Θεοδώρα Αρχοντάκη

Ελευθερία Εκατομμάτη

################### ΓΙΑ ΤΑ ΑΡΧΕΙΑ ΣΩΡΟΥ ###################

-> Έχουμε φτιάξει ένα struct HP_block_info που περιέχει:
Πληροφορίες για το κάθε block. Έχουμε ένα δείκτη στο επόμενο block
και έναν ακέραιο που μετράει τις εγγραφές του κάθε block.

-> Έχουμε μία δομή HP_block που περιέχει:
Δείκτη στο block και έναν δείκτη σε ένα struct HP_block_info
που αποθηκεύουμε τις πληροφορίες του κάθε block οπώς αναλύσαμε
από πάνω.

-> Έχουμε ένα stuct HP_info που περιέχει πληροφορίες για το αρχείο:
Έχουμε ένα δείκτη που δείχνει στο block0 και έναν για το τελευταίο
block του αρχείου. Υπάρχουν και τρεις ακέραιοι ένας για τον αριθμό
των blocks στο αρχείο, ένας που αποθηκεύουμε τον αναγνωριστικό
κωδικό του αρχείου και ένα flag που ορίζει αν είναι αρχείο κατακερματισμού
ή σωρού (αν flag = 0 αρχείο σωρού αν flag = 1 αρχείο κατακερματισμού).

-> Για την HP_CreateFile:
Δεσμέυουμε μνήμη για έναν δείκτη που δείχνει στο αρχείο και για
το block0. Αρχικοποιούμε τον αναγνωριστικό αριθμό του αρχείου σε μηδέν
κάνουμε count_b = 0 αφού ακόμα δεν έχουμε φτιάξει κάποιο block και αρχικοποιούμε
το flag σε μηδέν αφου φτιάχνουμε αρχείο σωρού.
Και καλούμε την BF_CreateFile.

-> Για την HP_OpenFile:
Αυξάνουμε κατά ένα το fileDesc κάθε φορά που καλείται η συνάρτηση
και έτσι μπορούμε να ξέρουμε πόσες φορές έχουμε ανοίξει το αρχείο.

Καλούμε την BF_OpenFile, αρχικοποιούμε το block0 και
δεσμεύουμε την κατάλληλη μνήμη. Ο δείκτης last_entry δείχνει πλέον
στο πρώτο block (block0) στο οποίο δεν βάζουμε εγγραφές οπότε αρχικοποιούμε
το count των εγγραφών για το block0 σε έξι έτσι ώστε όταν θα καλέσουμε
πρώτη φορά την HP_InsertEntry θα δημιουργηθεί το block1.
Και αυξάνουμε τον count_b (αριθμός των blocks στο αρχείο).
Κάνουμε έλεγχο αν είναι αρχείο σωρού.

-> Για την HP_CloseFile:

Κλείνει το αρχείο και αποδεσμέυει την μνήμη που χρησιμοποιήθηκε.

-> Για την HP_InsertEntry:

Αν ο αριθμός των εγγραφών σε ένα block είναι μικρότερο από έξι
τότε δεν δημιουργούμε νέο block και βάζουμε την εγγραφή στο ήδη
υπάρχον. Σε έναν ακέραιο αποθηκεύουμε πόσες εγγραφές έχει ήδη το block
οπότε στην επόμενη "ελεύθερη θέση" βάζουμε την εγγραφή και αυξάνουμε
το count των εγγραφών κατά ένα. Καλούμε την BF_Block_SetDirty
για να δηλώσουμε ότι αλλάξαμε τα δεδομένα οπότε πρέπει να αλλαχτούν
και στο δίσκο και κάνουμε UnPin το block.

Αν ο αριθμός των εγγραφών στο block είναι ίσος με έξι (άρα δεν
χωράει άλλη εγγραφή στο συγκεκριμένο block) αρχικοποιούμε και δεσμέυουμε
μνήμη για το επόμενο block από το last_entry. Αποθηκεύουμε στο καινούριο
block την εγγραφή (θα είναι η εγγραφή[0]). Καλούμε την BF_Block_SetDirty
για να δηλώσουμε ότι αλλάξαμε τα δεδομένα οπότε πρέπει να αλλαχτούν
και στο δίσκο και κάνουμε UnPin το block. Αυξάνουμε το count_b κατά ένα αφού
φτιάξαμε ένα καινούριο block. Και το count_records του κανούριου block
είναι ίσο με ένα αφού έχουμε μία εγγραφή.
Επιστρέφουμε σε ποιο block βάλαμε την εγγραφή που θα είναι ο αριθμός
των blocks στο αρχείο αφού βάζουμε την εγγραφή στο τελευταίο block.

-> Για την HP_GetAllEntries:

Φτιάχνουμε ένα HP_block που δείχνει στο πρώτο block μετά το block0 (άρα το πρώτο block στο οποίο βάζουμε εγγραφές). Για όλα τα blocks ψάχνουμε όλες τις εγγραφές αν κάποια εγγραφή έχει id ίσο με value τότε τυπώνουμε την εγγραφή και επιστρέφουμε πόσα blocks ψάξαμε.

#################### ΓΙΑ ΤΑ ΑΡΧΕΙΑ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΥ ####################

-> Έχουμε φτιάξει ένα struct HT_block_info που περιέχει:

Πληροφορίες για το κάθε block. Έχουμε ένα δείκτη στο επόμενο block (block υπερχείλισης)

έναν ακέραιο που μετράει τις εγγραφές του κάθε block και έναν ακέραιο που δηλώνει την σειρά με την οποία φτιάχτηκε το block.

-> Έχουμε μία δομή HT_block που περιέχει:

Δείκτη στο block και έναν δείκτη σε ένα struct HT_block_info που αποθηκεύουμε τις πληροφορίες του κάθε block οπώς αναλύσαμε από πάνω.

-> Έχουμε ένα stuct HT_info που περιέχει πληροφορίες για το αρχείο:

Έχουμε ένα δείκτη που δείχνει στο block0. Υπάρχουν και τρεις ακέραιοι ένας για τον αριθμό των blocks στο αρχείο, ένας που αποθηκεύουμε τον αναγνωριστικό κωδικό του αρχείου και ένα flag που ορίζει αν είναι αρχείο κατακερματισμού ή σωρού (αν flag = 0 αρχείο σωρού αν flag = 1 αρχείο κατακερματισμού). Υπάρχει και ένας long int που είναι ίσος με τον αριθμό των buckets στο αρχείο. Και έχουμε ένα δυσδιάστοτο πίνακα που δείχνει σε HT_Block και κάθε γραμμή αφορά το αντίστοιχο block (π.χ. η γραμμή 1 αναφέρεται στο πρώτο block που δέχεται εγγραφές),

η πρώτη στήλη δείχνει στο βασικό block και η δεύτερη
στήλη δείχνει στο τελευταίο block υπερχείλισης (αν έχει δημιουργηθεί)
για το συγκεκριμένο bucket (αν δεν υπάρχει block υπερχείλισης τότε δεν δείχνει
κάπου). Υπάρχει και ένας πίνακας ακεραίων που αποθηκεύουμε για κάθε block
πόσα block υπερχείλισης έχει. Έχουμε και έναν πίνακα ακεραίων που σε καθε θέση i
έχει τον αριθμό των εγγρφών για το bucket i.

-> Για την HT_CreateFile:
Δεσμέυουμε μνήμη για έναν δείκτη που δείχνει στο αρχείο, για
το block0 και για το δυσδιάστατο πίνακα τύπου HT_block.
Αρχικοποιούμε τον αναγνωριστικό αριθμό του αρχείου σε μηδέν
κάνουμε count_b = 0 αφού ακόμα δεν έχουμε φτιάξει κάποιο block, αρχικοποιούμε
το flag σε ένα αφου φτιάχνουμε αρχείο κατακερματισμού ,κάνουμε το numBuckets
ίσο με τον ακέραιο buckets που είναι παράμετρος της συνάρτησης
και αρχικοποιούμε με μηδέν τον αριθμό των blocks υπερχείλισης
για κάθε κάδο αφού με το που φτιάχνεται το αρχείο δεν υπάρχουν blocks περχείλισης.
Και καλούμε την BF_CreateFile.

-> Για την HT_OpenFile:
Αυξάνουμε κατά ένα το fileDesc κάθε φορά που καλείται η συνάρτηση
και έτσι μπορούμε να ξέρουμε πόσες φορές έχουμε ανοίξει το αρχείο.
Καλούμε την BF_OpenFile και δεσμεύουμε την κατάλληλη μνήμη για
το block0 το οποίο δεν περιέχει εγγραφές μόνο τα μεταδεδομένα του αρχείου.
Και αυξάνουμε τον count_b (αριθμός των blocks στο αρχείο). Κάνουμε έλεγχο
αν είναι αρχείο κατακερματισμού.

-> Για την HT_CloseFile:
Κλείνει το αρχείο και αποδεσμέυει την μνήμη που χρησιμοποιήθηκε.

-> Έχουμε φτιάξει δύο βοηθητικές συναρτήσεις:

Την hash_function που δέχεται δύο ακέραιους (a, base) και επιστρέφει a mod base + 1. Είναι η συνάρτηση κατακερματισμού που χηρσιμοποιούμε για να εισάγουμε κάποια εγγραφή στο αρχείο. Το base θα είναι NUM_Buckets Έχουμε βάλει να επιστρέφει το υπόλοιπο της διαίρεσης συν ένα γιατί στο block0 δεν αποθηκεύουμε εγγραφές οπότε πρέπει να ξεκινάμε την εισαγωγή από το block1.

Την find που δέχεται έναν ακέραιο και έναν πίνακα από ακεραίους και επιστρέφει ένα αν ο ακέραιος a βρίσκεται στον πίνακα και στην θέση a αλλιώς επιστρέφει μηδέν. Χρησιμοποιούμε την συνάρτηση για να ξέρουμε αν ένα block έχει δημιουργηθεί δηλαδή κάθε φορά που θέλουμε να κάνουμε μία εισαγωγή πρέπει να ξέρουμε αν το συγκεκριμένο block υπάρχει. Οπότε θα έχουμε έναν πίνακα count στον οποίο θα αποθηκεύουμε ποιοι "κάδοι" έχουν φτιαχτεί. π.χ. αν στο αρχείο θέλουμε να έχουμε τέσσερις κάδους και έχουν γίνει μερικές εισαγωγές στο block1 και στο block3 τότε ο πίνακας count θα είναι: 0 1 0 3 0. Έτσι όταν θα θέλουμε να κάνουμε την επόμενη εισαγωγή στον κάδο i θα καλούμε την find και έτσι θα ξέρουμε αν χρειάζεται να δεσμέυσουμε μνήμη για τον καινούριο κάδο ή αν αυτός υπάρχει.

-> Για την HT_InsertEntry:

Αρχικά καλούμε την hash_function για να βρούμε σε ποιον κάδο πρέπει να βάλουμε την εγγραφή.

Ελέγχουμε την περίπτωση που το αρχικό block ή το τελευταίο block υπερχείλισης έχει γεμίσει. Δεσμεύουμε μνήμη για το νέο block υπερχείλισης και κάνουμε insert την πρώτη εγγραφή του. Καλούμε την BF_Block_SetDirty για να δηλώσουμε ότι αλλάξαμε τα δεδομένα οπότε πρέπει να αλλαχτούν και στο δίσκο και κάνουμε UnPin το block. Κάνουμε το count_records ίσο με ένα για το συγκεκριμένο block αφού έχει μόνο μία εγγραφή.
Αυξάνουμε το count_b κατά ένα για το αρχείο αφού φτιάξαμε ένα καινούριο block. Και η δεύτερη στήλη του πίνακα BLOCKS για τον συγκεκριμένο κάδο

δείχνει στον κάδο υπερχείλισης. Και κάνουμε το time_created ίσο με
τον αριθμό των blocks που υπάρχουν στο αρχείο.

Αν ο κάδος δεν έχει φτιαχτεί ακόμα (το ελέγχουμε με την find) αρχικοποιούμε
και δεσμεύουμε μνήμη για το block, βάζουμε στον πίνακα count τον αριθμό
του block και έτσι ξέρουμε για την επόμενη φορά που θα θέλουμε να βάλουμε
εγγραφή σε αυτό το block ότι υπάρχει άρα δεν ξανά δεσμέυουμε μνήμη. Αποθηκεόυμε
την εγγραφή στη θέση μηδέν (είναι η πρώτη εγγραφή του block).
Καλούμε την BF_Block_SetDirty
για να δηλώσουμε ότι αλλάξαμε τα δεδομένα οπότε πρέπει να αλλαχτούν
και στο δίσκο και κάνουμε UnPin το block. Κάνουμε το count_records ίσο με
ένα για το συγκεκριμένο block. Αυξάνουμε τον αριθμό των blocks του αρχείου
κατά ένα.

Στην περίπτωση που υπάρχει το βασικό block αλλά δεν έχει
γεμίσει (δηλαδή δεν έχουν φτιαχτεί ακόμα block υπερχείλισης)
τότε απλά βάζουμε την εγγραφή στην κενή θέση στο block.
Καλούμε την BF_Block_SetDirty για να δηλώσουμε ότι αλλάξαμε τα
δεδομένα οπότε πρέπει να αλλαχτούν και στο δίσκο και κάνουμε UnPin
το block. Αυξάνουμε το count_records κατά ένα για το συγκεκριμένο block.
Επιστρέφουμε σε ποιό block έγινε η εισαγωγή.

Στην περίπτωση που υπάρχει το βασικό block αλλά έχει γεμίσει
και υπάρχει και το block υπερχείλισης και δεν έχει γεμίσει
(δηλαδή ο αριθμός των blocks υπερχείλισης είναι > 0)
τότε απλά βάζουμε την εγγραφή στην κενή θέση του τελευταίου block υπερχείλισης.
Καλούμε την BF_Block_SetDirty για να δηλώσουμε ότι αλλάξαμε τα
δεδομένα οπότε πρέπει να αλλαχτούν και στο δίσκο και κάνουμε UnPin
το block. Αυξάνουμε το count_records κατά ένα για το συγκεκριμένο block.

-> Για την HT_GetAllEntries:

Φτιάχνουμε ένα HT_block που δείχνει στο πρώτο block μετά το block0 (άρα το πρώτο block στο οποίο βάζουμε εγγραφές) και ένα που δείχνει στο πρώτο block υπερχείλισης του bucket που θέλουμε. Αρχικά εξετάζουμε την περίπτωση που η εγγραφή ανήκει στο βασικό "κάδο" δηλαδή αν το id της είναι μικρότερο από τον (αριθμό των κάδων που έχουμε)*(τις max εγγραφές που χωράει ο κάθε κάδος) = (numBuckets*6). Σε αυτή την περίπτωση ελέγχουμε όλες τις εγγραφές του συγκεκριμένου κάδου και ο αριθμός των κάδων που διαβάστηκαν είναι ένας. Αν η εγγραφή ανήκει σε κάποιο block υπερχείλισης ψάχνουμε σε όλα τα block υπερχείλισης εκτός από το τελευταίο. Και στο τέλος ψάχνουμε όλες τις εγγραφές για το τελευταίο block υπερχείλισης. Και επιστρέφουμε τον αριθμό των "κάδων" που διαβάσμε μέχρι να βρούμε την εγγραφή.