Λειτουργικά Συστήματα
Εργασία 2

Αθανασία Τουρνάκη
AM:1115201600172

Σε αυτήν την εργασία υλοποιήθηκε μία προσομοίωση ενός συστήματος διαχείρισης μνήμης βάσει πραγματικού ίχνους αναφορών με χρήση αρχείων. Ολοκληρώθηκαν όλα τα ζητούμενα της άσκησης. Ο χρήστης κατά την εκτέλεση του προγράμματος καλείται να δώσει σαν ορίσματα την γραμμή εντολών τον μέγιστο αριθμό σφαλμάτων σελίδας που μπορεί να εμφανίσει μία διεργασία(k), τον συνολικό αριθμό πλαισίων της κύριας μνήμης(frames), το πλήθος των ιχνών που θα διαβάζεται από κάθε διεργασία σε κάθε επανάληψη(q) και προαιρετικά τον συνολικό αριθμό ιχνών που θα διαβαστούν από το αρχείο(max). Σε περίπτωση που ο χρήστης δεν δώσει ένα πλήθος max, αυτό αρχικοποιείται σε 1,000,000, όσο είναι δηλαδή το πλήθος των ιχνών σε κάθε αρχείο από αυτά που δίνονται από την εκφώνηση της άσκησης.

Δημιουργήθηκαν οι **τρεις διεργασίες** που ζητούνται(PM1, PM2 και MM), με τις PM1, PM2 να διαβάζουν το αρχείο με τα ίχνη αναφορών και μεταβιβάζουν τα τις αναφορές στην διεργασία MM η οποία λειτουργεί ως εικονική μνήμα με την χρήση ενός πίνακα κατακερματισμού.

Η επικοινωνία μεταξύ καθεμίας από τις διεργασίες PM1, PM2 και του διαχειριστή μνήμης MM επιτυγχάνεται με την **χρήση ενός τμήματος διαμοιραζόμενης μνήμης δύο θέσεων** που η κάθε θέση έχει χώρο για έναν πίνακα μεγέθους q που αποθηκεύει δομές τύπου trace. Επίσης, για τον συγχρονισμό των διεργασιών χρησιμοποιήθηκαν **6 σημαφόροι**, από 3 για κάθε θέση στην κοινή μνήμη για την πραγματοποίηση ανταλλαγής δεδομένων με βάση το μοντέλο Producer-Consumer.

Το πρόγραμμα έχει διασπαστεί σε πολλά διαφορετικά αρχεία, το καθένα με διαφορετικό περιεχόμενο και λειτουργία.

Πιο αναλυτικά:

- **Sem_shm_fun.c:** Πρόκειται για το ίδιο αρχείο που χρησιμοποιήθηκε και για την Εργασία 1. Περιέχει συναρτήσεις για δημιουργία, ανάκτηση και διαγραφή σημαφόρων και τμημάτων διαμοιραζόμενης μνήμης με τη χρήση αρχείων. Για την υλοποίηση των συναρτήσεων αυτών έγινε χρήση των σημειώσεων του εργαστηριού 2 του μαθήματος και συγκεκριμένα του κώδικα 4.

- **Semaphores.c:** Πρόκειται για το ίδιο αρχείο που χρησιμοποιήθηκε και για την Εργασία 1. Περιέχει συναρτήσεις για αρχικοποίηση και τροποποίηση σημαφόρων. Για την υλοποίηση των συναρτήσεων αυτών έγινε χρήση των σημειώσεων του εργαστηρίου 2 του μαθήματος και συγκεκριμένα του κώδικα 3.

- **list.c:** Σε αυτό το αρχείο περιέχονται συναρτήσεις για την διαχείριση, όπως απαιτείται από την εργασία, μίας απλά συνδεδεμένης λίστα με δύο δείκτες για αρχή και τέλος. Η δομή list που αποθηκεύει στιγμιότυπα δομής node που ορίζονται στο αντίστοιχο αρχείο list.h χρησιμοποιούνται για την αναπαράσταση των λιστών αναφορών που δημιουργούνται κατά την εισαγωγή εγγραφών στον πίνακα κατακερματισμού με το node να περιέχει πληροφορίες για την αναφορά που προστίθεται κάθε φορά, δηλαδή τον αριθμό της σελίδας και το εάν αυτή είναι τροποποιημένη ή όχι, καθώς επίσης και το ποια διεργασία οφείλεται για την αναφορά. Αναλυτικά για κάθε συνάρτηση:
  - **Void listInit(list* List):** Συνάρτηση για αρχικοποίηση της λίστας
  - **node* inList(list* List, int PageNo, int processIndex):** Συνάρτηση που ελέγχει εάν μέσα στη λίστα υπάρχει node που περιέχει το PageNo και έχει δημιουργηθεί από την διεργασία processIndex.

- **Int listInsert(list* List, int pageNo, char command, int processIndex):** Συνάρτηση για την εισαγωγή εγγραφών. Γίνεται όλη η διαδικασία για την εγγραφή σε μία λίστα. Αν υπάρχει στην λίστα ήδη node με το ίδιο processIndex και το ίδιο pageNo, αυτό δεν ξαναπροστίθεται. Αν η εντολή που δόθηκε για το command ειναι R τότε το πεδίο dirty του node αρχικοποιείται σε μηδέν, αλλιώς σε 1. Αν προστεθεί το αντικείμενο στη λίστα επιστρέφει 1, αλλιώς 0.
- **int listDeleteAll(list* List, int* count):** Συνάρτηση για την διαγραφή όλων των εγγραφών της λίστας. Επιστρέφει τον αριθμό των εγγραφών που διαγράφηκαν και των οποίων το πεδίο dirty ήταν 1(τα writes στον δίσκο), ενώ αποθηκεύει στην μεταβλητή count τον αριθμό των εγγραφών που διαγράφηκαν.
- **Int listDeleteProcessEntries(list* List, int* count, int processIndex):** Συνάρτηση που χρησιμοποιείται κατά την εφαρμογή του αλγορίθμου FlushWhenFull(FWF) παίρνοντας σαν όρισμα τον αριθμό της διεργασίας PM που προκάλεσε την κλήση του και διαγράφοντας μόνο τις αναφορές που προστέθηκαν στον πίνακα κατακερματισμού από αυτή την διεργασία. Επιστρέφει τον αριθμό των εγγραφών που διαγράφηκαν και των οποίων το πεδίο dirty ήταν 1(τα writes στον δίσκο), ενώ αποθηκεύει στην μεταβλητή count τον αριθμό των εγγραφών που διαγράφηκαν.

◆ **HashTable.c:** Σε αυτό το αρχείο περιέχονται συναρτήσεις για την διαχείριση της δομής hashTable, όπως αυτή ορίζεται στο αντίστοιχο αρχείο hashTable.h, καθώς και για την δημιουργία ενός στιγμιότυπου δομής trace(ίχνους αναφοράς). Αναλυτικά για κάθε συνάρτηση:
- **void createTrace(struct refTrace* trace, unsigned int pageNo, char command):** Συνάρτηση δημιουργίας και αρχικοποίησης εγγραφής
- **unsigned int hashFunction(unsigned int pageNo, int numBuckets):** Universal Hash Function.
- **hashTable* hashCreate(int numbuckets):** Συνάρτηση δημιουργία και αρχικοποίηση του πίνακα κατακερματισμού.
- **int hashInsert(refTrace trace, hashTable* HT, int processIndex):** Συνάρτηση για την εισαγωγή μιας εγγραφής στον πίνακα κατακερματισμού. Ανάλογα με την τιμή πυ θα επιστρέψει η Hash Function προστίθεται στην λίστα του αντίστοιχου bucket μια γγραφή. Αν η listInsert επιστρέψει 1 αυξάνεται ο αριθμός των reads από τον δίσκο και ο συνολικός αριθμός εγγραφών στον πίνακα.
- **void hashEmpty(hashTable* HT, int processIndex):** Συνάρτηση που χρησιμοποιείται κατά την εφαρμογή του αλγορίθμου FWF. Διαγράφει όλες τις εγγραφές που δημιουργήθηκαν από μία διεργασία. Ανάλογα με τον αριθμό που επιστρέφει η listDeleteProcessEntries αυξάνεται ο αιρθμός των writes στον δίσκο.
- **void hashDelete(hashTable* HT):** Συνάρτηση διαγραφής του πίνακα κατακερματισμού.

◆ **Functions.c:** Σε αυτό το αρχείο περιέχονται οι συναρτήσεις που εκτελούνται από κάθε διεργασία. Αναλυτικά για καθεμία από τις συναρτήσεις αυτές:
- **int PM(int index, int q, int max):** Η συνάρτηση αυτή χρησιμοποιείται από τις πρώτες δύο διεργασίες PM1 και PM2 και λειτουργεί ανάλογα για την καθεμία με βάση το index(1 ή 2) που της δίνεται σαν όρισμα. Συγκεκριμένα, αφού ανακτήσει το τμήμα διαμοιραζομενης μνήμης και τους σημαφόρους, επισυνάπτει στο τμήμα που ανακτήθηκε έναν πίνακα μεγέθους 2 * q που κρατάει δομές τύπου refTrace. Έπειτα, ανοίγει(ανάλογα με το ποια διεργασία την καλεί) είτε το αρχείο gcc.trace, είτε το bzip.trace και αναθέτει σε εναν δείκτη που δείχνει στο τμήμα μνήμης είτε την πρώτη θέση του πίνακα(αν έχει καλέσει τη συνάρτηση η διεργασία 1), είτε την q-οστή(αν έχει καλέσει την συνάρτηση η διεργασία 2). Μέχρι τα ίχνη που διαβάζονται από το αρχείο να είναι ίσα με max η συνάρτηση λειτουργεί σαν το κομμάτι Producer στο μοντέλο Producer-Consumer και για όσο τα ίχνη που διαβάζονται από το αρχείο σε αυτήν την επανάληψη είναι λιγότερα από q και το αρχείο δεν έχει τελειώσει δημιουγεί την εγγραφή που αντιστοιχεί στο εκάστοτε trace και την προσθέτει στην κοινή μνήμη.
- **int MM(int q, int k, int frames, int max):** Η συνάρτηση αυτή χρησιμοποιείται από την διεργασία που υλοποιεί το MM. Αφού ανακτήσει το τμήμα διαμοιραζόμενης μνήμης και τους

σημαφόρους, αρχικοποιεί κάποιες μεταβλητές για τη διαχείριση του πίνακα κατακερματισμού και την σύλλεξη στατιστικών στοιχείων. Έπειτα, εκτελεί το κομμάτι Consumer του μοντέλου Producer-Consumer εναλλάξ για q αναφορές από κάθε διεργασία και εισάγει την εγγραφές που αντλούνται από την κοινή μνήμη στον πίνακα κατακερματισμού. Όσο ο αριθμός των Page Faults που προκύπτουν είναι μικρότερος από k, η εισαγωγή γίνεται κανονικά. Μόλις όμως φτάσει στο k-οστό στοιχείο, η συνάρτηςη χησιμοποιεί την universal hash function για να προσδιορίσει το bucket στο οποίο θα προστεθεί η εγγραφή και ελέγχει αν αυτή υπάρχει ήδη. Αν αυτή δε βρεθεί σημαίνει ότι θα προκληθεί Page Fault κατά την εισαγωγή της, οπότε η συνάρτηση χρησιμοποιεί τον αλγόριθμο FWF, δηλαδή πριν προσθέσει τη νέα εγγραφή διαγράφει όλες όσες έχουν δημιουργηθεί από την διεργασία που την δημιούργησε. Τέλος, η συνάρτηση εκτυπώνει τα στατιστικά που έχουν συλλεγεί κατά την διάρκεια της εκτέλεσης του προγράμματος, πριν και μετά διακοπή χρήσης της κύριας μνήμης και διαγράφει το Hash Table.

- **main.c:** Αποθηκεύει σε μεταβλητές το ορίσματα που δίνει ο χρήστης, δημιουργεί σημαφόρους και τμήμα διαχειριζόμενης μνήμης και έπειτα δημιουργεί τις διεργασίες PM1, PM2, MM, περιμένει να τελειώσουν, διαγράφει σημαφόρους/κοινή μνήμη και τερματίζει.

<u>Οδηγίες για την εκτέλεση:</u>

Έχει δημιουργηθεί makefile, οπότε γίνεται χρήση των εντολών make και make clean για την μεταγλώττιση και την διαγραφή των αρχείων αντίστοιχα. Μετά την μεταγλώττιση ο χρήστης καλείται να πληκτρολογήσει **./Project <k> <αριθμός frames> <q> <max αριθμός ιχνών που θα διαβαστεί(προαιρετικό)>**

<u>Αποτελέσματα διαφορετικών εκτελέσεων:</u>

1. **./Project 500 1000 10 10000**

Statistics before the Hash Table is Deleted:

Total Disk Reads: 1038
Total Disk Writes: 436
Total Page Faults: 1038
Total Traces Examined: 20000
Maximum Number of Frames in Use at Once: 719

Statistics after the Hash Table is Deleted:

Total Disk Reads: 1038
Total Disk Writes: 636
Total Page Faults: 1038
Total Traces Examined: 20000
Maximum Number of Frames in Use at Once: 719

2. **./Project 250 1000 10 10000**

Statistics before the Hash Table is Deleted:

Total Disk Reads: 1347
Total Disk Writes: 745
Total Page Faults: 1347
Total Traces Examined: 20000
Maximum Number of Frames in Use at Once: 478

Statistics after the Hash Table is Deleted:

Total Disk Reads: 1347
Total Disk Writes: 868
Total Page Faults: 1347
Total Traces Examined: 20000
Maximum Number of Frames in Use at Once: 478

3. **./Project 50 1000 10 10000**

Statistics before the Hash Table is Deleted:

Total Disk Reads: 2296
Total Disk Writes: 1117
Total Page Faults: 2296
Total Traces Examined: 20000
Maximum Number of Frames in Use at Once: 100

Statistics after the Hash Table is Deleted:

Total Disk Reads: 2296
Total Disk Writes: 1133
Total Page Faults: 2296
Total Traces Examined: 20000
Maximum Number of Frames in Use at Once: 100

Μέσα από αυτά τα παραδείγματα γίνεται προφανές ότι όσο ο αριθμός k είναι πλησίον του frames/2 ο αριθμός των αναγνωσεων από τον δίσκο(άρα και των Page Faults), αλλά και των εγγραφών σε αυτών είναι μικρότερος. Επίσης, παρατηρείται καλύτερη αξιοποίηση της μνήμης με την χρήση μεγαλύτερου χώρου για την αποθήκευση των εγγραφών.