου, το ξεκαρφιτσώνουμε με τις αλλαγές, στην κλήση της $HP\_CloseFile()$, δηλαδή κατά το κλείσιμο του αρχείου.

Τα μέλη της δομής είναι τα εξής:

fileDesc: Ο αναγνωριστικός αριθμός ανοίγματος αρχείου *data.db* από το επίπεδο *block*.

fileType: Ο τύπος του αρχείου (π.χ. $Heap\ File$).

recordSize: Το μέγεθος μιας εγγραφής.

maxRecords_per_Block: Το μέγιστο πλήθος εγγραφών που χωράει ένα μπλοκ.

lastBlock_id: Το αναγνωριστικο του τελευταίου μπλοκ που εκχωρήθηκε στο τέλος του αρχείου σωρού.

header_block: Δείκτης στο μπλοκ κεφαλής, (με αναγνωριστικό $id = 0$) που έχει αποθηκευμένα τα μεταδεδομένα του αρχείου σωρού.

2. *struct* HP_block_info:

Το *struct* αυτό κρατάει τα *metadata* για κάθε μπλοκ, το οποίο είναι αποθηκευμένο στο τέλος του κάθε μπλοκ.Η πληροφορία που κρατάει αποθηκευμένη είναι το τρέχον πλήθος εγγραφών κάθε μπλοκ ($numberOfRecords$).

(B) Συναρτήσεις HashTable

- **HT_CreateFile**

Η συνάρτηση $HT\_CreateFile()$ λειτουργεί παρόμοια με την $HP\_CreateFile$. Εκτός από τις προφανείς αλλαγές (π.χ. χρήση του *struct* HT_info έναντι του *struct* HP_info για την αναπαράσταση των μεταδεδομένων του αρχείου, το *string type*[ ] είναι "$Hash\ File$" και όχι "$Heap\ File$"), στην συνάρτηση αυτή γίνεται επίσης και η δέσμευση/αρχικοποίηση στην υπάρχουσα ενδιάμεση μνήμη των αρχικών $block-buckets$ μέσω της συνάρτησης $BF\_Block\_Init()$. Κατόπιν με την κλήση της $BF\_AllocateBlock()$, δεσμεύεται στο τέλος του αρχείου κατακερματισμού ένα καινούριο μπλοκ, το οποίο είναι το αντίστοιχο μπλοκ αυτού που είχαμε δεσμεύσει στην ενδιάμεση μνήμη. Στο τέλος, αφού γράψουμε για κάθε τέτοιο *block* τα μετα-δεδομένα (αριθμο των *records* στο *block*, το οποίο αρχικά είναι 0 και το *id* του επόμενου *block*, στην περίπτωση υπερχείλισης, το οποίο αρχικά είναι -1) του στην ενδιάμεση μνήμη, τα κάνουμε *dirty* (μέσω της $BF\_Block\_SetDirty()$), ώστε όταν διαγραφεί το μπλοκ από την ενδιάμεση μνήμη και γραφεί στο δίσκο, αφού πρώτα ξεκαρφιτσωθεί ($BF\_UnpinBlock()$), να ενημερωθεί το αρχείο στο δίσκο με τις αλλαγές που έγιναν στο μπλοκ, και μετά να γίνει η αποδέσμευση της μνήμης που καταλαμβάνει η δομή $BF\ BLOCK$ με την χρήση της συνάρτησης $BF\_Block\_Destroy()$.

- **HT_OpenFile**

Η λειτουργία της συνάρτησης $HT\_OpenFile()$ είναι παρόμοια με αυτή της $HP\_OpenFile()$. Η διαφορά εδώ είναι ότι γίνεται έλεγχος, εάν το αρχείο που ανοίχτηκε είναι $Hash\ File$.

- **HT_CloseFile**

Η συνάρτηση $HT\_CloseFile()$ αποδεσμεύει($BF\_UnpinBlock()$ και $BF\_Block\_Destroy()$) από το σύστημα το ενεργό μπλοκ με τα μεταδεδομένα αλλά και διάφορους δείκτες. Συγκεκριμένα, κάνουμε $free$ το *string type*[ ] και τον πίνακα κατακερματισμού $**HashTable$ που είναι αποθηκευμένα στο *struct*, καθώς επίσης και την *global* συμβολοσειρά ($fileName\_HT$) που κρατάει το όνομα του ανοιχτού αρχείου. Τέλος με την βοήθεια της μεθόδου $BF\_Close\_File()$ γίνεται το κλείσιμο του ανοιχτού αρχείου, διαγράφοντας έτσι τον περιγραφέα αρχείου.

- **HT_InsertEntry**

Η συνάρτηση αυτή διαβάζει και υπολογίζει διάφορα δεδομένα και πληροφορίες από το μπλοκ 0 με τα μετα-δεδομένα, όπως το *id* του *bucket* που θέλουμε να κάνουμε εισαγωγή, το μέγιστο αριθμό *records* που χωράει κάθε *bucket* και πόσα *records* είναι αποθηκευμένα σε αυτό *bucket*.Πριν την εισαγωγή του *record* ελέγχουμε εάν το *Bucket* είναι γεμάτο, αν βρισκόμαστε στην περίπτωση όπου δεν υπάρχει άλλος διαθέσιμος χώρος, τότε δημιουργούμε νέο *bucket* υπερχείλισης του αρχικού *bucket*-εισαγωγής, εισάγουμε την εγγραφή και ανανεώνουμε τον πίνακα κατακερματισμού στα μεταδεδομένα αλλά και τα μεταδεδομένα του *Bucket* που επιλέχθηκε αρχικά (το γεμάτο).Διαφορετικά εισάγουμε την εγγραφή στο διαθέσιμο χώρο του *Bucket*.

- **HT_GetAllEntries**

Σκοπός της συνάρτησης είναι να βρει αν υπάρχει εγγραφή με κλειδί ίσο με τιμή *correctId*, διαφορετικά επιστρέφει -1. Αρχικά η συνάρτηση δεσμεύει ένα βοηθητικό μπλοκ στην ενδιάμεση μνήμη όπου θα της είναι χρήσιμο για την ανάγνωση των περιεχομένων των μπλοκ, ώστε να βρει την εγγραφή που αντιστοιχεί στο $id = value$. Έπειτα υπολογίζει το *hash_id* του κλειδιού (μέσω της $hashFunction()$) και μετά με την συνάρτηση $BF\_GetBlock()$ παίρνουμε το πρώτο μπλοκ με το σωστό *id* γιά αναζήτηση.Εάν το *record* δεν βρεθεί σε αυτό το *block*, τότε εξαντλητικά αναζητούμε το επόμενο *bucket* υπερχείλισης (μέσω του HT_block_info) μέχρι να βρεθεί το *record*.Εάν δεν υπάρχει άλλο *block* υπερχείλισης και δεν έχει βρεθεί το *record*, επιστρέφουμε *error*. Πριν την οποιαδήποτε επιστροφή, αποδεσμεύουμε τα βοηθητικά *blocks* (μέσω $BF\_UnpinBlock()$ και $BF\_Block\_Destroy()$).Με το παραπάνω τρόπο μειώνουμε πολύ τον αριθμό των *blocks* που διαβάζουμε μέχρι να βρούμε το σωστό *record* (αλλά και τον αριθμό των *blocks* που διαβάζουμε μέχρι να καταλάβουμε ότι ένα τέτοιο *record_id* δεν υπάρχει στο αρχείο μας), το οποίο αποτελεί ένα από τα πλεονεκτήματα του αρχείου κατακερματισμού.

- **Βοηθητικές Συναρτήσεις**

1. *hashFunction*:

   Δέχεται ως όρισμα τον αριθμό για *hash* (*id* ενός *record*) και τον αριθμό των αρχικών *buckets* και επιστρέφει το *id* του *block* που θα αντιστοιχεί στο *record* αυτό.(Επιστρέφει $hashNumber + 1$ διότι το *block* 0 με τα μετα-δεδομένα πρέπει να αγνοηθεί, και να μην θεωρηθεί ως πιθανό *bucket* εισαγωγής)

2. $print\_Metadata\_SecondaryIndex$:

   Print τα μετα-δεδομένα. Χρησιμοποιείται στην συνάρτηση $HT\_OpenFile$.

3. $print\_SecondaryIndex$:

   Εκτυπώνει το αρχείο πρωτεύοντος ευρετηρίου. Χρησιμοποιείται στις $main()$, των αρχείων *ht_main.c*, *sht_main.c* και *statistics_main.c*, αλλά έχουν σχολιαστεί, το χρησιμοποιούμε αποκλειστικά για *debugging*.

- Βοηθητικά structs με τα μέτα-δεδομένα -

1. $struct\ HT\_info$:

Το *struct* αυτό κρατάει τα *metadata* ολόκληρου του αρχείου και γράφεται στην θέση *block* 0.Περιέχει διάφορα *variables* τα οποία μας βοηθούν να χειριστούμε το αρχείο κατακερματισμού.Το *block* αυτό καρφιτσώνεται στην μνήμη με την $openfile$ και παραμένει καρφιτσωμένο μέχρι το κλείσιμο του αρχείου.

Πιο συγκεκριμένα:

fileDesc: Ο αναγνωριστικός αριθμός ανοίγματος αρχείου *data.db* από το επίπεδο *block*.

InfoAboutHTable Κρατάει δείκτη στο *block*0 με τα μετα-δεδομένα. Χρησιμοποιείται στο $closeFile$, για να κάνουμε το *Block* αυτο *unpin*.

HashTable: Πίνακας κατακερματισμού. Πρώτη στήλη[$bucketId$]: κρατάει το id των αρχικών $Buckets$ πριν την υπερχείλιση, Δεύτερη στήλη[$lastBucketWithFreeSpace$]: Για κάθε τέτοιο αρχικό $Bucket$ δείχνει το τελευταίο $block$-υπερχείλισης με ελεύθερο χώρο.
Αρχικά $bucketId == lastBucketWithFreeSpace$.

recordSize: Η χωρητικότητα μιας εγγραφής.

numberOfRecordsInBlock: Η χωρητικότητα ενός $block$ σε εγγραφές.

numOfBuckets: Το πλήθος των αρχικών "κάδων" του αρχείου κατακερματισμού. Δίνεται στην $HT\_CreateFile$ μέσω της $main$.

fileType: $'Hash\ Table'$ *or* $'Heap\ File'$.

2. *struct* HT_block_info:

Το *struct* αυτό κρατάει τα *metadata* για κάθε *block*. Ακολουθώντας παρόμοια τακτική με το HT_info το *struct* αυτό φορτώνεται στην τελευταία θέση του κάθε *block*.Οι πληροφορίες που κρατάει είναι το πλήθος των εγγραφών ($numberOfRecords$) και το αναγνωριστικό του επόμενου μπλοκ υπερχείλισης ($nextBlockId$).