Λειτουργικά Συστήματα
(Εργασία 2η)

Ονοματεπώνυμο: Δημήτριος Σιταράς
Αριθμός μητρώου: 1115201800178
Εξάμηνο: 5ο

Μεταγλώττιση του προγράμματος: στον αρχικό κατάλογο “os_ask2_sdi1800178” πληκτρολογούμε στο τερματικό την εντολή make (αν θελουμε να διαγραψουμε τα αρχεία που παραγονται απλά πληκτρολογούμε make clean) και παραγεται το εκτελέσιμο αρχείο “simulator”.

Εκτέλεση του προγράμματος: στον αρχικό κατάλογο “os_ask2_sdi1800178” γράφουμε το όνομα του εκτελεσιμου αρχειου ακολουθούμενο με τα κατάλληλα ορίσματα. Πιο συγκεκριμένα γράφουμε:
./simulator “lru/second_chance” “frames_number” “references_per_process” “maximum_references”
Δηλαδή, ο χρήστης δίνει ως ορίσματα: τον αλγόριθμο αντικατάστασης των σελίδων, τον αριθμό πλαισίων στην μνήμη, το πλήθος των αναφορών q και το μέγιστο αριθμός των αναφορών που θα εξεταστούν απο τα αντίστοιχα αρχεία ίχνους (αν δεν θέλει ο χρήστης να προσδιορίσει τον μέγιστο αριθμό των αναφορών “δίνει” στην θέση του την τιμή -1).
Παραθέτω μερικά παραδείγματα εκτέλεσης του προγράμματος:
./simulator lru 100 250 1000
./simulator second_chance 10 20 100
./simulator lru 20 5 -1

Γενικά σχόλια:

- Ο κώδικας είναι πλήρως σχολιασμένος.

- Η μνήμη που δεσμεύεται δυναμικά κατα την εκτέλεση του προγράμματος, αποδεσμεύεται πλήρως και σωστά.
  (Έχει ελεγχθεί με την χρήση του valgrind στα μηχανήματα linux της σχολής).

Δομή Αρχείων:

Στον φάκελο “traces” βρίσκονται τα αρχεία ίχνους: bzip.trace και gcc.trace.
Στους φακέλους “hash_table” και “list” βρίσκονται οι υλοποιήσεις του κατακερματισμένου (hashed) πίνακα σελίδων και της μονά συνδεδεμένης λίστας αντίστοιχα.
Στον φάκελο “replacement_algorithms” βρίσκονται οι υλοποιήσεις των αλγορίθμων αντικατάστασης σελίδας που ζητούνται: Least Recently Used (LRU) και Second Chance.
Στον φάκελο “memory” βρίσκεται η υλοποίηση της λειτουργίας της κύριας μνήμης.

Υλοποίηση και Λειτουργία:

Έχω 2 (εικονικές) διεργασίες, όπου η κάθε μια έχει τον δικό της κατακερματισμένο (hashed) πίνακα σελίδων και “διαβάζουν” αναφορές από δύο διαφορετικά αρχεία ίχνους (η “1η διεργασία” διαβάζει από το αρχείο ίχνους bzip.trace και η “2η διεργασία” από το αρχείο ίχνους gcc.trace). Έτσι, κάθε φορά από μια “διεργασία” διαβάζεται συγκεκριμένος αριθμός αναφορών q από το αρχείο ίχνους του. Έτσι, αφού έχει προηγηθεί η αρχικοποίηση κύριας μνήμης,για κάθε μια αναφορά γίνεται προσπάθεια ανάκτησης της αντίστοιχης σελίδας από την εικονική μνήμη. Επομένως, στην αρχή η σελίδα (page) αναζητείται στο κατακερματισμένο (hashed) πίνακα σελίδων και σε περίπτωση που βρεθεί, ενημερώνονται οι αντίστοιχες πληροφορίες που την συνοδεύουν. Αντίθετα, η σελίδα (page) πρέπει να φορτωθεί απο τον δίσκο, οπότε έχω σφάλμα σελίδας (Page Fault). Αν υπάρχει κενό πλαίσιο (frame), τότε αυτό δεσμεύεται και η σελίδα (page) φορτώνεται στον κατακερματισμένο πίνακα σελίδων. Αν όμως δεν υπάρχει ελεύθερο πλαίσιο (frame), δηλαδή η μνήμη είναι ήδη κορεσμένη, τότε μέσω ενός αλγορίθμου αντικατάστασης σελίδων που ορίζεται στην γραμμη εντολων (lru/second chance) “αποφασίζεται” ποια σελίδα (page) από την μνήμη πρέπει να απομακρυνθεί, ώστε η νέα σελίδα (page) να φορτωθεί από τον δίσκο στο κατακερματισμένο (hashed) πίνακα σελίδων στο αντίστοιχο πλαίσιο (frame) που απελευθερώθηκε.
Εάν οι σελίδες που απομακρύνονται έχουν τροποποιηθεί (το dirty bit ειναι 1, δηλαδή έχει γίνει write σε κάποια αναφορά τους) τότε “γράφονται” πάλι πίσω στον σκληρό δίσκο.

Έχω υλοποιήσει τον αλγόριθμο Least Recently Used (LRU) με την χρήση μετρητή (χρόνου). Συγκεκριμένα, χρησιμοποιώ ως μετρητή μια μήτρα χρόνου, έτσι κάθε φορά που γίνεται αναφορά σε μια σελίδα (page) αποθηκεύεται μια καταχώρηση με βάση τον πραγματικό χρόνο.
Επομένως, για κάθε σελίδα (page) που βρίσκεται στο κατακερματισμένο πίνακα σελίδων έχω αποθηκεύσει την χρονική στιγμή της τελευταίας αναφοράς της. Οπότε ο αλγόριθμος LRU επιλέγει να αντικατασταθει απο την κύρια μνήμη η σελίδα (page) στην οποία δεν έγινε αναφορά για το μεγαλύτερο χρονικό διάστημα.

Έχω υλοποιήσει τον αλγόριθμο Second Chance με την προσθήκη ενός bit αναφοράς (reference bit) σε κάθε πλαίσιο (frame) μνήμης. Κάθε φορά που γίνεται αναφορά σε μια σελίδα (page) που βρίσκεται στον κατακερματισμένο πίνακα σελίδων το bit αναφοράς αλλάζει από 0 σε 1. Όταν το bit αναφοράς γίνει 1, δίνεται στην σελίδα (page) που έχει φορτωθεί στο αντίστοιχο πλαίσιο (frame) μια “δεύτερη ευκαιρία”. Συνεπώς, ο αλγόριθμος Second Chance αρχίζει κάθε φορά από την πιο “παλιά” σελίδα (page) και συνεχίζει έως ότου βρεί την πρώτη σελίδα (page) που έχει bit αναφοράς ίσο με 0 (αν το bit αναφοράς της σελίδας είναι 1 τότε αυτό γίνεται 0, δίνοντας μια δεύτερη ευκαιρία, και ο αλγόριθμος συνεχίζει με την αμέσως επόμενη σελίδα), την οποία επιλέγει να αντικατασταθεί από την κύρια μνήμη.

Διευκρινίσεις:

- Όλες οι σελίδες (pages) και τα πλαίσια (frames) έχουν μέγεθος 4096 bytes. Επομένως, το offset ειναι $log_2(4096) = 12$ bits.

- Οι λογικές διευθύνσεις που δίνονται από αρχεία αναφορών είναι των 32 bits. Επομένως, αφού τα τελευταία 12 bits είναι το offset, τα πρώτα 20 bits αποτελούν τον αριθμό σελίδας (page number).

- Στην αρχή, όλες οι σελίδες βρίσκονται στον σκληρό δίσκο.

- Η εικονική μνήμη αναπαριστάται από έναν κατακερματισμένο (hashed) πίνακα σελίδων.

- Οι κατακερματισμένοι πίνακες σελίδων που δημιουργουνται έχουν 10 buckets ο καθένας (το έχω ορίζει ως σταθερά “const int buckets=10; “)
- Έχω εφαρμόσει την πολιτική του Global replacement που ζητήθηκε στο μάθημα, δηλαδή η σελίδα θύμα επιλέγεται από όλες τις διεργασίες που βρίσκονται στην κύρια μνήμη.

- Μετά το πέρας της διαδικασίας της ανάκτησης σελίδων από την μνήμη, εκτυπώνονται τα στατιστικά στοιχεια της προσομοίωσης αυτης (ακριβώς αυτά που ζητούνται στην εκφώνηση της άσκησης).

- Οι δύο διεργασίες είναι εικονικές.