Μάθημα : Υλοποίηση Συστημάτων Βάσεων Δεδομένων
Εργασία 3η: External Sort - Εξωτερική Ταξινόμηση
Περίδος : 2017-2018

Ονόματα Συμμετεχόντων	Αριθμός Μητρώου	Email
Γιαλίτσης Νικόλαος	1115201400027	sdi1400027@di.uoa.gr
Κουγιουμτζιάν Λόρι	1115201400076	sdi1400076@di.uoa.gr
Στυλιανού-Νικολαΐδου Σόφία	1115201400195	sdi1400195@di.uoa.gr

Αρχεία:
- sort_file.c
- merge.c
- sort_file.h

Εντολές εκτέλεσης:
- make
- ./build/sr_main1 μόνο μια φορά για να δημιουργήθεί η αρχική βάση
- ./build/sr_main2
- ./build/sr_main3 > output

Γενικά σχόλια για την υλοποίηση:

Ορθότητα:
Το πρόγραμμά μας ταξινομεί τις εγγραφές κατάλληλα για οποιοδήποτε δεκτό bufferSize, αριθμό εγγραφών ή αλγόριθμο αντικατάστασης(LRU,MRU).Ο έλεγχος με Valgrind μας δείχνει ότι δεν υπάρχουν errors ή invalid reads ή invalid writes στον κώδικά μας.

Makefile:
Έχουμε τροποποιήσει το Makefile ώστε να συμπεριλάβουμε αρχικά το δεύτερο αρχείο μας με συναρτήσεις .c αλλά και τα flags -g3 -Wall που χρησιμεύουν για λόγους debugging.

Αλγόριθμοι και Χρόνος:
Η sr_main2 εκτελείται στους υπολογιστές μας σε CPU time : 0.000002+0.033393+0.024689.Η πολυπλοκότητα είναι η επιθυμητή καθώς τηρήσαμε πιστά τους σχετικούς αλγορίθμους ταξινόμησης και συγχώνευσης.

Αρχεία:
temp:Περιέχει τα blocks του buffer
tempOut:Περιέχει τα αποτελέσματα της quicksort και των συγχωνεύσεων

Initialize/Open/Close:

Η SR_Init() περιέχει ενα προαιρετικό τμήμα κώδικα που το χρησιμοποιούμε για τη διευκόλυνση μας.Διαγράφει μέσω system calls τα αρχεία sorted_name sorted_surname και sorted_id ώστε να μην τύχει να γραψουμε τα αποτελέσματα δύο συνεχόμενων εκτέλέσεων στο ίδιο output_file.Ωστόσο, για την εκτέλεση της sr_main3 θα πρέπει να σχολιάσουμε τη γραμμή SR_Init() καθώς δεν θα βρεθούν τα αρχεία από την εκτέλεση της sr_main2.

Η SR_CreateFile() δημιουργεί το αρχέιο fileName και το αρχικοποιεί δημιουργώντας το πρώτο του Block,το οποίο χρησιμεύει σαν αναγνωριστικό του είδους του αρχείου και περιέχει τη συμβολοσειρά "sr_file".

Η access() ελέγχει αν το δεδομένο αρχείο υπάρχει ήδη στον φάκελό μας ενημερώνοντας τη μεταβλητή errno κατάλληλα.Έτσι για τα προσωρινά αρχεία μας,temp,tempOut αλλά και για τα δεδομένα αρχεία output,input ελέγχουμε τη κατάστασή τους ώστε να αποφύγουμε την περίπτωση της κλήσης της create

σε αρχείο που ήδη υπάρχει ή να διαβάσουμε δεδομένα από προηγούμενες εκτελέσεις.Ειδικά στην περίπτωση που το input_filename είναι λάθος τότε τερματίζουμε το πρόγραμμά μας με ανάλογο κωδικό λάθους. Ο κωδικός του errno που δηλώνει ότι το αρχείο δεν βρέθηκε είναι το ENOENT.

Σχεδίασή External Sort:

Φάση Ταξινόμησης Quicksort

Χρησιμοποιούμε ένα προσωρινό αρχείο,το temp ως Buffer,στο οποίο δεσμεύουμε BufferSize blocks.

Στην πρώτη φάση με τη χρήση της GetNextBlocks() διαβάζουμε σε κάθε επανάληψη ,τα επόμενα BufferSize blocks από το input file.Στην περίπτωση που άπομένουν λιγότερα blocks από bufferSize τότε τα θέτουμε 0 και τα αγνοούμε στη συνέχεια.

Καλούμε αναδρομικά τη Quicksort στον Buffer η οποία διαιρεί το input σε δύο μέρη με βάση ένα στοιχείο (pivot) ώστε το pivot να βρίσκεται στη σωστή θέση,τα στοιχεία στο αριστερό τμήμα να είναι μικρότερα ή ίσα του και στο δεξί μεγαλύτερα του.Στη συνέχεια καλύμε τη quicksort σε κάθε τμήμα ξεχωριστά.

Στη SortedFile , πριν τη κλήση της Quicksort, έχουμε υπολογίσει τον συνολικό αριθμό των records, και της δίνουμε ως πρώτο όρισμα το start = 0 , end = Ίast_record_num δηλαδή όλο το εύρος των εγγραφών.Στη partition ακολουθούμε τον γνωστό αλγόριθμο με τη διαφορά ότι τα στοιχεία δεν βρίσκονται σειριακά και έτσι δεν είναι δυνατό να τα προσπελάσουμε με πίνακα πχ. Record[i]. Για αυτόν τον λόγο έχουμε δημιουργήσει μια νέα συνάρτηση , τη GetRecord() ή οποία εντοπίζει το block που περιέχει την εγγραφή με το νούμερο record_num εγγραφή,και επιστρέφει έναν δείκτη στην αρχή της.

Όταν τελειώσει η επανάληψη θα έχουν σχηματιστεί ομάδες μεγέθους BufferSize blocks στο αρχείο tempOut.

Φάση συγχώνευσης Merge:

Σε κάθε εξωτερική επανάληψη ο αριθμός block που περιέχει η κάθε ομάδα πολλαπλασιάζεται κατά (BufferSize-1).
Για παράδειγμα αν το BufferSize είναι 5 τότε στη φάση της αρχικής ταξινόμησης με quicksort θα έχουν φτιαχτεί
ομάδες μεγέθους 5 Blocks.Στην πρώτη εξωτερική επανάληψη της Merge,θα σχηματιστούν ομάδες των 5*4 = 20 blocks καθώς χρησιμοποιούμε BufferSize-1 blocks του Buffer για τη συγχώνευση και το τελευταίο Block για την αποθήκευση του αποτελέσματος.Ομοίως,Στη δέυτερη επανάληψη θα έχουμε ομάδες των 20*4 = 80 bocks,μέχρις ότου σχηματιστεί μία ταξινομημένη ομάδα με όλες τις εγγραφές.

Σε κάθε εσωτερική επανάληψη,καλούμε τη συνάρτηση getNextGroup() η οποία έχει ανάλογη λειτουργία με τη getNextBlocks() που χρησιμοποιήσαμε στη φάση της Quicksort με τη διαφορά ότι δεν φορτώνει τα blocks στον buffer σειριακά , δηλαδή b1,b2,b3 αλλά σε κάθε θέση του buffer μετάφέρει το πρώτο block από την έπόμενη ομάδα-group.Δηλαδή αν έχουμε ομάδες των 4 blocks(και BufferSize = 4) τότε θα έχουμε ομάδες (b1 b2 b3 b4),(b5 b6 b7 b8),(b9 b10 b11 b12).... Η getNextGroup() τότε στη πρώτη επανάληψη θα μεταφέρει στον buffer τα blocks b1,b5,b9 και στη δεύτερη επανάληψη τα b13,b17,b21.

Στη σύνεχεια καλέιται η συνάρτηση Merge() η οποία ενοποιεί τις ομάδες των οποίων τα πρώτα blocks φορτώθηκαν στις Buffer-1 θέσεις του buffer.Η Merge() πραγματοποιεί μια συγχώνευση k-way Merge με k =BufferSize-1 επαναληπτικά μέχρις ότου γραφούν όλα τα records των ομάδων.Χρησιμοποιούμε έναν δυνάμικό δισδιάστατο πίνακα char** data μεγέθους bufferSize-1 όπου σε κάθε θέση του έχει έναν δείχτη στο επόμενο record της κάθε ομάδας.Επίσης,κρατάμε τρείς πίνακες, τους

```
int NumRecs[bufferSize-1];
int Index[bufferSize-1];
int GroupOffset[bufferSize-1];
```

όπου NumRecs είναι ο συνολικός αριθμός των Record που περιέχει το block στο οποίο βρίσκεται το data[*] για κάθε ομάδα.Το χρειαζόμαστε είναι για να ξέρουμε πότε ένα block μιας ομάδας έχει φτάσει στο τέλος του και πότε θα πρέπει να φορτώσουμε το επόμενο block της συγκεκριμένης ομάδας.

Το Index είναι ο αριθμός του record στο block στο οποίο βρίσκεται τη τρέχουσα στιγμή η κάθε ομάδα .

Το GroupOffset είναι ο αριθμός των blocks που έχουν φορτωθεί από τη κάθε ομάδα.

Σε κάθε βήμα εντοπίζεται το min_value και το min_index δηλαδή το record που περιέχει τη μικρότερη τιμή και η θέση του στο data[?].Οι συγκρίσεις γίνονται με τη compare() ανάλογα με το δεδομένο πεδίο. Το min record γράφεται στο output προσωρινά,δηλαδή στο τελευταίο block του Buffer.Επίσης το index της ομάδας που περιείχε το record αυτό αυξάνεται όπως και ο δείχτης της που δείχνει στην επόμενη εγγραφή.Αν τυχόν το block της ομάδας έχει έξαντληθεί καλούμε τη συνάρτηση MoveGroupIndex() που αναθέτει στον δείχτη data[?] της ομάδας, το επόμενο block της.Τέλος σε αυτή την περίπτωση το GroupOffset[?] άυξάνεται κατά ένα.Αν έχουν γραφεί όλες οι εγγραφές από τα μπλοκ μιας ομάδας τότε την αγνοούμε.Έχουμε εξετάσει και την περίπτωση που τερματίζει η ομάδα που κρατούσε το τελευταίο min οπότε θα πρέπει εκείνο να επαναρχικοποιηθεί κατάλληλα.

Αν το output,δηλαδή το τελευταίο Block του buffer γεμίσει από ταξινομημένα records , τα γράφουμε στο output_filename είτε μέσω της InsertEntry είτε μέσω της InsertOutput.Η διαφορά είναι ότι η InsertEntry καλείται μόνο στην πρώτη εξωτερική επανάληψη της Merge όπου το αρχείο output δεν περιέχει κανένα block και έτσι θα πρέπει να κάνουμε όσα Allocate χρειάζονται.Από τη δεύτερη επανάληψη και μετά καλούμε την InsertOutput η οποία κάνει αντικατάσταση των εγγραφών από τη προηγούμενη επανάληψη χωρίς να δεσμεύσει παραπάνω blocks στο output_file.Ο διαχωρισμός αυτός γίνεται μέσω της μεταβλητής output_has_content που περνιέται ως όρισμα απο τη SortedFile.

Αν όλες οι ομάδες έχουν τερματίσει,τότε γράφουμε και τις εναπομείναντες εγγραφές από το output στο αρχείο εξόδου.

Στο τέλος της κάθε συγχώνευσης αντιγράφουμε στο tempOut τα αποτελέσματα της προηγούμενης επανάληψης ώστε να ταξινομηθούν επαναληπτικά.Η αντιγραφή γίνεται με τη συνάρτηση CopyContent().