ΥΛΟΠΟΙΗΣΗ ΣΥΣΤΗΜΑΤΩΝ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

ΕΡΓΑΣΙΑ 1

ΠΕΚΟΣ ΕΥΑΓΓΕΛΟΣ 1115201400157

<<ΥΛΟΠΟΙΗΣΑ ΤΟ heap_file_64 >>

Στην πρώτη ασκηση του μαθήματος υλοποιήσα όλες τις ζητούμενες συναρτήσεις.

- **HP_ErrorCode HP_CreateFile :** Εδω χρησιμοποιήσα τις αντιστοιχες συναρτήσεις BF για δημιουργια αρχείου και άνοιγμα αρχείου.Επειτα αρχικοποιήσα το block (Block_Block_Init) και δημιουργία (BF_AllocateBlock). Δημιούργησα μόνο το πρωτο block και η ενδείξη που δηλώνει οτι είναι ενα heap file είναι ενα string “Heap” και κλείνω το αρχείο αφού κάνω SetDirty και UnpinBlock

- **HP_ErrorCode HP_OpenFile:** Εδω ανοιγω το άρχειο με την αντιστοιχη BF συνάρτηση , αρχικοποιώ το block (BF_Block_Init) και ελεγξα αν όντως πρόκειται για heap file κάνοντας memcmp το πρωτο στοιχειο του πρώτου block με το string “Heap”. Τελος έκανα unpin.

- **HP_ErrorCode HP_CloseFile:** Εδω απλά καλώ την αντίστοιχη συνάρτηση BF_CloseFile

- **HP_ErrorCode HP_InsertEntry:** Εδώ αρχικά έκανα διάφορους ελέγχους. Πήρα τον αριθμό των μπλοκ απο την BF_GetBlockCounter και στην περίπτωση που υπήρχε μόνο ενα μπλοκ στο αρχείο , δηλαδή αυτό με τα metadata έφτιαχνα καινούριο , έβαζα counter ο οποιός απλα μετράει τις εγγραφές και εκανα insert. Αν υπηρχαν τουλάχιστον 2 μπλοκ έλεγχα αν το τελευταιό μπλοκ (σειριακη εισαγωγη) ειχε χώρο για την έγγραφη. Αν είχε ανανέωνα τον counter και έβαζα την εγγραφή. Αν δεν είχε έφτιαχνα νεο μπλοκ. Για την προσπέλαση μέσα σε ενα μπλοκ έκανα το έξης: Στα πρώτα 4 bytes βρισκοταν ο counter , οπότε για να προσπελάσω εγγραφές , μετακινούσα τον δείκτη με το sizeof(Record) και αναλογα του counter πηγαινα στην πρώτη διαθέσιμη θέση, άλλα ειχα συμπεριλάβει και τα 4 bytes του counter, δηλαδή : data += (counter -1)*sizeof(Record) + sizeof(int). Τέλος εκανα SetDirty και UnpinBlock

- **HP_ErrorCode HP_PrintAllEntries:** Εδώ για κάθε μπλοκ του αρχείου (εκτός του πρώτου) εκανα σειριάκη προσπέλαση στις εγγραφές και τις εκτύπωνα μια μια. Το σειριάκο το εξασφάλισα κάνοντας αρχικά τον δεικτη να δείχνει στην πρώτη εγγραφή δηλαδλή 4 bytes δεξία (λόγω του counter) και σε κάθες επανάληψη προσθετα sizeof(Record). Τέλος έκανα UnpinBlock.

- **HP_ErrorCode HP_GetEntry:** Εδώ διαιρούσα το rowId με τον μέγιστο αριθμό εγγραφών ανά μπλοκ για να βρω σε ποιό μπλοκ θα πρέπει να κοιτάξω. Αν η διαίρεση δεν είχε υπολοιπο τότε θα κοίταζα στο μπλοκ που έδειχνε η διάιρεση (πχ 80/20 =4 8α πηγαινα στον 4ο καδο) και θα έπερνα την τελευταία εγγραφή, αλλιώς (δηλαδή είχαμε υπολοιπο) στο επόμενο μπλοκ και θα έπερνα την εγγραφή που έδειχνε το υπόλοιπο (πχ 87/20 = 4 και υπολοιπο 7 αρα παω στο 5ο μπλοκ και 7η εγγραφη) Όταν διάλεγα μπλοκ για να βρω εγγραφή , φρόντιζα να πηγαλινω στο επόμενο από οτι θα έπρεπε κανονίκα μιας και στο πρώτο μπλοκ δεν ύπηρχαν εγγραφές και δεν θέλαμε να ψάξουμε εκέι. (πχ BF_GetBlock(fileDesc ,block_to_go + 1 ,block); το +1 που γίνεται μας εξασφάλιζει οτι θα πάμε στο σωστό μπλοκ)

Σε όλες τις συνάρτησεις αρχικοποιούσα καταλληλα οτι χρειάζονταν , έκανα τα απαραίτητα unpins και καθε φορά που έκανα κάποια αλλαγή φρόντιζα να κάνω και SetDirty.
Χρησημοποίησα την memcpy για να αντιγραφω τα δεδόμενα που χρειάζομουν κάθε φορά.
Επίσης να αναφέρω οτι στην υλοποίηση μου για την επιστροφή κωδικού λάθους στις συναρτήσεις που καλώ που σχετίζονται με το επίπεδο block χρήσιμοποιήσα την συνάρτηση CALL_OR_DIE οπώς αυτή χρησιμοποιήται στην main που μας δώθηκε.Δηλαδή αντι σε κάθε κλήση των συναρτήσεων του επιπεδου μπλοκ , αντί κάθε φορά να ελεγχω αν επιστρέφει BF_OK ή αλλιως να καλω την BF_PrintError χρησιμοποίω την CALL_OR_DIE η οποία τα κάνει αυτά.

BONUS ΕΡΩΤΗΜΑ

Παρατηρώ οτι χρησιμοποιώντας τον αλγόριθμο LRU έχουμε πόλλα παραπάνω reads σε σχέση με τον MRU αλγόριθμο αντικατάστασης. Πιο συγκεκριμένα στην υλοποίηση μου:

	LRU	**MRU**
READS	50105	10621
WRITES	1907	1907

Αυτό συμβαίνει επειδή η MRU θα αντικαταστήσει το τελευταιό μπλοκ οταν πρέπει να εισάγει ενα νεο , ενω από την άλλη η LRU θα αντικαταστησει το πρωτο. Ωστοσο το πρώτο μπλοκ θα χρειάστει να το διαβάσουμε πιο νωρίς απο ότι οτι το τελευταίο το οποιό η LRU έχει αντικαταστήσει. Ετσι προκύπτουν τα παραπάνω reads.