Λειτουργικά Συστήματα
Εργασία 2
2020-2021

Ονοματεπώνυμο: *Δημητρακόπουλος Κωνσταντίνος*
ΑΜ: *1115201500034*

Τα αρχεία που δίνονται μεταγλωτίζονται και συνδέονται με την εντολή:

make

και το εκτελέσιμο simulator εκτελείται με την εντολή:

./simulator algorithm numframes q numtraces

όπου:

- στο πεδίο algorithm εισάγεται η συμβολοσειρά **lru** ή **secondChance**
- στο πεδίο numframes εισάγεται ένας θετικός αριθμός που αναπαριστά το **πλήθος των frames**
- στο πεδίο q εισάγεται ένας θετικός αριθμός που αναπαριστά το **q** βάσει της εκφώνησης
- στο πεδίο numtraces εισάγεται ένας αριθμός που αναπαριστά το **πλήθος των αναφορών** που θα εξεταστούν (-1 για να εξεταστούν όλες οι αναφορές).

Μία **ενδεικτική εκτέλεση** είναι η παρακάτω:

./simulator lru 100 250 20000

Τα δυο hash tables που ζητουνται αποτελούνται απο ένα πίνακα δεικτών σε λίστες με κόμβους τύπου **hashnode**.

Σε κάθε τέτοιο κόμβο αποθηκεύεται το **frame** (θετικος αριθμος απο 0 εως numframes), η **σελίδα** όπως λήφθηκε απο την εκάστοτε αναφορά, το **reference bit** όπου απασχολεί τον αλγόριθμο αντικατάστασης second Chance και το **LRUcount** όπου είναι ο εκάστοτε μετρητής που χρησιμοποιείται στον αλγόριθμο αντικατάστασης LRU.

Η κύρια μνήμη αναπαρίσταται απο ένα πίνακα απο μεταβλητές τύπου **memoryFrame** (με πλήθος στοιχείων numframes).

Σε κάθε τέτοια δομή αποθηκεύεται ένας **δείκτης προς το hashnode** που αναπαριστά τη σελίδα στο hash η οποία είναι αποθηκευμένη στο συγκεκριμένο frame, τη μεταβλητή **dirty bit** που μεταβάλλεται ανάλογα έτσι ώστε η σελίδα να επαναγγραφεί στο δίσκο κατα την αφαίρεση της απο τη μνήμη, και ένα **δείκτη προς το hash** στο οποίο είναι αποθηκευμένη η σελίδα του συγκεκριμένου frame.

Simulator.c:

Ο δείκτης σε συνάρτηση memory_replacementAlgorithm() θα φιλοξενήσει κατάλληλα τη συνάρτηση LRU ή second chance.

Αρχικά ελέγχονται τα ορίσματα του χρήστη για ορθότητα, γίνεται αρχικοποίηση των hash tables και του πίνακα memoryStructure που αντιπροσωπεύει τη κύρια μνήμη και ανοίγονται τα αρχεία αναφορών.

Στη συνέχεια λαμβάνουμε αναφορές απο το αρχείο bzip.trace q φορές. Μετά την ανάγνωση της εκάστοτε αναφοράς, ελέγχεται αν πρέπει μηδενίσουμε το reference bit των ενεργών σελίδων αν ο αλγόριθμος αντικατάστασης είναι ο second chance και έχει επέλθει η ώρα μηδενισμού-ολίσθησης (συμβαίνει κάθε numframes/3 αναφορές).

Λαμβάνουμε τα 20 αριστερά bit της λογικής διεύθυνσης ως την σελίδα και τα 12 δεξιά ως την παράμετρο μετατόπισης, αφού η σελίδα αποτελείται απο 4096 bytes και 2^12=4096.

Έπειτα ελέγχεται αν η σελίδα είναι στο bzip hash μέσω της hash_searchPage. Στην περίπτωση που δεν υπάρχει, εισάγεται, αυξάνεται ο μετρητής pageFault αφου βρισκόμαστε σε κατάσταση page fault και αυξανεται ο μετρητής readCount αφού θα χρειαστεί να διαβάσουμε τη σελίδα απο το δίσκο.

Κατα τη συνάρτηση memory_replacementAlgorithm εκτελείται ο αλγόριθμος αντικατάστασης, ο οποίος δέχεται δείκτη σε hashnode όπου αναπαριστά τη σελίδα και δείκτη σε hash όπου περιέχει το συγκεκριμένο hashnode. Επιστρέφοντας απο τη συνάρτηση οι παραπάνω μεταβλητές αποτυπώνουν τη σελίδα προς διαγραφή σε περίπτωση που υπήρξε αντικατάσταση της σελίδας σε πλαίσιο μνήμης.

Αν η σελίδα αυτή υπάρχει αφαιρείται απο το hash ενώ αν χρειαστει να γραφει στο δίσκο ο μετρητής writeCount ενημερώνεται. Τελικά αποθηκεύεται ο αριθμος του frame στο hashnode.

Σε περίπτωση που η hash_searchPage() βρηκε την σελίδα στο hash τότε απλά μεταβάλλεται το dirty bit του πλαισίου που κρατά τη σελίδα αν αυτό κριθεί απαραίτητο.

Η ίδια ακριβώς διαδικασία ακολουθείται για q λογικές διαευθύνσεις απο το gcc.trace αρχείο. Η διαδικασία ολοκληρώνεται όταν ληφθεί ο απαραίτητος αριθμός αναφορών. Τελικά γράφονται στο δίσκο οι σελίδες με dirty bit = 1, αποδεσμεύονται τα hash tables, κλείνουν τα αρχεία αναφορών και τυπώνονται τα τελικά αποτελέσματα.

PageTable.c:

hash_Initialize(): Δεσμεύει, αρχικοποιεί και επιστρέφει ένα hash table απο hashnode*.

hash_function(): hash function που επιστρέφει το υπόλοιπο της διαίρεσης αριθμού σελίδας και πλήθους buckets.

SearchPage(): Αναζητεί τη σελίδα στο hash που δίνεται. Αν τη βρει, μεταβάλλει το reference bit και το LRUcounter για τους αλγορίθμους αντικατάστασης και επιστρέφει τον αριθμό του frame. Σε περίπτωση που δε βρεθεί, δεσμεύει χώρο,αρχικοποιεί και συνδέει ένα hashnode που θα αναπαριστά αυτή τη σελίδα, ενώ επιστρέφει -1.

hash_removePage(): Αφαιρεί τη σελίδα που δίνεται απο το hash που δίνεται.

hash_destroy(): Αποδέσμευση του χώρου του δοθέντος hash.

Λοιπές βασικές συναρτήσεις που αφορούν δομές δεδομένων δεν αναγράφονται για την εξοικονόμηση χρόνου του αναγνώστη, παρόλα αυτα σχολιάζονται επαρκώς.

MemoryStructure.c:

memory_setDirtyBit(): μεταβάλλει το dirty bit στην τιμή 1 σε περίπτωση που η τιμή του είναι διάφορη του 1 και η αναφορά είναι τύπου 'W'.

memory_lru(): Αρχικά υπολογίζεται η σελίδα προς αφαίρεση όπου έχει τη μικρότερη LRUCount τιμή. Σε περίπτωση που η εγγραφή αυτή πρέπει να γραφει στο δίσκο ενημερώνεται ο κατάλληλος μετρητής. Ανταλλάσονται στις μεταβλητές ορίσματα και στα περιεχόμενα του παλισίου οι δείκτες προς τις σελίδες και το hash στο οποίο ανήκουν. Τελικά γράφεται κατάλληλα το dirty bit.

memory_secondChance(): Η μεταβλητή index μεταβάλλεται συνεχώς στο πεδίο [0,numframes). Αυξάνοντας κάθε φορά την τιμή της, αναζητεί μια σελίδα όπου το reference bit είναι 0, ενώ όσες επισκέυθηκε έλλαξε το reference bit σε 0. Τελικά όταν βρεθεί το κατάλληλο πλαίσιο η σελίδα γράφεται στο δίσκο (αν χρειάζεται) και αντικαθίσταται κατάλληλα.

memory_referenceBitRefresh(): Θέτει όλα τα reference bit των σελίδων στα υπάρχοντα πλαίσια ίσα με 0.

memory_getActiveDirtyBitCount(): Επιστρέφει το πλήθος των σελίδων στα υπάρχονται πλαίσια όπου έχουν dirty bit ίσο με 1.