ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών

«ΠΛΗΡΟΦΟΡΙΚΗ»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Μελέτη αλγορίθμων αντικατάστασης σελίδας με πραγματικά δεδομένα** **Study of page replacement algorithms with real data**
Ονοματεπώνυμο Φοιτητή	**Ιωάννης Καμπυλαυκάς**
Πατρώνυμο	**Ηλίας**
Αριθμός Μητρώου	**ΜΠΠΛ18024**
Επιβλέπων	**Ιωάννης Ε. Βενέτης, Επίκουρος Καθηγητής**

Ημερομηνία Παράδοσης **Οκτώβριος 2023**

Τριμελής Εξεταστική Επιτροπή

Ιωάννης Ε. Βενέτης Επίκουρος Καθηγητής	Χαράλαμπος Κωνσταντόπουλος Καθηγητής	Ιωάννης Τασούλας Επίκουρος Καθηγητής

Περίληψη

Η παρούσα μεταπτυχιακή διατριβή αφορά στους αλγορίθμους αντικατάστασης σελίδας και πιο συγκεκριμένα, στην ανάλυση της συμπεριφοράς εφαρμογών ως προς τα σφάλματα σελίδων που κάνουν. Αρχικά, χρησιμοποιήθηκε ένα εργαλείο, το οποίο κάνει ιχνηλάτιση (trace) τις αναφορές ενός προγράμματος στη μνήμη, για τη συλλογή και ανάλυση πραγματικών δεδομένων.

Το εργαλείο αυτό είναι το PIN της Intel. Με τη χρήση αυτού του εργαλείου, είναι δυνατή η εκτέλεση διαφόρων, έτοιμων μετροπρογραμμάτων (benchmarks) και η καταγραφή των θέσεων μνήμης που χρησιμοποιούν. Κατά συνέπεια, είναι δυνατή και η εύρεση της ακολουθίας αναφορών σε σελίδες.

Στη συνέχεια, υλοποιήθηκε ένα εργαλείο σε γλώσσα προγραμματισμού C, το οποίο βρίσκει ποια και πόσα σφάλματα σελίδας συμβαίνουν με βάση την ακολουθία αναφορών, για διαφορετικούς αλγορίθμους αντικατάστασης σελίδας και διαφορετικό αριθμό πλαισίων σελίδας (page frames). Τέλος, αναλύθηκε η συμπεριφορά του κάθε αλγορίθμου αντικατάστασης σελίδας, βάσει του αριθμού σφαλμάτων σελίδας του καθενός, μετά την εκτέλεση του κάθε benchmark ξεχωριστά.

Abstract

The paper presented is related to page replacement algorithms and, more specifically, the analysis of applications' behaviour as to the page faults they generate. A ready tool was initially used, which generates program traces in memory, for the collection and analysis of real data.

The tool mentioned is the Intel PIN. By using this tool, we can execute several benchmarks and record the memory positions they use. As a result, it is possible to find the reference sequence in the form of pages. The following step was to implement another tool, using the C programming language, that finds which page faults take place, as well as their total number, according to the reference sequence, for different page replacement algorithms as well as page frames. Finally, the behaviour of each algorithm was analysed, according to its total number of page faults, after the execution of each benchmark separately.

Περιεχόμενα

1 ΕΙΣΑΓΩΓΗ

1.1 Αντικείμενο της μεταπτυχιακής διατριβής

Το αντικείμενο της παρούσας εργασίας είναι η ανάλυση της συμπεριφοράς διαφόρων προγραμμάτων, βάσει των σφαλμάτων σελίδας, στα οποία υποπίπτουν. Η εν λόγω ανάλυση γίνεται μέσω της μελέτης συγκεκριμένων αλγορίθμων αντικατάστασης σελίδας (page replacement algorithms), με χρήση πραγματικών δεδομένων για τις προαναφερθείσες εφαρμογές.

1.2 Σκοπός και στόχοι

Ο σκοπός της εν λόγω μεταπτυχιακής διατριβής είναι η εξαγωγή πληροφοριών και συμπερασμάτων για τη συμπεριφορά διαφορετικών αλγορίθμων αντικατάστασης σελίδας.

Συγκεκριμένα, θέλουμε να εξετάσουμε ποιός από τους αλγορίθμους αντικατάστασης σελίδας που χρησιμοποιήσαμε παράγει τα λιγότερα σφάλματα σελίδας.

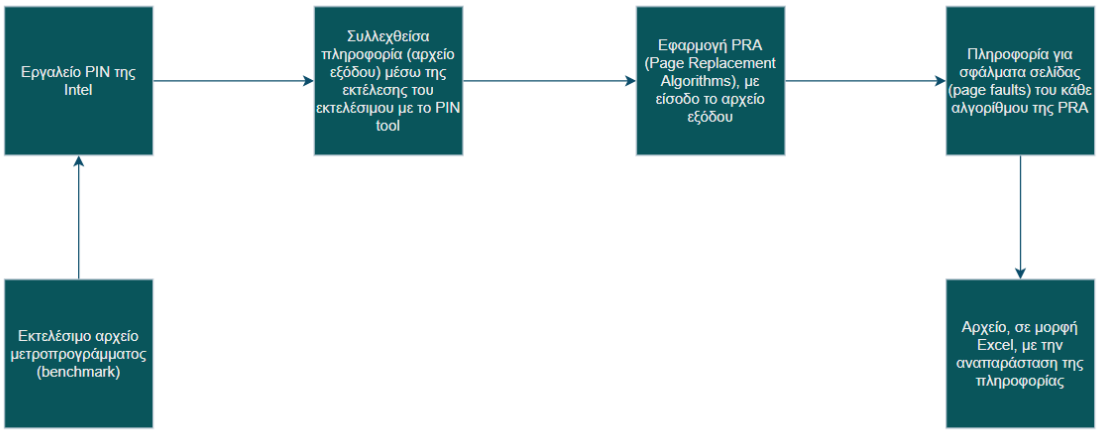
1.3 Μεθοδολογία

Η υλοποίηση της διατριβής έγινε σε τρία στάδια-βήματα:

1.Συλλογή της απαραίτητης πληροφορίας, μέσω της εκτέλεσης μετροπρογραμμάτων (benchmarks), με τη βοήθεια του Intel PIN tool. Η πληροφορία αφορά στην εύρεση των διευθύνσεων μνήμης, στις οποίες έγινε προσπέλαση. Αυτές οι αναφορές σε διευθύνσεις μνήμης αποθηκεύτηκαν σε ένα αρχείο. Με την κατάλληλη μετατροπή του εργαλείου (tool) που χρησιμοποιήθηκε, αποθηκεύτηκε, στο ίδιο αρχείο, η ακολουθία αναφοράς σελίδων.

2.Έρευνα για την υλοποίηση ενός ενιαίου εργαλείου, σε γλώσσα προγραμματισμού C, το οποίο περιλαμβάνει διαφορετικούς αλγορίθμους αντικατάστασης σελίδας.

3.Μελέτη, έχοντας ως είσοδο, στο παραπάνω εργαλείο, την ακολουθία αναφοράς σελίδων από το παραπάνω αρχείο, του τρόπου με τον οποίο θα συμπεριφερθούν οι διαφορετικοί αλγόριθμοι αντικατάστασης σελίδας, για το ίδιο benchmark για το οποίο συλλέξαμε αρχικά την πληροφορία. Με βάση τον αριθμό σφαλμάτων σελίδας που παρήγαγε ο κάθε αλγόριθμος, μπορέσαμε να εξετάσουμε ποιός αλγόριθμος είναι ο καλύτερος, αυτός δηλαδή με το μικρότερο αριθμό σφαλμάτων.



Διαγραμματική απεικόνιση των βημάτων της διπλωματικής διατριβής.

1.4 Καινοτομία

Ένα σημαντικό στοιχείο που καθιστά τη συγκεκριμένη μεταπτυχιακή διατριβή πρωτότυπη αποτελεί η χρήση του PIN tool της εταιρείας Intel. Το συγκεκριμένο εργαλείο περιλαμβάνει πολλά επιμέρους εργαλεία, η χρήση των οποίων βοηθά να εξαχθεί πληροφορία πάνω στα προγράμματα που ενδιαφέρουν τον χρήστη ή τον προγραμματιστή.

2 Διαχείριση μνήμης και σελιδοποίηση (paging)

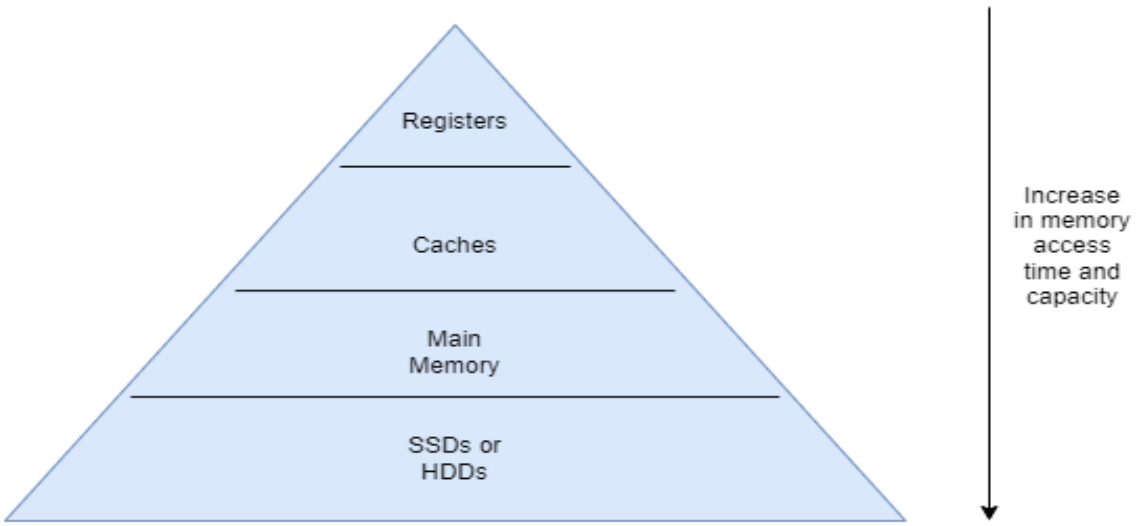
2.1 Διαχείριση μνήμης

Αρχικά, στα πλαίσια της παρούσας μεταπτυχιακής διατριβής, θα πρέπει να γίνει μια απαραίτητη εισαγωγή στις έννοιες της διαχείρισης μνήμης (memory management) και της σελιδοποίησης (paging).

Κάθε σύγχρονο υπολογιστικό σύστημα διαθέτει κύρια μνήμη. Πρόκειται για τη Μνήμη Άμεσης Προσπέλασης, γνωστή στα αγγλικά ως Random Access Memory (RAM). Αποτελεί βασικό μέρος του υλικού (hardware) κάθε Η/Υ και ιδιαίτερα σημαντικό πόρο (resource). Στην κύρια μνήμη «φορτώνονται» τα προγράμματα από το σκληρό δίσκο (δευτερεύουσα μνήμη), προκειμένου να εκτελεστούν από την Κεντρική Μονάδα Επεξεργασίας του συστήματος (Central Processing Unit ή CPU) ή πιο απλά, τον επεξεργαστή.

Ο λόγος που γίνεται διαχωρισμός σε κύρια (primary) και δευτερεύουσα (secondary) μνήμη είναι το γεγονός ότι, σε κάθε υπολογιστικό σύστημα υφίσταται η έννοια της «ιεραρχίας μνήμης» (memory hierarchy)[1]. Η ιδανική πραγματικότητα για κάθε χρήστη ή προγραμματιστή θα ήταν να κάνει χρήση απεριόριστης, σε μέγεθος, μνήμης, η οποία μάλιστα να είναι ταχύτατη και μη πτητική (non volatile). Μη πτητική σημαίνει να μην υφίσταται απώλεια δεδομένων στη μνήμη όταν προκύπτει διακοπή της παροχής ρεύματος στο υπολογιστικό σύστημα[2].

Βάσει της υπάρχουσας τεχνολογίας, αυτό δεν είναι εφικτό. Με την ανακάλυψη και ανάπτυξη της ιεραρχίας μνήμης, κάθε ηλεκτρονικός υπολογιστής διαθέτει διαφορετικά, σε χωρητικότητα, ταχύτητα και κόστος, είδη μνήμης[3]. Υπάρχουν μικρές σε χωρητικότητα, αλλά ταχείες, γρήγορες και ακριβές μνήμες, όπως οι καταχωρητές (hardware registers) και η «κρυφή» μνήμη (cache), μνήμες μέσης χωρητικότητας και ταχύτητας όπως η μνήμη RAM καθώς και μνήμες μεγάλης χωρητικότητας (σκληρός δίσκος). Τα τρία πρώτα είναι τα κύρια, εσωτερικά και πτητικά είδη μνήμης ενός υπολογιστή, ενώ ο σκληρός δίσκος αποτελεί δευτερεύουσα, εξωτερική και μη πτητική μνήμη. Όπως φαίνεται και από τα παραπάνω, τα μεγέθη των ειδών μνήμης είναι αντιστρόφως ανάλογα του κόστους και της ταχύτητάς τους.



Σχηματική απεικόνιση της ιεραρχίας μνήμης. Στη βάση της πυραμιδοειδούς αναπαράστασης βρίσκεται η δευτερεύουσα, εξωτερική και μη πτητική μνήμη ή σκληρός δίσκος (Hard Disk Drive-Solid State Drive). Η χωρητικότητα της κάθε μνήμης είναι αντιστρόφως ανάλογη της ταχύτητας και του κόστους της (Πηγή: https://www.embedded.com/understanding-cache-placement/).

1,2,3. Andrew S. Tanenbaum, «Σύγχρονα Λειτουργικά Συστήματα», 3η αμερικανική έκδοση, επιστημονική επιμέλεια ελληνικής έκδοσης Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών, εκδόσεις «Κλειδάριθμος», σελ.227.

Υπεύθυνο για τη διαχείριση της ιεραρχίας μνήμης σε έναν ηλεκτρονικό υπολογιστή είναι το Λειτουργικό Σύστημα και συγκεκριμένα, ένα τμήμα του, ο διαχειριστής μνήμης (memory manager).

Η διαχείριση μνήμης (memory management) χωρίζεται σε τρεις (3) εργασίες[4,5]: 1) έλεγχος για το ποια τμήματα της μνήμης χρησιμοποιούνται, 2) έλεγχος για το ποιες διαδικασίες απαιτούν μνήμη καθώς και η κατανομή της σε αυτές και 3) έλεγχος για το ποιες διαδικασίες δε χρειάζονται πλέον μνήμη, με αποτέλεσμα την αποδέσμευσή της.

2.2 Σελιδοποίηση (paging)

Για το έργο της διαχείρισης μνήμης έχουν επινοηθεί και χρησιμοποιούνται διάφορες μέθοδοι. Στα σύγχρονα υπολογιστικά συστήματα χρησιμοποιείται κυρίως η μέθοδος της σελιδοποίησης (paging).

Συγκεκριμένα, η σελιδοποίηση αποτελεί μέθοδο διαχείρισης εικονικής μνήμης (virtual memory). Κατά συνέπεια, προκειμένου να περιγραφεί, θα πρέπει πρώτα να απαντηθεί το εξής ερώτημα: τί είναι η εικονική μνήμη;

Τα προγράμματα που εκτελούνται σε έναν υπολογιστή αυξάνονται ταχύτερα, σε μέγεθος, αναλογικά με τη μνήμη του. Συνέπεια αυτού αποτελεί το γεγονός ότι στα σημερινά υπολογιστικά συστήματα πρέπει να εκτελούνται ταυτόχρονα πολλαπλά προγράμματα μεγαλύτερης, τόσο μεμονωμένα όσο και συνολικά, χωρητικότητας από τη φυσική μνήμη ενός υπολογιστή. Το πρόβλημα αυτό λύνεται με τη βοήθεια της εικονικής μνήμης[6,7].

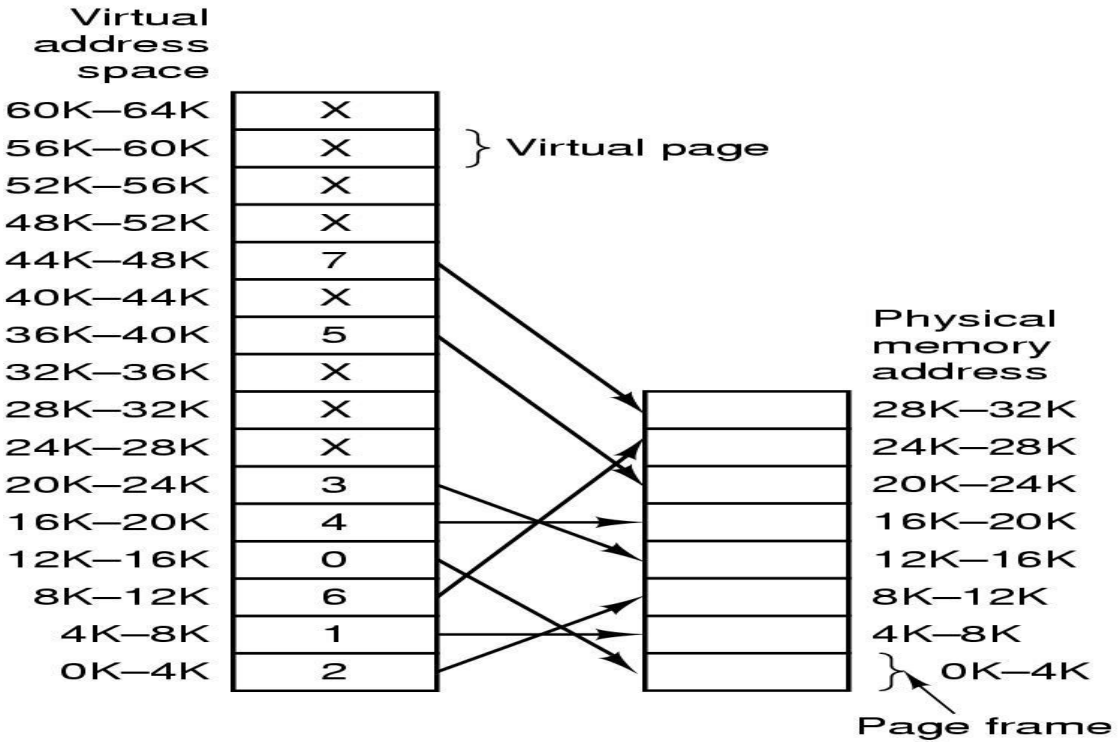
Η κεντρική ιδέα πίσω από το αφαιρετικό αυτό μόντελο είναι το γεγονός ότι κάθε διαδικασία (δηλαδή, κάθε πρόγραμμα που εκτελείται) έχει το δικό της εικονικό χώρο διευθύνσεων (virtual address space), στο σκληρό δίσκο (δευτερεύουσα μνήμη). Ο εικονικός χώρος διευθύνσεων αποτελεί την εικονική μνήμη (virtual memory). Προκειμένου να εκτελεστεί μια διαδικασία, θα πρέπει πρώτα να «φορτωθεί» στην κύρια, φυσική μνήμη, τη RAM. Όπως προαναφέρθηκε όμως, μία και μόνο διαδικασία είναι δυνατόν να έχει μεγαλύτερη χωρητικότητα από την ίδια τη RAM, ενώ θα πρέπει να ληφθεί υπόψη και το γεγονός ότι αυτήν τη RAM θα πρέπει να τη μοιραστούν περισσότερες από μία διαδικασίες.

Η λύση στο πρόβλημα αυτό είναι η μέθοδος της σελιδοποίησης[8]. Σύμφωνα με αυτή, δε χρειάζεται κάθε διαδικασία να «φορτώνεται» όλοκληρη στην κύρια μνήμη και να εκτελείται από τον επεξεργαστή του υπολογιστή, παρά μόνο ένα συγκεκριμένο τμήμα της, μια δεδομένη χρονική στιγμή. Με αυτόν τον τρόπο, πολλές διαδικασίες μπορούν να μοιράζονται ταυτόχρονα την ίδια κύρια μνήμη. Δίνεται έτσι, η εντύπωση ότι, όλη η εικονική μνήμη «χωράει» στην πιο περιορισμένη, φυσική μνήμη (RAM).

Τα τμήματα αυτά των διαδικασιών ονομάζονται σελίδες (pages)[9,10]. Κάθε διαδικασία χωρίζεται σε σελίδες ίσου μεγέθους. Η κάθε σελίδα περιέχει ένα τμήμα του εικονικού χώρου διευθύνσεων, δηλαδή της εικονικής μνήμης, για την οποία έγινε λόγος παραπάνω. Αντίστοιχα, η RAM χωρίζεται με τη σειρά της σε τμήματα, τα οποία ονομάζονται πλαίσια σελίδων (page frames)[11,12]. Τα πλαίσια σελίδων έχουν ίσο μέγεθος μεταξύ τους, αλλά και με τις σελίδες. Στα σύγχρονα υπολογιστικά συστήματα το μέγεθος αυτό είναι συνήθως τα 4096 bytes (4KB).

Έστω ότι μια δεδομένη χρονική στιγμή, ο επεξεργαστής του υπολογιστή χρειάζεται μια συγκεκριμένη σελίδα. Αν αυτή η σελίδα βρίσκεται ήδη σε ένα πλαίσιο σελίδας της κύριας μνήμης (RAM), τότε εκτελείται κανονικά. Υπάρχει όμως πιθανότητα αυτή η σελίδα να μη βρίσκεται μέσα σε κάποιο πλαίσιο. Τότε, εμφανίζεται το λεγόμενο «σφάλμα σελίδας»[13]. Σε αυτήν την περίπτωση, το Λειτουργικό Σύστημα θα πρέπει να ανασύρει τη σελίδα από τη δευτερεύουσα μνήμη (σκληρό δίσκο) και να την τοποθετήσει σε ένα διαθέσιμο πλαίσιο. Αν δεν υπάρχει κανένα ελεύθερο πλαίσιο, θα πρέπει να αντικατασταθεί κάποια από τις ήδη υπάρχουσες σελίδες, εντός της κύριας μνήμης, από τη ζητούμενη.

4. Andrew S. Tanenbaum, «Σύγχρονα Λειτουργικά Συστήματα», 3η αμερικανική έκδοση, επιστημονική επιμέλεια ελληνικής έκδοσης Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών, εκδόσεις «Κλειδάριθμος», σελ.227-228.
5. Peter J. Denning, «Third Generation Computer Systems», ACM Computing Surveys, Vol. 3, No. 4, December 1971, p. 190.
6. Kartik Moudgil, Anushka Ringshia, Harshal Bharatkumar Parekh, Ria Maheshwari, Sheetal Chaudhari, «Systematic Analysis and Simulation of Classical Page Replacement Algorithms: A Proposed Novel WSClock-Derivative», IEEE, 2017 2nd International Conference for Convergence in Technology (I2CT), p. 887.
7. Andrew S. Tanenbaum, «Σύγχρονα Λειτουργικά Συστήματα», 3η αμερικανική έκδοση, επιστημονική επιμέλεια ελληνικής έκδοσης Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών, εκδόσεις «Κλειδάριθμος», σελ.242.



Απεικόνιση της σχέσης ανάμεσα στον εικονικό χώρο διευθύνσεων μιας διαδικασίας και τη φυσική, κύρια μνήμη ενός Η/Υ. Κάθε διαδικασία χωρίζεται σε τμήματα ίσου μεγέθους (σελίδες), που "φορτώνονται" στα αντίστοιχα τμήματα, επίσης ίσου μεγέθους (πλαίσια σελίδων), στα οποία χωρίζεται η κύρια μνήμη (Πηγή: "Σύγχρονα Λειτουργικά Συστήματα" (Modern Operating Systems), Andrew S. Tanenbaum).

Η παρούσα εργασία έχει ως αντικείμενο τη μελέτη αλγορίθμων που είναι υπεύθυνοι για αυτήν ακριβώς την αντικατάσταση σελίδων. Η μελέτη αυτή έγινε με χρήση πραγματικών δεδομένων.

8. Kartik Moudgil, Anushka Ringshia, Harshal Bharatkumar Parekh, Ria Maheshwari, Sheetal Chaudhari, «Systematic Analysis and Simulation of Classical Page Replacement Algorithms: A Proposed Novel WSClock-Derivative», IEEE, 2017 2nd International Conference for Convergence in Technology (I2CT), p. 887.
9. Andrew S. Tanenbaum, «Σύγχρονα Λειτουργικά Συστήματα», 3η αμερικανική έκδοση, επιστημονική επιμέλεια ελληνικής έκδοσης Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών, εκδόσεις «Κλειδάριθμος», σελ. 244.
10. Alfred V. Aho, Peter J. Denning, Jeffrey D. Ullman, «Principles of Optimal Page Replacement», January 1971, Journal of the ACM, Volume 18, Issue 1, p. 80.
11. Andrew S. Tanenbaum, «Σύγχρονα Λειτουργικά Συστήματα», 3η αμερικανική έκδοση, επιστημονική επιμέλεια ελληνικής έκδοσης Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών, εκδόσεις «Κλειδάριθμος», σελ. 244.
12. Alfred V. Aho, Peter J. Denning, Jeffrey D. Ullman, «Principles of Optimal Page Replacement», January 1971, Journal of the ACM, Volume 18, Issue 1, p. 80.
13. Kartik Moudgil, Anushka Ringshia, Harshal Bharatkumar Parekh, Ria Maheshwari, Sheetal Chaudhari, «Systematic Analysis and Simulation of Classical Page Replacement Algorithms: A Proposed Novel WSClock-Derivative», IEEE, 2017 2nd International Conference for Convergence in Technology (I2CT), p. 887.

3 Το εργαλείο PIN της Intel

3.1 Εισαγωγή

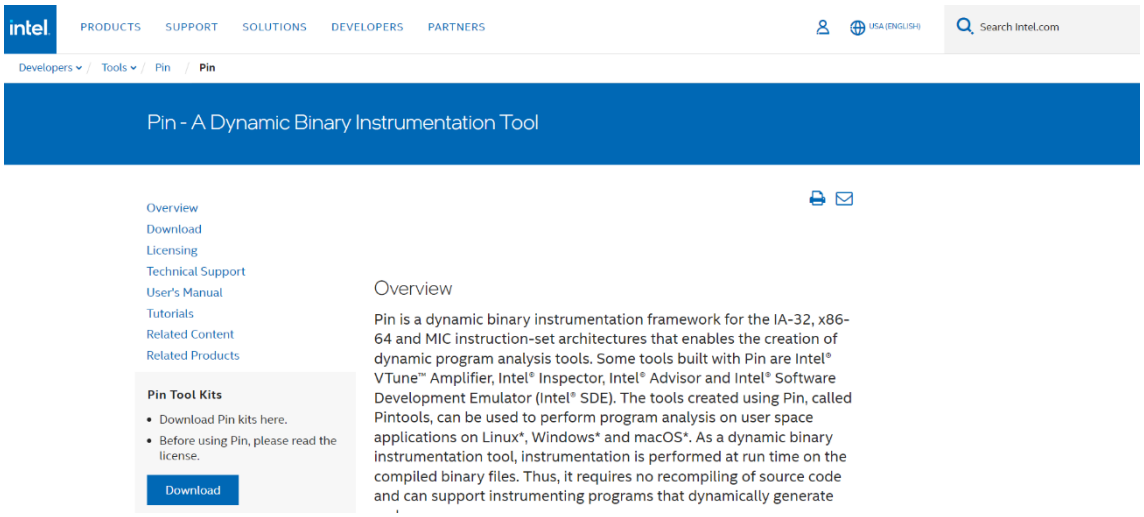
Για τη μελέτη των αλγορίθμων αντικατάστασης σελίδας, απαραίτητη προϋπόθεση είναι η παραγωγή πραγματικών δεδομένων προγραμμάτων. Για να καταστεί αυτό εφικτό, έγινε χρήση του εργαλείου PIN της εταιρείας Intel.

Το Intel PIN είναι ένα εργαλείο συλλογής δεδομένων. Αναπτύχθηκε από την Intel και βοηθά τους χρήστες και προγραμματιστές να αναλύσουν και να τροποποιήσουν ένα πρόγραμμα, στα πλαίσια ερευνητικής δραστηριότητας ή προκειμένου να βελτιστοποιήσουν την απόδοσή του. Είναι συμβατό τόσο με Windows όσο και με Linux, σε ό,τι αφορά τα λειτουργικά συστήματα, ενώ υποστηρίζει διαφορετικές αρχιτεκτονικές επεξεργαστών, όπως για παράδειγμα οι x86 (32-bit επεξεργαστές), οι IA-32 (32-bit επεξεργαστές) και οι x86-64 (64-bit επεξεργαστές).

Το PIN περιλαμβάνει πολλά επιμέρους εργαλεία (tools), κάθε ένα από τα οποία εισάγει τυχαίο κώδικα (γραμμένο σε γλώσσα C ή C++) , σε τυχαία σημεία του εκτελέσιμου αρχείου ενός προγράμματος. Η τελική πληροφορία που παράγεται από αυτήν τη διαδικασία μπορεί να περιλαμβάνει τον αριθμό εντολών (instruction count) ενός προγράμματος, τις εικονικές διευθύνσεις τους (virtual memory addresses), το χρόνο εκτέλεσής τους κ.τ.λ.

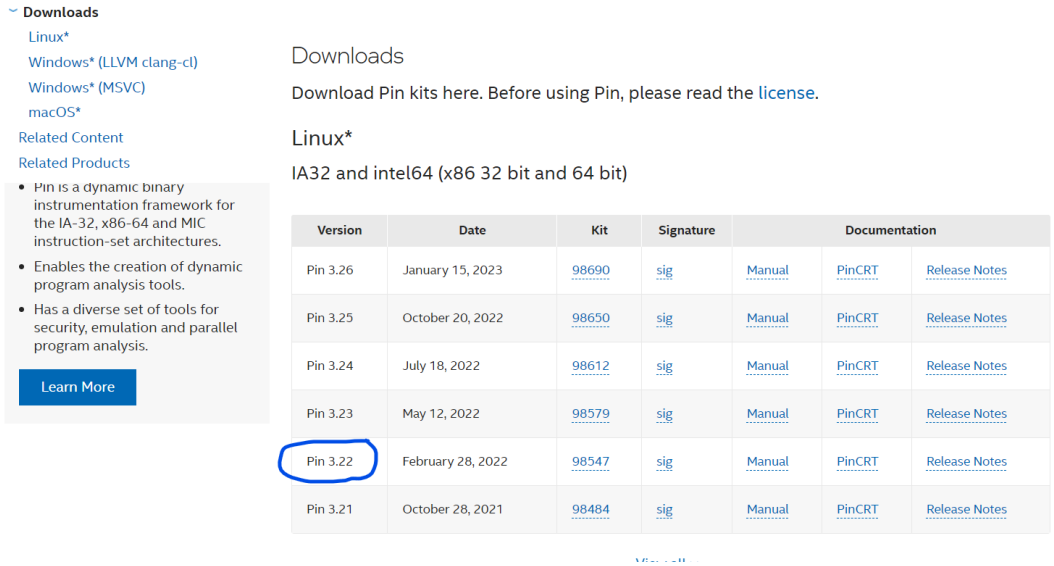
Μερικά ενδεικτικά εργαλεία του Intel PIN είναι τα «inscount.cpp», «itrace.cpp», «strace.cpp», «proccount.cpp» και «pinatrace.cpp». Κάθε ένα από αυτά παρέχει διαφορετικού είδους πληροφορία για ένα συγκεκριμένο πρόγραμμα. Για τις ανάγκες της εργασίας, χρησιμοποιήθηκε το «pinatrace.cpp». Όπως φανερώνει και η επέκταση «.cpp», τόσο αυτό το εργαλείο, όσο και τα υπόλοιπα που αναφέρθηκαν, είναι γραμμένα στη γλώσσα C++. Επίσης, σε αυτό το το σημείο θα πρέπει να σημειωθεί ότι όλο το έργο της μελέτης των αλγορίθμων αντικατάστασης σελίδας έγινε σε περιβάλλον Linux και συγκεκριμένα σε Debian 10 (64-bit), το οποίο αποτελεί διανομή (distribution) του Linux.

Το πρώτο πράγμα που πρέπει να κάνει ο χρήστης σε αυτήν τη φάση της εργασίας είναι να κάνει λήψη (download) του PIN από τον αντίστοιχο ιστότοπο της Intel[14]. Υπάρχουν διαφορετικές εκδόσεις του Intel PIN. Για τις ανάγκες της μεταπτυχιακής διατριβής, χρησιμοποιήθηκε η έκδοση 3.22.



Η κεντρική σελίδα του ιστοτόπου για το εργαλείο PIN της Intel. Στα αριστερά της σελίδας, υπάρχει η επιλογή "Download", προκειμένου ο χρήστης να κάνει λήψη της έκδοσης του PIN που επιθυμεί (Πηγή: https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html).

14. https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html.



Downloads

Linux*

Windows* (LLVM clang-cl)

Windows* (MSVC)

macOS*

Related Content

Related Products

- Pin is a dynamic binary instrumentation framework for the IA-32, x86-64 and MIC instruction-set architectures.
- Enables the creation of dynamic program analysis tools.
- Has a diverse set of tools for security, emulation and parallel program analysis.

Learn More

Downloads

Download Pin kits here. Before using Pin, please read the license.

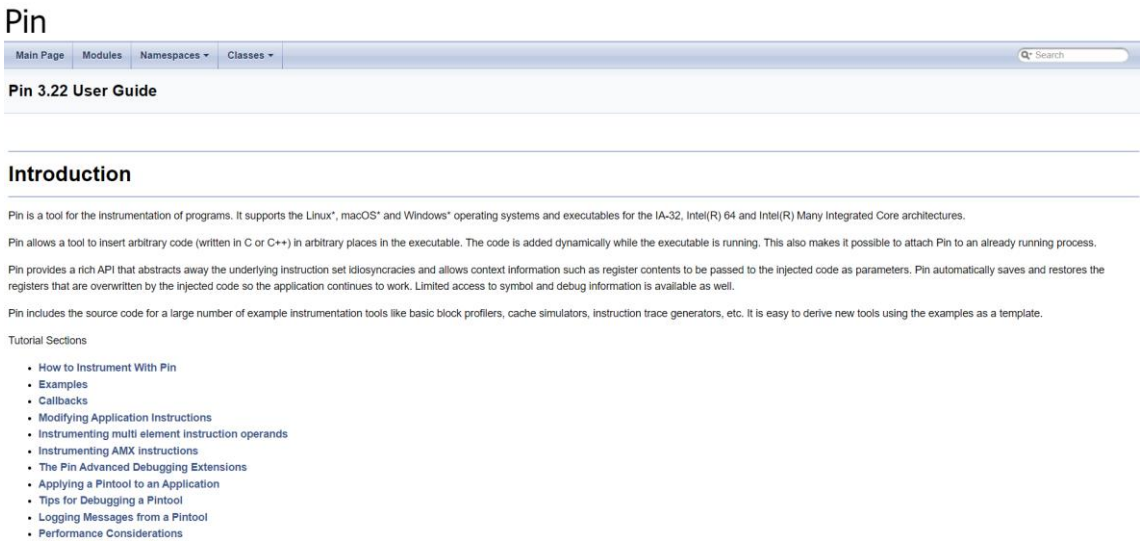
Linux*

IA32 and intel64 (x86 32 bit and 64 bit)

Version	Date	Kit	Signature	Documentation		
Pin 3.26	January 15, 2023	98690	sig	Manual	PinCRT	Release Notes
Pin 3.25	October 20, 2022	98650	sig	Manual	PinCRT	Release Notes
Pin 3.24	July 18, 2022	98612	sig	Manual	PinCRT	Release Notes
Pin 3.23	May 12, 2022	98579	sig	Manual	PinCRT	Release Notes
Pin 3.22	February 28, 2022	98547	sig	Manual	PinCRT	Release Notes
Pin 3.21	October 28, 2021	98484	sig	Manual	PinCRT	Release Notes

Αφού ο χρήστης πάτησει την επιλογή "Download" και στη συνέχεια ακολουθήσει, πατώντας επίσης, το σύνδεσμο "Pin - A Binary Instrumentation Tool – Downloads", θα βρεθεί στη σελίδα με τις διαθέσιμες εκδόσεις του PIN. Στη φωτογραφία, οι εκδόσεις για Linux και αρχιτεκτονικές IA32 και intel64 (Πηγή: https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-binary-instrumentation-tool-downloads.html).

Όπως φαίνεται και από την παραπάνω φωτογραφία, στη σελίδα «Downloads», ο χρήστης θα βρει, πέρα από τις διαθέσιμες εκδόσεις του Intel PIN, πλήρες εγχειρίδιο (Manual) για την κάθε μια από αυτές. Το εγχειρίδιο αποτελεί τον Οδηγό Χρήστη, με αναλυτικές πληροφορίες για τα επιμέρους εργαλεία του PIN, καθώς και οδηγίες για την κατασκευή και εκτέλεσή τους (build and run), τόσο σε 32-bit αρχιτεκτονικές (IA32), όσο και 64-bit (intel64).



Pin

Main Page | Modules | Namespaces | Classes | Search

Pin 3.22 User Guide

Introduction

Pin is a tool for the instrumentation of programs. It supports the Linux*, macOS* and Windows* operating systems and executables for the IA-32, Intel(R) 64 and Intel(R) Many Integrated Core architectures.

Pin allows a tool to insert arbitrary code (written in C or C++) in arbitrary places in the executable. The code is added dynamically while the executable is running. This also makes it possible to attach Pin to an already running process.

Pin provides a rich API that abstracts away the underlying instruction set idiosyncracies and allows context information such as register contents to be passed to the injected code as parameters. Pin automatically saves and restores the registers that are overwritten by the injected code so the application continues to work. Limited access to symbol and debug information is available as well.

Pin includes the source code for a large number of example instrumentation tools like basic block profilers, cache simulators, instruction trace generators, etc. It is easy to derive new tools using the examples as a template.

Tutorial Sections

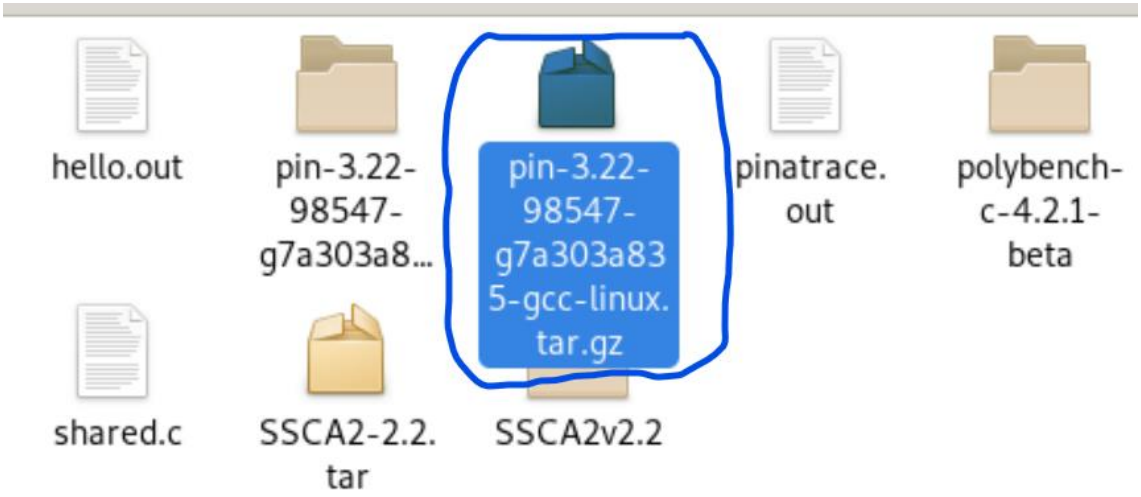
- How to Instrument With Pin
- Examples
- Callbacks
- Modifying Application Instructions
- Instrumenting multi element instruction operands
- Instrumenting AMX instructions
- The Pin Advanced Debugging Extensions
- Applying a Pintool to an Application
- Tips for Debugging a Pintool
- Logging Messages from a Pintool
- Performance Considerations

Φωτογραφία του εγχειριδίου (Manual) για το PIN. Πρόκειται για τον Οδηγό Χρήστη (User Guide), με αναλυτικές πληροφορίες για τα επιμέρους εργαλεία, όπως επίσης και οδηγίες για την κατασκευή και εκτέλεσή τους σε 32-bit και 64-bit αρχιτεκτονικές (Πηγή: https://software.intel.com/sites/landingpage/pintool/docs/98547/Pin/html/index.html).

3.2 Κατασκευή και εκτέλεση εργαλείου του PIN

Όπως αναφέρθηκε και στην υποενότητα 3.1 (Εισαγωγή), κάθε εργαλείο του PIN της Intel μπορεί να κατασκευαστεί και να εκτελεστεί (build and run) τόσο σε περιβάλλον Windows, όσο και σε Linux, με kits διαθέσιμα για 32-bit αλλά και 64-bit αρχιτεκτονικές.

Αφού ο χρήστης λάβει το kit που επιθυμεί, σε συμπιεσμένη μορφή, κάνει αποσυμπίεση του αρχείου και μπορεί να βρει τον πηγαίο κώδικα (source code), για το εργαλείο που θα επιλέξει, μέσα στον αντίστοιχο φάκελο (folder). Προκειμένου όμως, να προχωρήσει στην κατασκευή και την εκτέλεση του εργαλείου, θα πρέπει πρώτα να ανατρέξει στο εγχειρίδιο (Manual) της έκδοσης του PIN που έχει επιλέξει για λήψη.



Στη φωτογραφία, με μπλε επισήμανση, το συμπιεσμένο αρχείο του Intel PIN. Στα αριστερά του, η αποσυμπιεσμένη του μορφή. Έχει επιλεγεί για λήψη η έκδοση 3.22.

Το Manual του PIN της Intel είναι ο Οδηγός Χρήστη (User Guide) και περιλαμβάνει, μεταξύ άλλων, τις απαραίτητες εντολές, που πρέπει να εισαγάγει ο χρήστης προκειμένου να κατασκευάσει και να εκτελέσει ένα συγκεκριμένο εργαλείο.

Τα εργαλεία που βρίσκονται στους φακέλους είναι έτοιμα παραδείγματα, τον κώδικα των οποίων ο χρήστης μπορεί ακόμα και να επεξεργαστεί, ώστε να λαμβάνει τις πληροφορίες που επιθυμεί, για μια συγκεκριμένη εφαρμογή.Το σετ εντολών για 32-bit και 64-bit αρχιτεκτονικές, σε περιβάλλον Linux, είναι το εξής:

- ```
  $ cd source/tools/ManualExamples.
  $ make all TARGET=ia32.
  ```
- ```
  $ cd source/tools/ManualExamples.
  $ make all TARGET=intel64.
  ```
- ```
  $ cd source/tools/ManualExamples.
  $ make inscount0.test TARGET=intel64.
  ```
- ```
  $ cd source/tools/ManualExamples.
  $ make obj-intel64/inscount0.so TARGET=intel64.
  ```
- ```
  $ cd source/tools/ManualExamples.
  $ make obj-ia32/inscount0.so TARGET=ia32.
  ```
  ```

Building the Example Tools

To build all examples in a directory for ia32 architecture:

```
$ cd source/tools/ManualExamples
$ make all TARGET=ia32
```

To build all examples in a directory for intel64 architecture:

```
$ cd source/tools/ManualExamples
$ make all TARGET=intel64
```

To build and run a specific example (e.g., inscount0):

```
$ cd source/tools/ManualExamples
$ make inscount0.test TARGET=intel64
```

To build a specific example without running it (e.g., inscount0):

```
$ cd source/tools/ManualExamples
$ make obj-intel64/inscount0.so TARGET=intel64
```

The above applies to the Intel(R) 64 architecture. For the IA-32 architecture, use TARGET=ia32 instead.

```
$ cd source/tools/ManualExamples
$ make obj-ia32/inscount0.so TARGET=ia32
```

Το σετ εντολών για την κατασκευή και εκτέλεση των PIN εργαλείων, όπως φαίνεται στον ιστότοπο του Intel PIN (Πηγή: https://software.intel.com/sites/landingpage/pintool/docs/98547/Pin/html/index.html#BuildingExamples).

Σε κάθε ένα από τα παραπάνω bullets, η πρώτη από τις δύο εντολές είναι η διαδρομή (path) που οδηγεί στο φάκελο με τα εργαλεία-παραδείγματα. Στην προκειμένη περίπτωση, ο χρήστης οδηγείται στο φάκελο «ManualExamples» του PIN. Η δεύτερη εντολή του πρώτου bullet εισάγεται προκειμένου να κατασκευαστούν, χωρίς να εκτελεστούν, όλα τα εργαλεία που βρίσκονται στο συγκεκριμένο φάκελο, σε 32-bit αρχιτεκτονική (TARGET=ia32). Την ίδια λειτουργία έχει και η δεύτερη εντολή του δεύτερου bullet, με τη διαφορά ότι αφορά στη 64-bit αρχιτεκτονική (TARGET=intel64).

Η δεύτερη εντολή του τρίτου bullet εισάγεται από το χρήστη για την κατασκευή και εκτέλεση ενός συγκεκριμένου παραδείγματος, εντός του φακέλου «ManualExamples». Εδώ, πρόκειται για το εργαλείο «inscount0», που δίνει το άθροισμα των εντολών για μια εφαρμογή. Η δεύτερη εντολή του τέταρτου bullet αφορά στην κατασκευή, χωρίς εκτέλεση, του συγκεκριμένου παραδείγματος, επίσης σε 64-bit αρχιτεκτονική. Τέλος, η δεύτερη εντολή του πέμπτου bullet έχει τον ίδιο στόχο με την προηγούμενη, χρησιμοποείται όμως για 32-bit αρχιτεκτονική. Στην περίπτωση κατασκευής και εκτέλεσης ενός εργαλείου, δημιουργείται ένα OUT file (αρχείο με επέκταση .out), το οποίο περιλαμβάνει την πληροφορία για μια εφαρμογή.

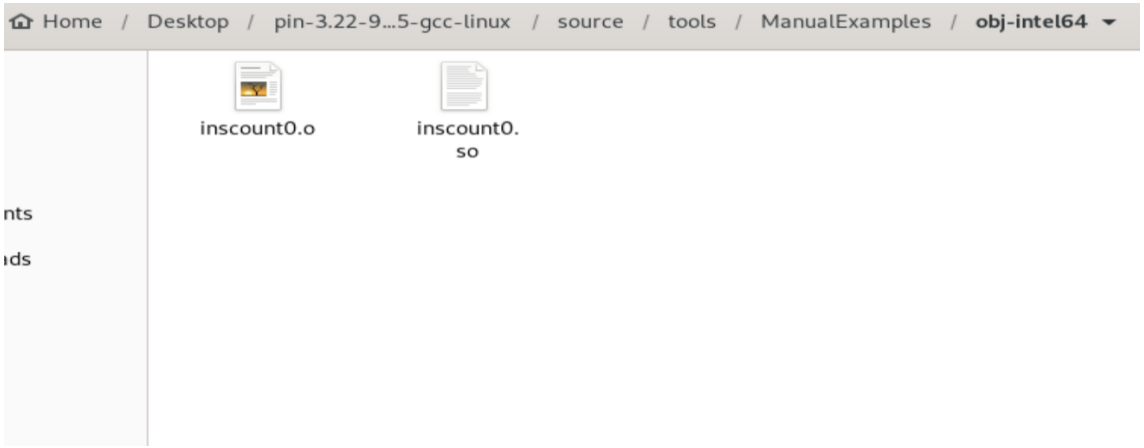
Όπως έχει αναφερθεί και στην υποενότητα 3.1 (Εισαγωγή), η κατασκευή και εκτέλεση ενός έτοιμου παραδείγματος μπορεί να γίνει και σε περιβάλλον Windows, για τις ανάγκες της εργασίας όμως, χρησιμοποιήθηκε η διανομή Debian 10 (64-bit) για Linux.

```
osboxes@osboxes:~/Desktop/pin-3.22-98547-g7a303a835-gcc-linux/source/tools/ManualExamples$ ls
2mm_base                fork_app.cpp        nonstatica.cpp
3mm_base                fork_jit_tool.cpp   obj-ia32
adi_base                hello.out           obj-intel64
atax_base               imageload.cpp       pinatrace.cpp
bicg_base               inscount0.cpp       pin.log
buffer_linux.cpp        inscount1.cpp       proccount.cpp
buffer_windows.cpp      inscount2.cpp       replacesigprobed.cpp
countreps.cpp           inscount.out        safecopy.cpp
covariance_base         inscount_tls.cpp    seidel-2d_base
detach.cpp              invocation.cpp      shared.c
divide_by_zero_unix.c   isampling.cpp       SSCA2
divide_by_zero_win.c    itrace.cpp          stack-debugger.cpp
doitgen_base            itrace.out          stack-debugger-tutorial.sln
dumpargv.cpp            jacobi-1d_base      stack-debugger-tutorial.vcxproj
durbin_base             jacobi-2d_base      stack-debugger-tutorial.vcxproj.filters
emudiv.cpp              little_malloc.c     statica.cpp
fdtd-2d_base            lu_base             staticcount.cpp
```

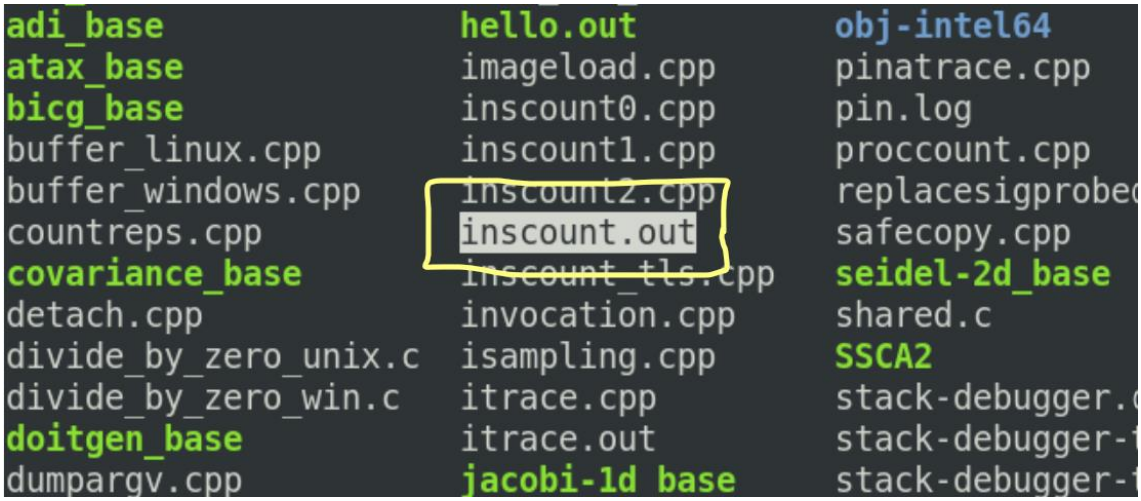
Το πρώτο στάδιο της διαδικασίας κατασκευής και εκτέλεσης ενός έτοιμου παραδείγματος. Συγκεκριμένα, όταν ο χρήστης ακολουθήσει τη διαδρομή με τη μπλε επισήμανση, οδηγείται στο φάκελο "ManualExamples", όπου θα βρει έτοιμα εργαλεία του PIN, όπως το "inscount0.cpp", το "itrace.cpp" και το "pinatrace.cpp".

```
osboxes@osboxes:~/Desktop/pin-3.22-98547-g7a303a835-gcc-linux/source/tools/ManualExamples$ make in
scount0.test TARGET=intel64
g++ -Wall -Werror -Wno-unknown-pragmas -DPIN_CRT=1 -fno-stack-protector -fno-exceptions -funwind-t
ables -fasynchronous-unwind-tables -fno-rtti -DTARGET_IA32E -DHOST_IA32E -fPIC -DTARGET_LINUX -fab
i-version=2 -faligned-new  -I../../../source/include/pin -I../../../source/include/pin/gen -isyste
m /home/osboxes/Desktop/pin-3.22-98547-g7a303a835-gcc-linux/extras/stlport/include -isystem /home/
osboxes/Desktop/pin-3.22-98547-g7a303a835-gcc-linux/extras/libstdc++/include -isystem /home/osboxe
s/Desktop/pin-3.22-98547-g7a303a835-gcc-linux/extras/crt/include -isystem /home/osboxes/Desktop/pi
n-3.22-98547-g7a303a835-gcc-linux/extras/crt/include/arch-x86_64 -isystem /home/osboxes/Desktop/pi
n-3.22-98547-g7a303a835-gcc-linux/extras/crt/include/kernel/uapi -isystem /home/osboxes/Desktop/pi
n-3.22-98547-g7a303a835-gcc-linux/extras/crt/include/kernel/uapi/asm-x86 -I../../../extras/compone
nts/include -I../../../extras/xed-intel64/include/xed -I../../../source/tools/Utils -I../../../sou
```

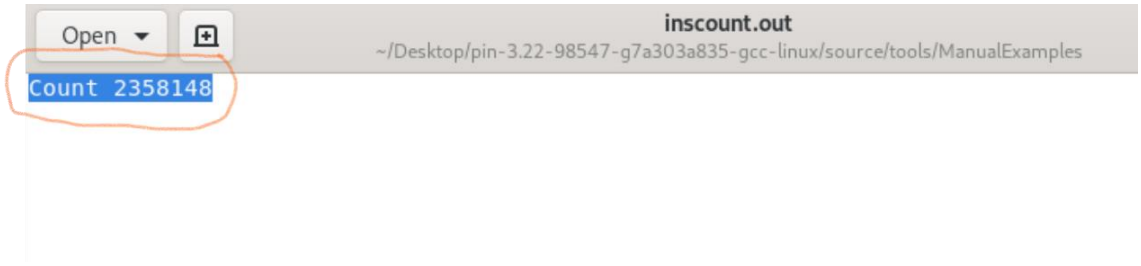
Με τη γκρίζα επισήμανση, η εντολή που εισάγει ο χρήστης για την κατασκευή και εκτέλεση του παραδείγματος "inscount0.cpp" και η οποία εφαρμόζεται για 64-bit αρχιτεκτονική (TARGET=intel64).



Κατά την κατασκευή του παραδείγματος "inscount0.cpp", δημιουργούνται, στον υποφάκελο "obj-intel64", τα αρχεία "inscount0.o" και "inscount0.so".



Κατά την εκτέλεση του παραδείγματος, δημιουργείται, εντός του "ManualExamples", το αρχείο "inscount.out". Το συγκεκριμένο .out αρχείο περιέχει το συνολικό αριθμό εντολών για το "inscount0.cpp".



Το αρχείο "inscount.out" περιέχει το συνολικό αριθμό εντολών (count) για το "inscount0.cpp".

3.3 Το εργαλείο «pinatrace.cpp» του Intel PIN

3.3.1 Περιγραφή του εργαλείου

Στα πλαίσια της παρούσης εργασίας, χρησιμοποιήθηκε το «pinatrace.cpp». Το συγκεκριμένο εργαλείο βρίσκεται εντός του φακέλου «ManualExamples», στην αποσυμπιεσμένη μορφή του kit που λαμβάνει ο χρήστης.

Όπως και τα υπόλοιπα, έτοιμα παραδείγματα, το «pinatrace.cpp» δίνει με τη σειρά του πληροφορίες στο χρήστη για μία εφαρμογή. Η διαδικασία κατασκευής και εκτέλεσης του εργαλείου είναι ίδια με αυτήν του παραδείγματος της προηγούμενης υποενότητας.

Χρησιμοποιείται η ίδια εντολή, ενώ, κατά την κατασκευή, δημιουργούνται τα αντίστοιχα «.o» και «.so» αρχεία στον υποφάκελο «obj-intel64». Επιπλέον, κατά την εκτέλεση, παράγεται το αντίστοιχο OUT αρχείο, το «pinatrace.out».

Το συγκεκριμένο αρχείο εξόδου περιέχει διαφορετικού είδους πληροφορία σε σχέση με το παραχθέν αρχείο για το «inscount0.cpp». Μόλις ο χρήστης το ανοίξει, θα δει τρεις στήλες. Από αριστερά προς τα δεξιά, αυτές εμφανίζουν τις εικονικές διευθύνσεις των εντολών της διαδικασίας, τα δικαιώματα εγγραφής ή ανάγνωσης και τέλος, τις διευθύνσεις της φυσικής, κύριας μνήμης στις οποίες αντιστοιχούν. Να σημειωθεί εδώ πως η φράση «δικαιώματα εγγραφής ή ανάγνωσης» σημαίνει ότι μια εντολή κάνει εγγραφή στην κύρια μνήμη ή ανάγνωση από αυτήν, αντίστοιχα.

```
0x7ffc686e988e: R 0x61f080
0x7ffc686e9890: R 0x61f088
0x7ffc686e9892: R 0x61f090
0x7ffc686e9894: R 0x61f098
0x7ffc686e9896: R 0x61f0a0
0x7ffc686e9897: R 0x61f0a8
0x7ffc6873c24e: R 0x7ffc68836ab0
0x7ffc6873c254: R 0x61f128
0x7ffc6873c2fc: W 0x7ffc6883cbc8
0x7ffc6873c303: R 0x61f208
0x7ffc6873c30d: R 0xf84b8
0x7ffc6873c30d: W 0xf84b8
0x7ffc6873c325: R 0xf8480
0x7ffc6873c329: W 0x61f0a8
0x7ffc686ea6cc: W 0x61f0b0
0x7ffc686ea6d1: W 0x61f0b8
0x7ffc686ea6d6: W 0x61f0a0
0x7ffc686ea6db: R 0x336060
```

Τα περιεχόμενα του "pinatrace.out". Στην πρώτη στήλη από αριστερά, οι εικονικές διευθύνσεις των εντολών του "pinatrace.cpp", στη δεύτερη τα δικαιώματα εγγραφής και ανάγνωσης για την κάθε εντολή και στην τρίτη στήλη, οι φυσικές διευθύνσεις των εντολών στην κύρια μνήμη.

Παρακάτω, παρουσιάζεται ο κώδικας για το «pinatrace.cpp»:

```
/*
 * Copyright (C) 2004-2021 Intel Corporation.
 * SPDX-License-Identifier: MIT
 */

/*
 *  This file contains an ISA-portable PIN tool for tracing memory accesses.
 */
#include <stdio.h>
#include "pin.H"

FILE* trace;
// Print a memory read record
VOID RecordMemRead(VOID* ip, VOID* addr) { fprintf(trace, "%p: R %p\n", ip,
addr); }
// Print a memory write record
VOID RecordMemWrite(VOID* ip, VOID* addr) { fprintf(trace, "%p: W %p\n", ip,
addr); }
// Is called for every instruction and instruments reads and writes
VOID Instruction(INS ins, VOID* v)
{
    // Instruments memory accesses using a predicated call, i.e.
    // the instrumentation is called iff the instruction will actually be
executed.
    // On the IA-32 and Intel(R) 64 architectures conditional moves and REP
    // prefixed instructions appear as predicated instructions in Pin.
    UINT32 memOperands = INS_MemoryOperandCount(ins);
    // Iterate over each memory operand of the instruction.
    for (UINT32 memOp = 0; memOp < memOperands; memOp++)
    {
        if (INS_MemoryOperandIsRead(ins, memOp))
        {
            INS_InsertPredicatedCall(ins, IPOINT_BEFORE,
(AFUNPTR)RecordMemRead, IARG_INST_PTR, IARG_MEMORYOP_EA, memOp,
                                     IARG_END);
        }
        // Note that in some architectures a single memory operand can be
        // both read and written (for instance incl (%eax) on IA-32)
        // In that case we instrument it once for read and once for write.
        if (INS_MemoryOperandIsWritten(ins, memOp))
        {
            INS_InsertPredicatedCall(ins, IPOINT_BEFORE,
(AFUNPTR)RecordMemWrite, IARG_INST_PTR, IARG_MEMORYOP_EA, memOp,
                                     IARG_END);
        }
    }
}
VOID Fini(INT32 code, VOID* v)
{
    fprintf(trace, "#eof\n");
    fclose(trace);
}
/* ===================================================================== */
/* Print Help Message                                                    */
/* ===================================================================== */
INT32 Usage()
```

```
{
    PIN_ERROR("This Pintool prints a trace of memory addresses\n" +
KNOB_BASE::StringKnobSummary() + "\n");
    return -1;
}
/* ===================================================================== */
/* Main                                                                  */
/* ===================================================================== */
int main(int argc, char* argv[])
{
    if (PIN_Init(argc, argv)) return Usage();
    trace = fopen("pinatrace.out", "w");
    INS_AddInstrumentFunction(Instruction, 0);
    PIN_AddFiniFunction(Fini, 0);
    // Never returns
    PIN_StartProgram();

    return 0;
}
```

Η αρχική μορφή του κώδικα κάνει ιχνηλάτηση μνήμης (memory trace) και εκτυπώνει τις εικονικές διευθύνσεις των εντολών που κάνουν ανάγνωση από την κύρια μνήμη ή εγγραφή σε αυτήν, καθώς και τις φυσικές διευθύνσεις στην κύρια μνήμη, στις οποίες αντιστοιχούν. Αυτό φαίνεται και στην πρώτη φωτογραφία, στην αρχή της υποενότητας.

3.3.2 Επεξεργασία του εργαλείου

Στα πλαίσια της παρούσης μεταπτυχιακής διατριβής, έπρεπε να γίνει επεξεργασία του εργαλείου, ούτως ώστε να εκτυπώνεται, στο αρχείο εξόδου (OUT file), η επιθυμητή πληροφορία. Η τελευταία αποτελείται από το χρόνο προσπέλασης των εντολών, καθώς και τον αριθμό σελίδας, στην οποία βρίσκεται η κάθε μία. Ειδικά η τελευταία πληροφορία έχει ιδιαίτερη σημασία, αφού έχει άμεση εφαρμογή στους αλγόριθμους αντικατάστασης σελίδας, για τους οποίους θα γίνει αναφορά σε επόμενο τμήμα της εργασίας:

```
// Print a memory read record
VOID RecordMemRead(VOID *time_stamp,VOID* ip,VOID* addr) {
    unsigned long long pageNo = (unsigned long long)addr/4096;
    fprintf(trace, "%llu %llu\n",(unsigned long long)time_stamp,(unsigned
long long)pageNo);
    }
// Print a memory write record
VOID RecordMemWrite(VOID *time_stamp,VOID* ip,VOID* addr) {
    unsigned long long pageNo = (unsigned long long)addr/4096;
    fprintf(trace, "%llu %llu\n",(unsigned long long)time_stamp,(unsigned
long long)pageNo);
    }
```

Όπως φαίνεται από το παραπάνω τμήμα κώδικα, έχουν γίνει ορισμένες μετατροπές. Τόσο στη συνάρτηση ανάγνωσης (`VOID RecordMemRead()`), όσο και σε αυτήν της εγγραφής (`VOID RecordMemWrite()`), έχει προστεθεί ως παράμετρος η μεταβλητή για το χρόνο προσπέλασης της κάθε εντολής (`VOID *time_stamp`). Επιπλέον, εντός και των δύο συναρτήσεων, έχει δηλωθεί η αντίστοιχη μεταβλητή για τον αριθμό σελίδας της κάθε εντολής (`unsigned long long pageNo`).

Η τιμή της προκύπτει από τη διαίρεση της εικονικής διεύθυνσης μνήμης με τον αριθμό 4096 (`(unsigned long long)addr/4096`). Ο συγκεκριμένος αριθμός αποτελεί το τυπικό μέγεθος, σε bytes, μίας σελίδας, στα σύγχρονα υπολογιστικά συστήματα.

Οι τιμές των μεταβλητών χρόνου προσπέλασης και αριθμού σελίδας είναι αυτές που εκτυπώνονται στο αρχείο «pinatrace.out», όταν καλούνται οι δύο συναρτήσεις.

```
190398183276834359289591
190399134601834359289591
190399148421134359289591
190399149609834359289591
190399150691634359289591
190399151752534359289591
190399152806134359289591
190399153862834172494443
190399156080434172494443
190399158404834172494444
190399159583234172494444
190399160689234172494444
190399468123734172494444
190399470982634172494443
190399502419834172494444
190399503922334172494443
190399887637834172494444
190400029424134172494443
190400053153934172494444
190400054621934172494443
```

Το αρχείο εξόδου (pinatrace.out), μετά την επεξεργασία των συναρτήσεων ανάγνωσης και εγγραφής. Στην πρώτη στήλη αριστερά, ο χρόνοι προσπέλασης για την κάθε εντολή (instruction), ενώ στη δεύτερη στήλη δεξιά, ο αριθμός σελίδας, στην εικονική μνήμη, στην οποία βρίσκεται η κάθε εντολή.

Επεξεργασία έχει γίνει και στο παρακάτω τμήμα κώδικα:

```
// Is called for every instruction and instruments reads and writes
VOID Instruction(INS ins, VOID* v)
{
    // Instruments memory accesses using a predicated call, i.e.
    // the instrumentation is called iff the instruction will actually be
executed.
    // On the IA-32 and Intel(R) 64 architectures conditional moves and REP
    // prefixed instructions appear as predicated instructions in Pin.
    UINT32 memOperands = INS_MemoryOperandCount(ins);
    // Iterate over each memory operand of the instruction.
    for (UINT32 memOp = 0; memOp < memOperands; memOp++)
    {
        if (INS_MemoryOperandIsRead(ins, memOp))
        {
            INS_InsertPredicatedCall(ins, IPOINT_BEFORE,
(AFUNPTR)RecordMemRead,IARG_TSC,IARG_INST_PTR, IARG_MEMORYOP_EA, memOp,
                                     IARG_END);
        }
        // Note that in some architectures a single memory operand can be
        // both read and written (for instance incl (%eax) on IA-32)
        // In that case we instrument it once for read and once for write.
        if (INS_MemoryOperandIsWritten(ins, memOp))
        {
            INS_InsertPredicatedCall(ins, IPOINT_BEFORE,
(AFUNPTR)RecordMemWrite,IARG_TSC,IARG_INST_PTR, IARG_MEMORYOP_EA, memOp,
                                     IARG_END);
        }
    }
}
```

Σε αυτό το τμήμα του κώδικα και στις δύο συνθήκες ελέγχου, έχει προστεθεί, στη συνάρτηση `INS_InsertPredicatedCall()`, μία επιπλέον παράμετρος, η `IARG_TSC`. Πρόκειται για τιμή μετρητή χρόνου προσπέλασης (Time Stamp Counter value), στο σημείο εισόδου της κλήσης της ανάλυσης.

4 Πραγματική εκτέλεση εφαρμογών μέσω του PIN

4.1 Εισαγωγή

Σε αυτό το τμήμα της εργασίας, γίνεται μια παρουσίαση της πραγματικής εκτέλεσης εφαρμογών, μέσω το εργαλείου «pinatrace.cpp» του Intel PIN. Η πληροφορία που περιέχεται στα αρχεία εξόδου αυτών των εφαρμογών χρησιμοποιείται ως είσοδος στα προγράμματα των αλγορίθμων αντικατάστασης σελίδας, ούτως ώστε να εξαχθεί πληροφορία για τη συμπεριφορά τους και να παρουσιαστούν τα αντίστοιχα αποτελέσματα και συμπεράσματα.

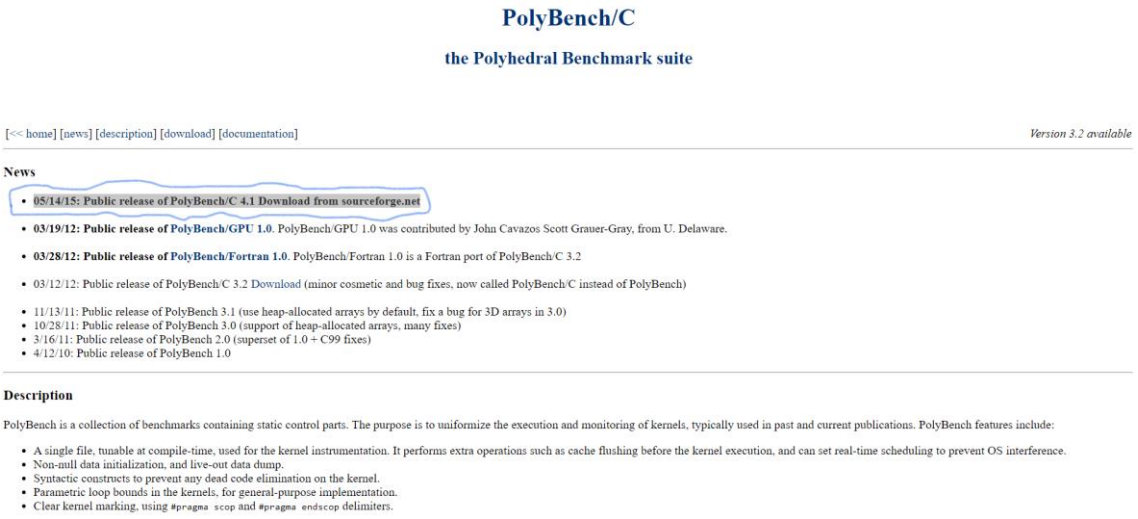
Οι εφαρμογές που χρησιμοποιήθηκαν ως μετροπρογράμματα (benchmarks) αφορούν στη συμπεριφορά των αλγορίθμων αντικατάστασης σελίδας, τόσο για regular όσο και για irregular access patterns στη μνήμη.

Πριν συνεχίσουμε, θα πρέπει πρώτα να γίνει μια επεξήγηση των όρων «memory access patterns», «regular memory access patterns» και «irregular memory access patterns». Memory access patterns είναι οι τρόποι ή τα μοτίβα με τα οποία ένα πρόγραμμα προσπελαύνει την κύρια μνήμη ενός Η/Υ. Με τον όρο «προσπέλαση» εννοούμε την εγγραφή, από ένα πρόγραμμα, στη μνήμη ή την ανάγνωση από αυτήν.

Regular memory access patterns είναι τα σταθερά και προβλέψιμα μοτίβα με τα οποία ένα πρόγραμμα προσπελαύνει τη μνήμη. Σε αυτήν την περίπτωση, η προσπέλαση στη μνήμη γίνεται με έναν τακτικό, επαναλαμβανόμενο τρόπο. Αντίθετα, τα irregular memory access patterns αναφέρονται σε απρόβλεπτους και ακανόνιστους τρόπους προσπέλασης της μνήμης, που χαρακτηρίζονται από τυχαιότητα.

4.2 Regular memory access patterns

Σε ό,τι αφορά τα regular memory access patterns, έγινε λήψη του σετ με benchmarks του «Polybench/C – the Polyhedral Benchmark Suite»[15]. Το σετ λαμβάνεται με τη μορφή συμπιεσμένου αρχείου.



PolyBench/C

the Polyhedral Benchmark suite

[<< home] [news] [description] [download] [documentation] *Version 3.2 available*

News

- **05/14/15: Public release of PolyBench/C 4.1 Download from sourceforge.net**
- **03/19/12: Public release of PolyBench/GPU 1.0.** PolyBench/GPU 1.0 was contributed by John Cavazos Scott Grauer-Gray, from U. Delaware.
- **03/28/12: Public release of PolyBench/Fortran 1.0.** PolyBench/Fortran 1.0 is a Fortran port of PolyBench/C 3.2
- 03/12/12: Public release of PolyBench/C 3.2 Download (minor cosmetic and bug fixes, now called PolyBench/C instead of PolyBench)
- 11/13/11: Public release of PolyBench 3.1 (use heap-allocated arrays by default, fix a bug for 3D arrays in 3.0)
- 10/28/11: Public release of PolyBench 3.0 (support of heap-allocated arrays, many fixes)
- 3/16/11: Public release of PolyBench 2.0 (superset of 1.0 + C99 fixes)
- 4/12/10: Public release of PolyBench 1.0

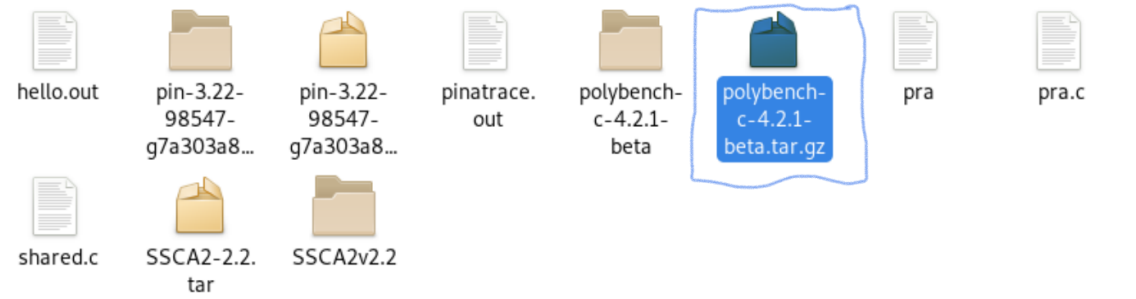
Description

PolyBench is a collection of benchmarks containing static control parts. The purpose is to uniformize the execution and monitoring of kernels, typically used in past and current publications. PolyBench features include:

- A single file, tunable at compile-time, used for the kernel instrumentation. It performs extra operations such as cache flushing before the kernel execution, and can set real-time scheduling to prevent OS interference.
- Non-null data initialization, and live-out data dump.
- Syntactic constructs to prevent any dead code elimination on the kernel.
- Parametric loop bounds in the kernels, for general-purpose implementation.
- Clear kernel marking, using `#pragma scop` and `#pragma endscop` delimiters.

Με κυκλική επισήμανση, η έκδοση προς λήψη, του σετ των polybench benchmarks. Ο χρήστης λαμβάνει το συμπιεσμένο αρχείο με ανακατεύθυνση στη σελίδα soulforge.net (Πηγή: https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/).

15. https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/.



Στη φωτογραφία, με μπλε επισήμανση, το συμπιεσμένο αρχείο του σετ εφαρμογών της "Polyhedral Benchmark Suite" ή "Polybench/C". Αριστερά του, ο αποσυμπιεσμένος φάκελος που περιέχει τις εφαρμογές – benchmarks.

Η εκτέλεση των εφαρμογών του «Polybench/C» μέσω του PIN έγινε επίσης σε περιβάλλον Linux (Debian 10 64-bit). Αφού ο χρήστης μεταβεί στον αποσυμπιεσμένο φάκελο των polybench benchmarks, θα πρέπει πρώτα να κάνει μεταγλώττιση (compiling) της εφαμοργής που τον ενδιαφέρει.

Έστω, για παράδειγμα, η «atax.c». Πρόκειται για εφαρμογή εντός του φακέλου «atax». Ο τελευταίος αποτελεί υποφάκελο του φακέλου «kernels», ο οποίος, με τη σειρά του, αποτελεί υποφάκελο του «linear-algebra», που βρίσκεται εντός του αποσυμπιεσμένου φακέλου του «Polybench/C».

Προκειμένου να γίνει μεταγλώττιση της εφαρμογής, ο χρήστης θα πρέπει να εισαγάγει, στο Linux shell, την ακόλουθη εντολή: `gcc -DMEDIUM_DATASET -I utilities -I linear-algebra/kernels/atax utilities/polybench.c linear-algebra/kernels/atax/atax.c -o atax_base.`

```
osboxes@osboxes:~/Desktop$ cd polybench-c-4.2.1-beta
osboxes@osboxes:~/Desktop/polybench-c-4.2.1-beta$ ls
2mm_base    CHANGELOG         fdtd-2d_base          linear-algebra  README          utilities
3mm_base    covariance_base   floyd-warshall_base   lu_base         seidel-2d_base
adi_base    datamining        jacobi-1d_base        medley          stencils
AUTHORS     doitgen_base      jacobi-2d_base        mvt_base        THANKS
bicg_base   durbin_base       LICENSE.txt           polybench.pdf   trisolv_base
osboxes@osboxes:~/Desktop/polybench-c-4.2.1-beta$ gcc -DMEDIUM_DATASET  -I utilities -I linear-alg
ebra/kernels/atax utilities/polybench.c linear-algebra/kernels/atax/atax.c -o atax_base
osboxes@osboxes:~/Desktop/polybench-c-4.2.1-beta$ ls
2mm_base    bicg_base         durbin_base           LICENSE.txt     polybench.pdf   trisolv_base
3mm_base    CHANGELOG         fdtd-2d_base          linear-algebra  README          utilities
adi_base    covariance_base   floyd-warshall_base   lu_base         seidel-2d_base
atax_base   datamining        jacobi-1d_base        medley          stencils
AUTHORS     doitgen_base      jacobi-2d_base        mvt_base        THANKS
```

Αφού ο χρήστης μεταβεί στον αποσυμπιεσμένο φάκελο των polybench benchmarks, θα πρέπει να εισαγάγει την υπογραμμισμένη εντολή, προκειμένου να γίνει μεταγλώττιση της εφαρμογής "atax.c". Δημιουργείται, με αυτόν τον τρόπο, το εκτελέσιμο "atax_base" (κυκλωμένο, κάτω αριστερά).

Όταν δημιουργηθεί το εκτελέσιμο «atax_base», θα πρέπει να γίνει αντιγραφή και επικόλλησή του στο φάκελο «ManualExamples», όπου βρίσκεται και το «pinatrace.cpp» του Intel PIN. Στη συνέχεια, ο χρήστης θα πρέπει να εκτελέσει την εφαρμογή μέσω του εργαλείου του PIN, προκειμένου να παραχθεί το αρχείο εξόδου για το «atax_base». Αυτό γίνεται με την εισαγωγή της ακόλουθης εντολής: `../../../pin -t obj-intel64/pinatrace.so -- ./atax_base`. Με αυτόν τον τρόπο, παράγεται το αρχείο εξόδου (pinatrace.out) για το «atax_base», το οποίο θα μπεί ως είσοδος στα προγράμματα για τους αλγόριθμους αντικατάστασης σελίδας.

```
osboxes@osboxes:~/Desktop/pin-3.22-98547-g7a303a835-gcc-linux/source/tools/ManualExamples$ ../../../pin -t obj-intel64/pinatrace.so -- ./atax_base
osboxes@osboxes:~/Desktop/pin-3.22-98547-g7a303a835-gcc-linux/source/tools/ManualExamples$ ls
2mm_base              fork_app.cpp       nonstatica.cpp
3mm_base              fork_jit_tool.cpp  obj-ia32
adi_base              hello.out          obj-intel64
atax_base             imageload.cpp      pinatrace.cpp
bicg_base             inscount0.cpp      pinatrace.out
```

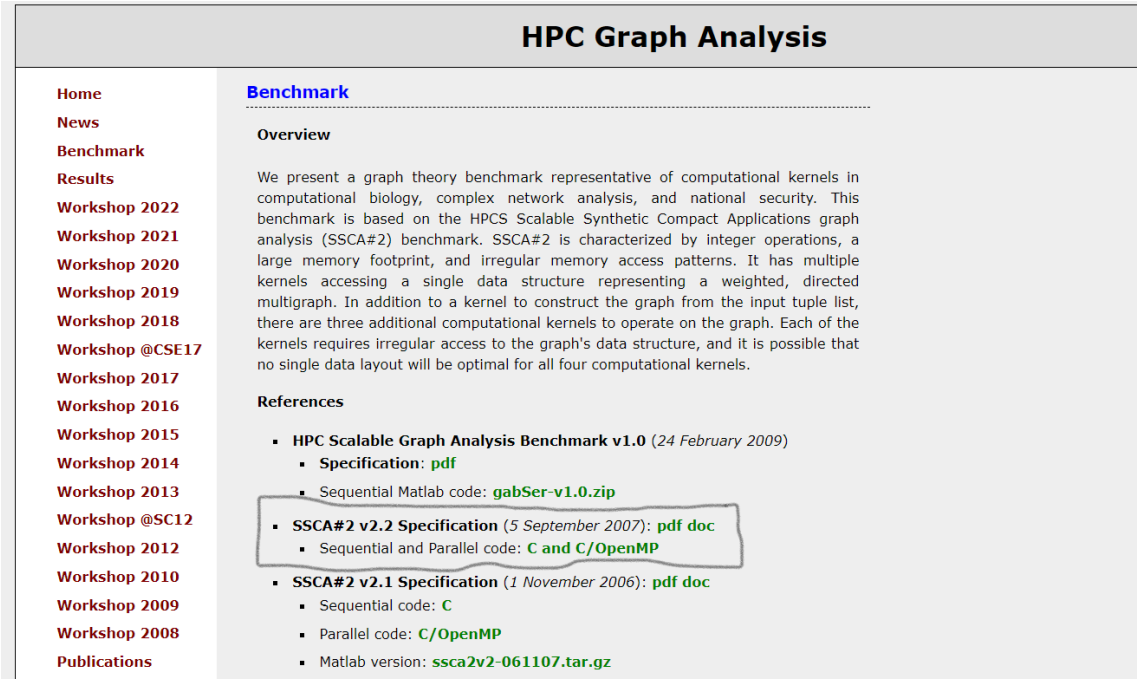
Με την εισαγωγή της υπογραμμισμένης εντολής, παράγεται το αρχείο εξόδου (pinatrace.out) για το "atax_base". Το αρχείο εξόδου θα μπει, με τη σειρά του, ως είσοδος στα προγράμματα των αλγορίθμων αντικατάστασης σελίδας.

Η ίδια διαδικασία με το παράδειγμα του «atax.c» ακολουθείται και για τις υπόλοιπες εφαρμογές του «Polybench/C».

4.3 Irregular memory access patterns

Στην προηγούμενη υποενότητα έγινε λεπτομερής αναφορά στην εκτέλεση πραγματικών εφαρμογών – μετροπρογραμμάτων (benchmarks) που σχετίζονται με regular access patterns στη μνήμη. Η παρούσα υποενότητα αναφέρεται στην εκτέλεση benchmarks για irregular memory access patterns.

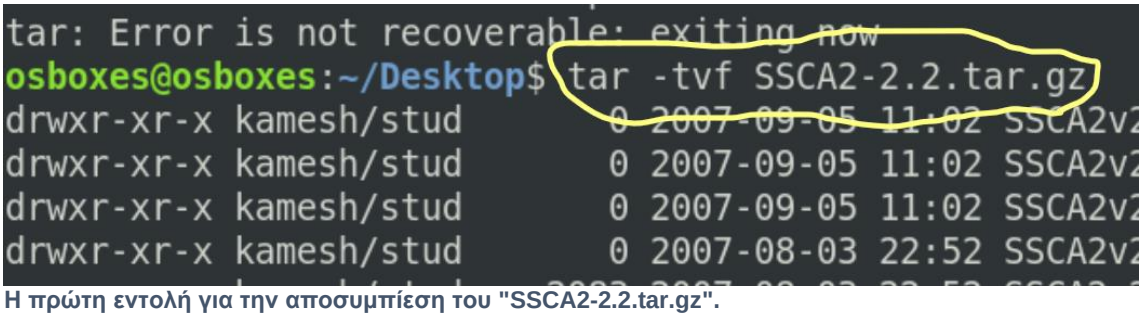
Για τη συγκεκριμένη εκτέλεση χρησιμοποιήθηκε η εφαρμογή «SSCA#2», συγκεκριμένα, η έκδοση 2.2 (SSCA#2 v2.2). Ο χρήστης, αφού επισκεφθεί τη σελίδα της εφαρμογής, με τίτλο «HPC Graph Analysis»[16], επιλέγει να κάνει λήψη του συμπιεσμένου αρχείου από το «SSCA#2 v2.2 Specification».



Η σελίδα "HPC Graph Analysis", από όπου ο χρήστης κάνει λήψη της έκδοσης 2.2 της εφαρμογής "SSCA#2" (Πηγή: http://www.graphanalysis.org/benchmark/).

Μετά τη λήψη του συμπιεσμένου αρχείου, γίνεται η αποσυμπίεση του. Αυτό μπορεί να γίνει με δύο τρόπους. Ο πρώτος τρόπος είναι να κάνει ο χρήστης right click πάνω στο αρχείο και να επιλέξει την εντολή «Extract Here».

Ο δεύτερος είναι μέσω της γραμμής εντολών του Linux (Linux shell). Ο χρήστης εισάγει πρώτα την εντολή: `tar -tvf SSCA2-2.2.tar.gz`. Μόλις εκτελεστεί, εισάγεται και η δεύτερη εντολή: `tar -xvf SSCA2-2.2.tar.gz`.

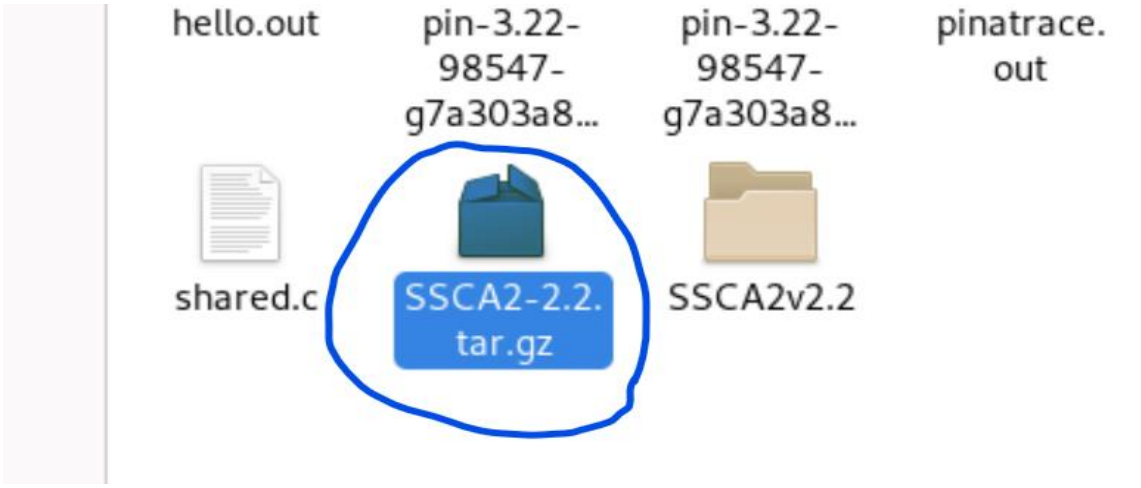


Η πρώτη εντολή για την αποσυμπίεση του "SSCA2-2.2.tar.gz".

16. http://www.graphanalysis.org/benchmark/.

```
-rw-r--r-- kamesh/stud    4178 2007-09-05 10:59 SSCA2v
osboxes@osboxes:~/Desktop$ tar -xvf SSCA2-2.2.tar.gz
SSCA2v2.2/
SSCA2v2.2/sprng2.0/
SSCA2v2.2/sprng2.0/lib/
SSCA2v2.2/sprng2.0/include/
```

Η δεύτερη εντολή για την αποσυμπίεση του "SSCA2-2.2.tar.gz".



Με κυκλική επισήμανση, το συμπιεσμένο αρχείο για το "SSCA#2". Δεξιά του, ο αποσυμπιεσμένος φάκελος.

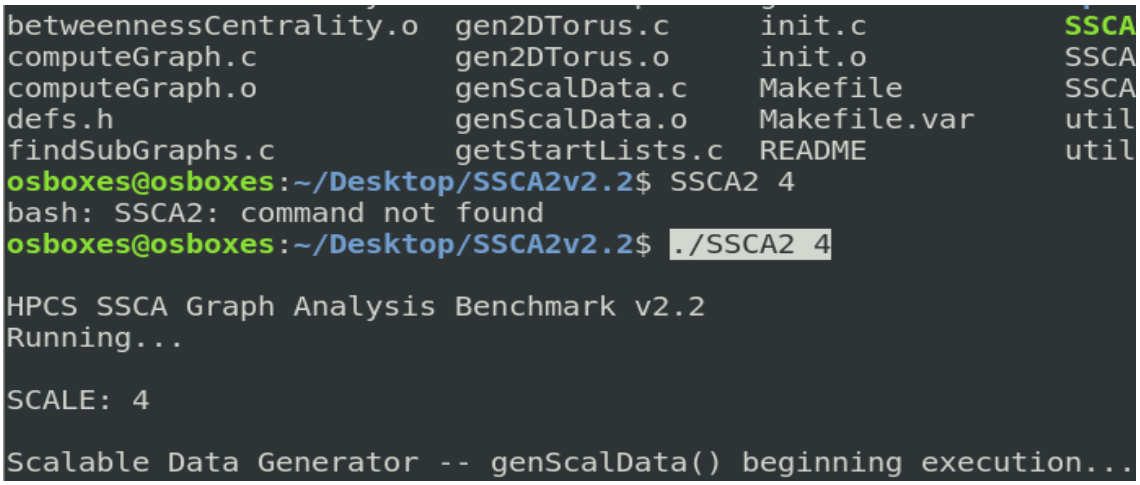
Μετά την αποσυμπίεση του αρχείου και αφού εισέλθει στον αποσυμπιεσμένο φάκελο «SSCA2v2.2», ο χρήστης εισάγει στο Linux shell, την εντολή make. Με τη συγκεκριμένη εντολή, δημιουργείται το εκτελέσιμο (SSCA2) του αρχείου «SSCA2.c», το οποίο βρίσκεται εντός του φακέλου.

```
betweennessCentrality.c  gen2DTorus.c     Makefile      SSCA2.c
computeGraph.c           genScalData.c    Makefile.var  utils.c
defs.h                   getStartLists.c  README
findSubGraphs.c          init.c           sprng2.0
osboxes@osboxes:~/Desktop/SSCA2v2.2$ make
(cd sprng2.0; make; cd ..)
make[1]: Entering directory '/home/osboxes/Desktop/SSCA2v2.2/sprng2.0'
gcc -c -O3 -DLittleEndian  primes_32.c
gcc -c -O3 -DLittleEndian  primes_64.c
gcc -c -O3 -DLittleEndian  -DAdd__ fwrap_mpi.c
gcc -c -O3 -DLittleEndian  -DAdd__ cputime.c
gcc -c -O3 -DLittleEndian  makeseed.c
```

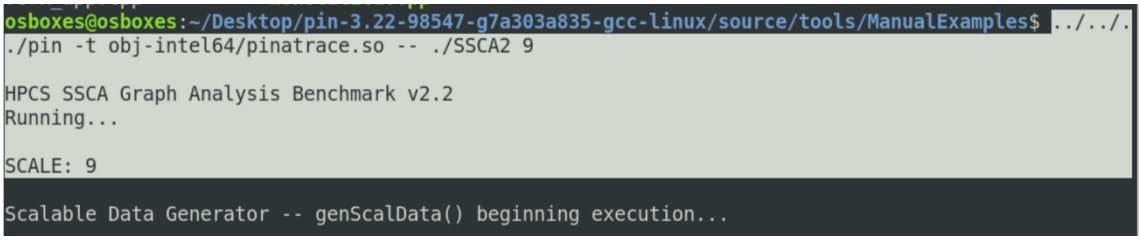
Εντός του φακέλου "SSCA2v2.2", ο χρήστης εισάγει την εντολή "make", για τη δημιουργία του εκτελέσιμου (SSCA2) για το "SSCA2.c".

Το «SSCA2.c» είναι ένας αλγόριθμος γραφημάτων. Κατά συνέπεια, ο χρήστης θα πρέπει να εκτελέσει το «SSCA2» για έναν ορισμένο αριθμό κόμβων. Αυτό επιτυγχάνεται για παράδειγμα, με την ακόλουθη εντολή: `./SSCA2 4`. Ο ακέραιος αριθμός «4» είναι μία παράμετρος που αφορά στον αριθμό των κόμβων.

Στο συγκεκριμένο παράδειγμα, η εκτέλεση του «SSCA2» γίνεται για ένα γράφημα με $2^4 = 16$ κόμβους. Αντίστοιχα, αν εισαχθεί ο ακέραιος αριθμός «8» ως παράμετρος, θα γίνει εκτέλεση για γράφημα με $2^8 = 256$ κόμβους. Για τις ανάγκες της παρούσης εργασίας, επελέγη ως παράμετρος ο ακέραιος «9». Έγινε δηλαδή, εκτέλεση για γράφημα $2^9 = 512$ κόμβων. Προκειμένου να δημιουργηθεί το αρχείο εξόδου του εκτελέσιμου για τον επιθυμητό αριθμό κόμβων, εισήχθη η ακόλουθη εντολή: `../../../pin -t obj-intel64/pinatrace.so -- ./SSCA2 9`. Το αρχείο εξόδου είναι πλέον έτοιμο να μπει ως είσοδος στους αλγορίθμους αντικατάστασης σελίδας.



Το "SSCA2.c" είναι ένας αλγόριθμος γραφημάτων. Ο χρήστης θα πρέπει να το εκτελέσει για έναν ορισμένο αριθμό κόμβων. Αυτό επιτυγχάνεται με την εντολή "./SSCA2 4" που έχει επισημανθεί στη φωτογραφία. Ο ακέραιος αριθμός "4" είναι παράμετρος που αφορά στον αριθμό των κόμβων.



Για τις ανάγκες της εργασίας, επελέγη ως παράμετρος ο ακέραιος αριθμός "9". Έγινε δηλαδή, εκτέλεση για γράφημα $2^9 = 512$ κόμβων. Προκειμένου να δημιουργηθεί το αρχείο εξόδου του εκτελέσιμου για τον επιθυμητό αριθμό κόμβων, εισήχθη η εντολή: ../../../pin -t obj-intel64/pinatrace.so -- ./SSCA2 9.

5 Το πρόγραμμα PRA

5.1 Εισαγωγή

Κάθε διαδικασία έχει το δικό της εικονικό χώρο διευθύνσεων (virtual address space), για κάθε εντολή της (instruction). Στα σύγχρονα υπολογιστικά συστήματα, με πολλές διαδικασίες να εκτελούνται ταυτόχρονα, είναι πρακτικά ανέφικτο να «φορτώνεται» ολόκληρη η καθεμία από αυτές τις διαδικασίες στη φυσική μνήμη (RAM), καθώς η τελευταία είναι περιορισμένη έναντι του εικονικού χώρου διευθύνσεων. Η λύση στο συγκεκριμένο πρόβλημα δόθηκε με τη σύλληψη της ιδέας ότι μια διαδικασία δε χρειάζεται να βρίσκεται ολόκληρη στη φυσική, κύρια μνήμη, παρά μόνο το τμήμα της που είναι απαραίτητο μια δεδομένη χρονική στιγμή. Για αυτό το λόγο, ο εικονικός χώρος διευθύνσεων χωρίζεται σε μικρότερα, ίσου μεγέθους τμήματα, τις «σελίδες» (pages). Οι σελίδες που χρειάζεται να εκτελεστούν μια συγκεκριμένη χρονική στιγμή βρίσκονται στην κύρια μνήμη, ενώ οι υπόλοιπες στη δευτερεύουσα (σκληρός δίσκος). Το τυπικό μέγεθος μιας σελίδας, στα σύγχρονα υπολογιστικά συστήματα, είναι 4096 bytes ή 4KB.

Όταν η Κεντρική Μονάδα Επεξεργασίας (επεξεργαστής) χρειαστεί μια σελίδα, η οποία δεν υπάρχει στην κύρια μνήμη, διακόπτει τη διαδικασία και εμφανίζεται το «σφάλμα σελίδας» (page fault). Το ΛΣ μπλοκάρει τη διαδικασία και θα πρέπει να φέρει την επιθυμητή σελίδα στη μνήμη, ώστε να υπάρχει επανεκκίνηση της διαδικασίας. Το ΛΣ θα αναζητήσει την απαιτούμενη σελίδα στον εικονικό χώρο διευθύνσεων. Μόλις η σελίδα τοποθετηθεί στην κύρια μνήμη, το ΛΣ δίνει σήμα στην ΚΜΕ, ώστε να συνεχιστεί η εκτέλεση της διαδικασίας.

Η κύρια μνήμη χωρίζεται επίσης σε μικρότερα τμήματα ίσου μεγέθους μεταξύ τους, αλλά και με τις σελίδες του εικονικού χώρου διευθύνσεων. Αυτά τα τμήματα ονομάζονται «πλαίσια σελίδων» (page frames). Όταν η ζητούμενη σελίδα δε βρίσκεται εντός της κύριας μνήμης, με αποτέλεσμα να προκύπτει σφάλμα σελίδας, αναζητείται και λαμβάνεται από το σκληρό δίσκο, προκειμένου να εισέλθει στο διαθέσιμο πλαίσιο σελίδας. Αν βρίσκεται ήδη σε ένα πλαίσιο σελίδας, τοτέ δεν υφίσταται σφάλμα (ορίζεται ως «page hit»).

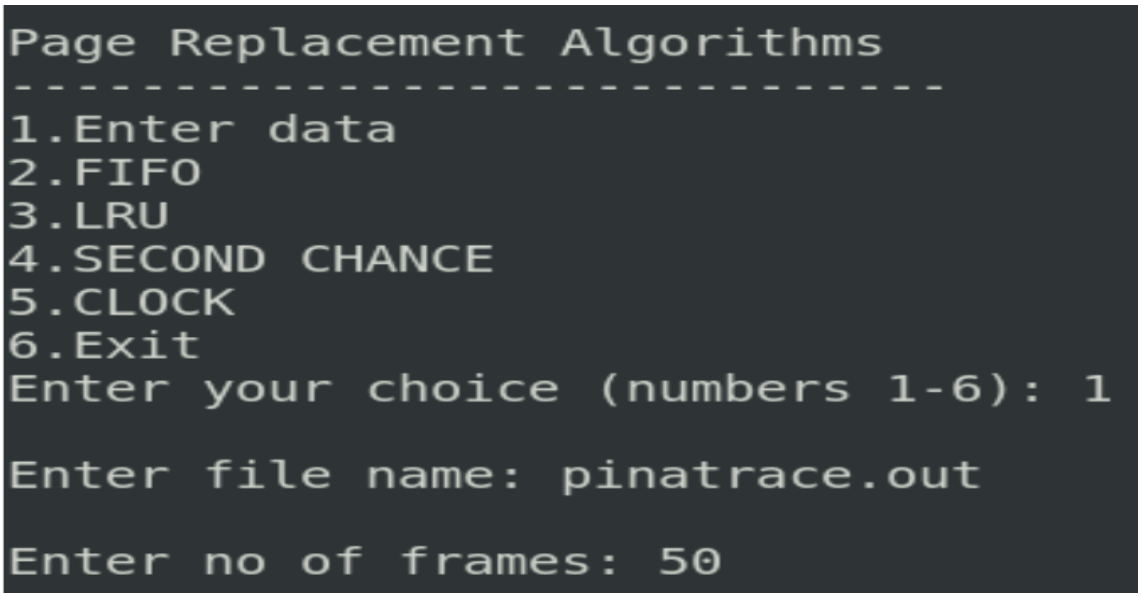
Υπάρχει πιθανότητα η ζητούμενη, σε μια δεδομένη χρονική στιγμή, σελίδα να μη μπορεί να μπει σε κάποιο πλαίσιο, αν όλα περιέχουν ήδη μια σελίδα. Σε αυτήν την περίπτωση, προκύπτει επίσης σφάλμα σελίδας και αναλαμβάνουν οι αλγόριθμοι αντικατάστασης σελίδας (page replacement algorithms). Κάθε τέτοιος αλγόριθμος έχει τον δικό του τρόπο, βάσει του οποίου αποφασίζει ποια από τις σελίδες στην κύρια μνήμη θα πρέπει να αντικατασταθεί, προκειμένου να «φορτωθεί», στη θέση της (στο δικό της πλαίσιο σελίδας), η νέα σελίδα που έρχεται από τη δευτερεύουσα μνήμη. Η επιτυχία του κάθε αλγορίθμου αντικατάστασης σελίδας, καθώς και η υπεροχή του έναντι των υπολοίπων, εξαρτάται από τη διατήρηση, σε όσο πιο χαμηλό επίπεδο, του αριθμού σφαλμάτων σελίδας.

5.2 Περιγραφή του προγράμματος PRA

Το πρόγραμμα PRA υλοποιήθηκε σε γλώσσα C. Η πλήρης ονομασία του είναι «Page Replacement Algorithms», αναφέρεται όμως με τα αρχικά του, χάριν συντομίας. Πρόκειται ουσιαστικά, για ένα μενού επιλογών, μέσω του οποίου γίνεται προσομοίωση της λειτουργίας των αλγορίθμων αντικατάστασης σελίδας, για ένα δεδομένο αριθμό πλαισίων σελίδας, τα οποία ορίζει ο χρήστης. Προκειμένου να γίνει μεταγλώττιση του πηγαίου κώδικα (pra.c) στο εκτελέσιμο αρχείο (pra), ο χρήστης θα πρέπει να εισαγάγει την εντολή `gcc -O3 -mcmodel=medium -o pra pra.c`, ενώ για να την εκκίνηση του εκτελέσιμου απαιτείται η εντολή `./pra`.

Τα βήματα που ακολουθεί ο χρήστης στη συγκεκριμένη προσομοίωση είναι τα εξής: πρώτα, εισάγει ως αρχείο εισόδου, το αρχείο εξόδου που παράγεται με τη βοήθεια του εργαλείου «pinatrace.cpp» της Intel (pinatrace.out). Όπως έχει αναφερθεί και σε προηγούμενο σημείο της εργασίας[17], το «pinatrace.cpp» έχει τροποποιηθεί, προκειμένου στο αρχείο εξόδου να εμφανίζεται ο αριθμός σελίδας (page number), για κάθε εντολή. Αυτό είναι απαραίτητο, προκειμένου να ξεκινήσει η προσομοίωση. Στη συνέχεια, ο χρήστης θα πρέπει να πληκτρολογήσει έναν αριθμό πλαισίων σελίδας. Εντός ενός κενού πλαισίου σελίδας, θα εισέλθει ένας αντίστοιχος αριθμός σελίδας.

Στη συνέχεια, ο χρήστης θα πρέπει να επιλέξει τον αλγόριθμο με τον οποίο θα γίνει η προσομοίωση αντικατάστασης σελίδας. Τέλος, με την ολοκλήρωση της προσομοίωσης, ο χρήστης επιλέγει να εξέλθει της εφαρμογής PRA. Επειδή η εργασία αφορά στη μελέτη και, κατά συνέπεια, σύγκριση αλγορίθμων αντικατάστασης σελίδας, η προσομοίωση θα πρέπει να γίνει για όλους του διαθέσιμους αλγορίθμους του προγράμματος. Υπάρχουν πολλοί, διαφορετικοί αλγόριθμοι αντικατάστασης σελίδας. Για τις ανάγκες της παρούσης διατριβής όμως, έχουν επιλεγεί οι εξής τέσσερις: ο αλγόριθμος FIFO (First In-First Out), ο LRU (Least Recently Used), ο αλγόριθμος Δεύτερης Ευκαιρίας (Second Chance) και ο αλγόριθμος Ρολογιού (Clock).



Το μενού επιλογών της εφαρμογής PRA. Η επιλογή 1 είναι η εισαγωγή δεδομένων από τον χρήστη, δηλαδή το αρχείο εξόδου του Intel PIN που εισάγεται ως είσοδος και ο αριθμός πλαισίων σελίδας. Οι επιλογές 2-5 αφορούν στην εκτέλεση το αρχείου εισόδου για τέσσερις διαφορετικούς αλγορίθμους αντικατάστασης σελίδας. Με την επιλογή 6, ο χρήστης εξέρχεται του προγράμματος.

17. Υποενότητα 3.3.2: «Επεξεργασία του εργαλείου», σελ. 19.

5.3 Οι αλγόριθμοι αντικατάστασης σελίδας του προγράμματος PRA

5.3.1 Ο Αλγόριθμος FIFO (First In-First Out)

Ο αλγόριθμος αντικατάστασης σελίδας FIFO[18,19,20] είναι ο πρώτος από τους τέσσερις αλγορίθμους που περιλαμβάνονται στο πρόγραμμα PRA της παρούσης μεταπτυχιακής διατριβής. Πρόκειται για τον πιο απλό από τους τέσσερις.

Στην περίπτωση του FIFO, το λειτουργικό σύστημα διατηρεί στη μνήμη του Η/Υ μια συνδεδεμένη λίστα των σελίδων. Στην αρχή της βρίσκεται η παλαιότερη σελίδα, δηλαδή αυτή που βρίσκεται στη λίστα το μεγαλύτερο χρονικό διάστημα. Όταν προκύψει σφάλμα σελίδας και δεν υπάρχει ελεύθερο πλαίσιο σελίδας, τότε η σελίδα στην αρχή-κεφαλή της λίστας αφαιρείται ενώ η τελευταία χρονικά σελίδα τοποθετείται στο τέλος-ουρά της λίστας.

Ακολουθεί ένα πράδειγμα με αριθμητικά δεδομένα. Έστω ότι, μια δεδομένη χρονική στιγμή, έχουμε μια λίστα με τις σελίδες 5, 4 και 2 και τρία (3) διαθέσιμα πλαίσια. Η επόμενη σελίδα που ζητείται από τον επεξεργαστή είναι η 5. Η συγκεκριμένη σελίδα υπάρχει ήδη σε πλαίσιο της κύριας μνήμης, οπότε δε γίνεται αντικατάσταση (page hit). Ας υποθέσουμε ότι ακολούθως ζητείται η σελίδα 1. Δεν υπάρχει στην κύρια μνήμη και θα πρέπει το ΛΣ να την «τραβήξει» από τη δευτερεύουσα μνήμη, δηλαδή το σκληρό δίσκο. Στη συγκεκριμένη περίπτωση, προκύπτει σφάλμα σελίδας και επιπλέον, κανένα πλαίσιο δεν είναι διαθέσιμο. Συνεπώς, θα πρέπει κάποια από τις σελίδες εντός των πλαισίων να αντικατασταθεί από τη νέα σελίδα 1. Σύμφωνα με την αρχή λειτουργίας του αλγορίθμου FIFO, η προς αντικατάσταση σελίδα είναι η 5 και αυτό επειδή είναι η πρώτη σελίδα που εμφανίστηκε και τοποθετήθηκε σε πλαίσιο της κύριας μνήμης, άρα και η χρονικά παλαιότερη.

Για να γίνει ακόμα πιο κατανοητή η συμπεριφορά του FIFO, ακολουθεί ένα παράδειγμα από την καθημερινότητα. Έστω ότι μία τράπεζα διαθέτει στο υποκατάστημά της έναν αριθμό ταμείων για την εξυπηρέτηση των πελατών της. Ας υποθέσουμε ότι υπάρχει μια x ουρά πελατών η οποία μια δεδομένη χρονική στιγμή είναι πλήρης. Όταν έρθει ένας νέος πελάτης να εξυπηρετηθεί, θα σταθεί στο τέλος της ουράς ενώ, ο παλαιότερος χρονικά πελάτης, στην αρχή της, θα προχωρήσει στο πρώτο διαθέσιμο ταμείο. Ο τελευταίος χρονικά πελάτης, στο τέλος της ουράς, αντιστοιχεί στην τελευταία χρονικά σελίδα στο τέλος της λίστας. Παρομοίως, η παλαιότερη σελίδα, στην αρχή της λίστας, αντιστοιχεί στον πελάτη που βρίσκεται στην αρχή της ουράς, άρα και για το μεγαλύτερο, χρονικά, διάστημα.

Ο κώδικας για τον αλγόριθμο FIFO είναι ο εξής:

```
void fifo() {
  unsigned long long i, j;
  initialize(); //Initialization function.
  for (i = 0; i < records; i++) {
    if (isHit(data[i].pageNo) == 0) { //FIFO acts as a queue. The page number
//which has come first is out when queue is full. The new page number is
//placed at the end of the queue.
      for (j = 0; j < nf - 1; j++) {
        frames[j] = frames[j + 1];
      }
      frames[j] = data[i].pageNo;
      pgfaultcnt++;
    } else {
      //printf("Page Hit");
    }
  }
  dispPgFaultCnt();
}
```

18. Andrew S. Tanenbaum, «Σύγχρονα Λειτουργικά Συστήματα», 3η αμερικανική έκδοση, επιστημονική επιμέλεια ελληνικής έκδοσης Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών, εκδόσεις «Κλειδάριθμος», σελ. 260.
19. Kartik Moudgil, Anushka Ringshia, Harshal Bharatkumar Parekh, Ria Maheshwari, Sheetal Chaudhari, «Systematic Analysis and Simulation of Classical Page Replacement Algorithms: A Proposed Novel WSClock-Derivative», 2017 2nd International Conference for Convergence in Technology (I2CT), p. 888.
20. Hitha Paulson, Dr. Rajesh Ramachandran, «Page Replacement Algorithms-Challenges and Trends», International Journal of Computer & Mathematical Sciences, Volume 6, Issue 9, September 2017, p. 113.

5.3.2 Ο Αλγόριθμος LRU (Least Recently Used)

Ο επόμενος αλγόριθμος της εφαρμογής PRA είναι ο LRU (Least Recently Used)[21,22,23]. Ο συγκεκριμένος αλγόριθμος αντικατάστασης σελίδας διαφέρει από τον FIFO, που αναλύθηκε παραπάνω. Όπως δηλώνει και το όνομά του, αφορά στην αντικατάσταση, σε περίπτωση σφάλματος σελίδας με πλήρη πλαίσια, της σελίδας που χρησιμοποιείται λιγότερο συχνά.

Ο τρόπος λειτουργίας του LRU έχει ως εξής: έστω ότι, όπως και στην περίπτωση του FIFO, το ΛΣ διατηρεί στη μνήμη του Η/Υ μια συνδεδεμένη λίστα σελίδων. Λαμβάνεται ως δεδομένο ότι κανένα πλαίσιο σελίδας δεν είναι διαθέσιμο. Όταν έρθει η στιγμή να εισέλθει μια νέα σελίδα στην κύρια μνήμη, προκύπτει σφάλμα σελίδας. Το ερώτημα είναι ποιά από τις ήδη υπάρχουσες σελίδες στη μνήμη θα αντικατασταθεί; Σύμφωνα με την αρχή λειτουργίας του LRU, θα αφαιρεθεί από το πλαίσιο, στο οποίο βρίσκεται, η σελίδα που δεν έχει χρησιμοποιηθεί για το μεγαλύτερο χρονικό διάστημα και τη θέση της στο πλαίσιο θα πάρει η νέα σελίδα.

Για να γίνει ακόμα πιο κατανοητή η συμπεριφορά του LRU, ακολουθεί και ένα παράδειγμα. Ας υποθέσουμε ότι μια δεδομένη χρονική στιγμή υπάρχουν τρεις σελίδες στην κύρια μνήμη, με διαθέσιμα τρία πλαίσια. Οι σελίδες είναι οι 5, 4, 2. Έστω ότι, η σελίδα που ακολουθεί είναι ξανά η 5. Σε αυτήν την περίπτωση, δεν πρόκειται να γίνει καμία αλλαγή, καθώς, εφόσον η 5 υπάρχει ήδη στη μνήμη, προκύπτει επιτυχία (page hit).

Ας υποθέσουμε ότι η επόμενη σελίδα που ζητείται απο τον επεξεργαστή είναι η 1. Η συγκεκριμένη σελίδα δεν υπάρχει στην κύρια μνήμη, οπότε το ΛΣ τη φέρνει από τον σκληρό δίσκο. Τώρα προκύπτει σφάλμα σελίδας. Δεδομένου ότι οι υφιστάμενες σελίδες καταλαμβάνουν όλα τα διαθέσιμα πλαίσια, κάποια από αυτές θα πρέπει να αντικατασταθεί. Κινούμενοι με φορά προς τα αριστερά, θα πρέπει να δούμε ποιά από τις σελίδες χρησιμοποιήθηκε λιγότερο πρόσφατα. Αυτή είναι η 4, της οποίας τη θέση θα πάρει η σελίδα 1.

Παρακάτω, ο κώδικας για τον αλγόριθμο LRU:

```
unsigned long long findLRU(unsigned long long time[], unsigned long long
records) {
  unsigned long long i, minimum = time[0], pos = 0;
  for (i = 1; i < records; ++i) { //Find the least recently used page number.
The one with the least access time
    if (time[i] < minimum) {
      minimum = time[i]; //Least access time
      pos = i;
    }
  }
  return pos;
}
void lru() {
  unsigned long long i, j, counter = 0, flag1, flag2;
  initialize(); //Initialization function.
  for (i = 0; i < records; i++) {
    flag1 = flag2 = 0;
    for (j = 0; j < nf; j++) {
      if (frames[j] == data[i].pageNo) {
        counter++;
        time[j] = counter;
        flag1 = flag2 = 1;
        break;
      }
    }
  /*When a new page comes and we have empty frames, we put the page in the
respective frame, mark the access time and increase number of page faults.*/
    if (flag1 == 0) {
      for (j = 0; j < nf; j++) {
        if (frames[j] == UNUSED_FRAME) {
          counter++;
          pgfaultcnt++;
```

```
          frames[j] = data[i].pageNo;
          time[j] = counter;
          flag2 = 1;
          break;
        }
      }
    }
  //When a new page comes and all frames are full, we apply the LRU
//algorithm.
    if (flag2 == 0) {
      pos = findLRU(time, nf);
      counter++;
      pgfaultcnt++;
      frames[pos] = data[i].pageNo;
      time[pos] = counter;
      //dispPages();
    }
  }
  dispPgFaultCnt();
}
```

21. Andrew S. Tanenbaum, «Σύγχρονα Λειτουργικά Συστήματα», 3η αμερικανική έκδοση, επιστημονική επιμέλεια ελληνικής έκδοσης Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών, εκδόσεις «Κλειδάριθμος», σελ. 262.
22. Kartik Moudgil, Anushka Ringshia, Harshal Bharatkumar Parekh, Ria Maheshwari, Sheetal Chaudhari, «Systematic Analysis and Simulation of Classical Page Replacement Algorithms: A Proposed Novel WSClock-Derivative», 2017 2nd International Conference for Convergence in Technology (I2CT), p. 890.
23. Hitha Paulson, Dr. Rajesh Ramachandran, «Page Replacement Algorithms-Challenges and Trends», International Journal of Computer & Mathematical Sciences, Volume 6, Issue 9, September 2017, p. 113.

5.3.3 Ο Αλγόριθμος Δεύτερης Ευκαιρίας (Second Chance)

Ο τρίτος, κατά σειρά, αλγόριθμος αντικατάστασης σελίδας της εφαρμογής PRA είναι ο Second Chance[24,25,26] ή Αλγόριθμος Δεύτερης Ευκαιρίας. Σε ό,τι αφορά τη λειτουργία του, μοιάζει με τον αλγόριθμο FIFO. Η διαφορά έγκειται στο ότι ο Second Chance περιλαμβάνει και ένα «bit αναφοράς» (reference ή used bit). Πριν γίνει αντικατάσταση σελίδας, ελέγχεται αν το bit αναφοράς της υποψήφιας προς αντικατάσταση σελίδας βρίσκεται στη θέση 1 ή 0. Στην περίπτωση που το bit βρίσκεται στη θέση 0, η υπάρχουσα σελίδα αντικαθίσταται από τη νέα. Αν όμως, το bit αναφοράς είναι στη θέση 1, η υφιστάμενη σελίδα έχει μια «δεύτερη ευκαιρία», το bit «σβήνει» (επιστρέφει δηλαδή, στη θέση 0) και αναζητείται η επόμενη σελίδα, σύμφωνα με τον αλγόριθμο FIFO.

Ακολουθεί και εδώ ένα παράδειγμα: όπως και στην περίπτωση του LRU, έχουμε και εδώ μια λίστα σελίδων με τους ίδιους αριθμούς: 5, 4, και 2. Επίσης, υπάρχουν τα ίδια τρία πλαίσια σελίδας, που τις περιέχουν. Κάθε μια από τις σελίδες έχει το δικό της reference bit στη θέση 0. Έστω ότι και σε αυτήν την περίπτωση, ζητείται από τον επεξεργαστή ξανά η σελίδα 5. Η συγκεκριμένη σελίδα υπάρχει ήδη σε ένα από τα τρία πλαίσια της κύριας μνήμης (page hit).

Σε αυτήν την περίπτωση, το reference bit της σελίδας αλλάζει στη θέση 1. Ας υποθέσουμε ότι η επόμενη σελίδα που ζητείται από τον επεξεργαστή είναι η σελίδα 3. Καθώς, όπως φαίνεται, αυτή η σελίδα δεν υπάρχει σε κάποιο πλαίσιο της κύριας μνήμης, θα πρέπει και σε αυτήν την περίπτωση, το ΛΣ να την ανασύρει από το σκληρό δίσκο. Τώρα προκύπτει σφάλμα σελίδας. Η σελίδα αυτή δεν υπάρχει στην κύρια μνήμη και επιπλέον κανένα πλαίσιο δεν είναι διαθέσιμο.

Σύμφωνα με τον αλγόριθμο FIFO, κανονικά θα έπρεπε από τις τρεις υπάρχουσες σελίδες να αφαιρεθεί η παλαιότερη, χρονικά, σελίδα ή με άλλα λόγια, αυτή που εμφανίστηκε πρώτη. Η ζητούμενη σελίδα θα ήταν η 5.

Στον Second Chance όμως, τη διαφορά την κάνει το reference bit. Εφόσον η σελίδα 5 έχει το reference bit της στη θέση 1, της δίνεται δεύτερη ευκαιρία: το reference bit που της αντιστοιχεί θα γυρίσει στη θέση 0 και θα αναζητηθεί η επόμενη υποψήφια σελίδα, όπως ακριβώς με τον αλγόριθμο FIFO. Συνεπώς, θα αντικατασταθεί η σελίδα 4, λόγω του ότι είναι η αμέσως παλαιότερη σελίδα μετά την 5 και το reference bit της βρίσκεται στη θέση 0.

Ο κώδικας για τον αλγόριθμο Second Chance:

```
void secondchance() {
  unsigned long long i, victimptr = 0;
  initialize(); //Initialization function.
  for (i = 0; i < nf; i++) {
    usedbit[i] = 0; //Initial value of used (or reference) bit is zero(0).
  }
  for (i = 0; i < records; i++) {
    if (isHit(data[i].pageNo)) {
      int hitindex = getHitIndex(data[i].pageNo);
      if (usedbit[hitindex] == 0) {
        usedbit[hitindex] = 1;
      }
    } else {
      pgfaultcnt++;
      if (usedbit[victimptr] == 1) {
        do {
          usedbit[victimptr] = 0;
          victimptr++;
          if (victimptr == nf) {
            victimptr = 0;
         }
        } while (usedbit[victimptr] != 0);
      }
      if (usedbit[victimptr] == 0) {
        frames[victimptr] = data[i].pageNo;
        victimptr++;
      }
```

```
    }
    if (victimptr == nf) {
      victimptr = 0;
    }
  }
  dispPgFaultCnt();
}
```

24. Andrew S. Tanenbaum, «Σύγχρονα Λειτουργικά Συστήματα», 3η αμερικανική έκδοση, επιστημονική επιμέλεια ελληνικής έκδοσης Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών, εκδόσεις «Κλειδάριθμος», σελ. 261.
25. Kiran Muhammad Saleem, Maria Iqbal, Hadia Saadi, Farheen Qazi, Dur-e-Shawar Agha, «Second Chance Page Replacement Algorithm with Optimal (SCAO)», 2019 International Conference on Information Science and Communication Technology (ICISCT), p. 2.
26. Hitha Paulson, Dr. Rajesh Ramachandran, «Page Replacement Algorithms-Challenges and Trends», International Journal of Computer & Mathematical Sciences, Volume 6, Issue 9, September 2017, p. 114.

5.3.4 Ο Αλγόριθμος Ρολογιού (Clock)

Ακολουθεί ο τέταρτος κατά σειρά και τελευταίος αλγόριθμος της PRA, ο Αλγόριθμος Ρολογιού (Clock)[27,28,29]. Ο Clock προσεγγίζει και αυτός τον αλγόριθμο FIFO, σε ό,τι αφορά τον τρόπο λειτουργίας του. Επιπλέον, όπως και ο Second Chance, χρησιμοποιεί το reference ή used bit για κάθε σελίδα εντός πλαισίου στην κύρια μνήμη, με λίγο διαφορετικό τρόπο όμως. Η διαφορά έγκειται στο ότι, το reference bit αλλάζει από τη θέση 0 στη θέση 1 με την πρώτη εμφάνιση μιας σελίδας στην κύρια μνήμη.

Για την καλύτερη κατανόηση του αλγορίθμου ρολογιού, ακολουθεί το παράδειγμα που χρησιμοποιήθηκε και στους προηγούμενους αλγορίθμους: έστω ότι και σε αυτήν την περίπτωση έχουμε τη λίστα με τις σελίδες 5, 4 και 2, με τρία πλαίσια σελίδας. Όπως αναφέρθηκε και παραπάνω, με την πρώτη εμφάνιση τους, το reference bit της κάθε σελίδας αλλάζει από την κατάσταση 0 στην 1. Αφού έχουν καλυφθεί όλα τα πλαίσια, επιστρέφουμε ξανά στο πλαίσιο με τη σελίδα 5. Ας υποθέσουμε ότι η επόμενη σελίδα που ζητά το ΛΣ είναι ξανά η σελίδα 5. Όπως βλέπουμε, η σελίδα υπάρχει ήδη σε πλαίσιο της μνήμης οπότε δε χρειάζεται να γίνει καμία αλλαγή (page hit).

Έστω ότι η επόμενη σελίδα που ζητείται είναι η 1. Αυτή δεν υπάρχει σε κανένα πλαίσιο μνήμης, άρα θα πρέπει να τη φέρει το ΛΣ από το σκληρό δίσκο στην κύρια μνήμη. Όπως γνωρίζουμε και από τα παραδείγματα των παραπάνω αλγορίθμων, έχουμε και εδώ σφάλμα σελίδας. Όλα τα πλαίσια στη μνήμη όμως είναι μη διαθέσιμα. Σε αυτήν την περίπτωση, το reference bit της σελίδας 5 θα μπει θα μπει στην κατάσταση 0 και θα γίνει αναζήτηση στο επόμενο πλαίσιο της σελίδας 4.

Θα γίνει και εδώ αλλαγή του bit σε 0 και θα έχουμε μετάβαση στη σελίδα 2, όπου και εδώ θα γίνει η ίδια μετατροπή του αντίστοιχου bit. Η αναζήτηση θα επαναληφθεί από το πρώτο πλαίσιο, που περιέχει τη σελίδα 5. Τώρα όμως, το reference bit είναι ήδη στη θέση 0. Σε αυτό το σημείο, έχουμε εφαρμογή του αλγορίθμου FIFO. Δεδομένου ότι η σελίδα 5 είναι η πρώτη που εμφανίστηκε στη μνήμη, άρα και η παλαιότερη χρονικά, θα αντικατασταθεί με τη σελίδα 1 και παράλληλα, το reference bit της νέας σελίδας θα αλλάξει στην κατάσταση 1, παρά το ότι είναι η πρώτη φορά που αυτή εμφανίζεται.

Ο κώδικας για τον αλγόριθμο Clock:

```
void clock() {
	unsigned long long locat, found, i, j;
  initialize();
	for(i=0; i<nf; i++) { /* Initialize all array elements to 0. */
		frames[i]=0;
		usedbit[i]=0;
	}
	locat=0;
	for(i=0; i<records; i++) {
		found=0; /* Reset. */
		for(j=0; j<nf; j++) { /* Check if page is in memory. */
			if(frames[j]==data[i].pageNo) {
				found=1;
				usedbit[j]=1;
			}
		}
		if(found==0) {
			do { /* If bit is 0 or NULL, load in page. */
				if(usedbit[locat]==0) {
					frames[locat]=data[i].pageNo;
					usedbit[locat]=1;
					found=1;
					pgfaultcnt++;
				}
				else { /* Reset used bit. */
					usedbit[locat]=0;
				}
				locat++; /* Move pointer. */
```

```
                        if(locat==nf) { locat=0; } /* Reset. */
                    } while (found!=1);
                }
        }
  dispPgFaultCnt();
}
```

27. Andrew S. Tanenbaum, «Σύγχρονα Λειτουργικά Συστήματα», 3η αμερικανική έκδοση, επιστημονική επιμέλεια ελληνικής έκδοσης Δημήτρης Γκιζόπουλος, Πανεπιστήμιο Αθηνών, εκδόσεις «Κλειδάριθμος», σελ. 262.
28. Kartik Moudgil, Anushka Ringshia, Harshal Bharatkumar Parekh,Ria Maheshwari, Sheetal Chaudhari, «Systematic Analysis and Simulation of Classical Page Replacement Algorithms: A Proposed Novel WSClock-Derivative», 2017 2nd International Conference for Convergence in Technology (I2CT), p. 889.
29. Hitha Paulson, Dr. Rajesh Ramachandran, «Page Replacement Algorithms-Challenges and Trends», International Journal of Computer & Mathematical Sciences, Volume 6, Issue 9, September 2017, p. 114.

5.4 Το μενού επιλογών του προγράμματος PRA

Ο κώδικας για το μενού επιλογών του PRA. Πρόκειται για το τελευταίο τμήμα του συνολικού κώδικα, εντός της κύριας συνάρτησης (`int main()`):

```
int main() {
  int choice;
  //Using Switch statement in order to create an options menu.
  while (true) {
    printf("\n\nPage Replacement Algorithms\n");
    printf("----------------------------");
    printf("\n1.Enter data\n2.FIFO\n3.LRU\n4.SECOND
CHANCE\n5.CLOCK\n6.Exit\nEnter your "
           "choice (numbers 1-6): ");
    scanf("%d", &choice);
    switch (choice) {
    case 1:
      getData(); //File name and page frame number input.
      break;
    case 2:
      fifo(); //Press '2' to run FIFO.
      break;
    case 3:
      lru(); //Press '3' to run LRU.
      break;
    case 4:
      secondchance(); //Press '4' to run SECOND CHANCE.
      break;
    case 5:
      clock(); //Press '5' to run CLOCK.
      break;
    case 6:
      printf("\nYou have exited successfully\n"); //Press '6' to exit menu.
      return 0;
      break;
    default:
      printf("\nInvalid choice, please try again"); //Error message if
//hitting non existing number key.
      break;
    }
  }
  return 0;
```

Σε αυτό το σημείο θα πρέπει να γίνει μια επισήμανση. Τόσο τα τμήματα κώδικα των συναρτήσεων `void fifo()`[30], `void secondchance()`[31], `unsigned long long findLRU(unsigned long long time[], unsigned long long records)`[32] και `void lru()`[33] του αλγορίθμου LRU, και `void clock()`[34], οι οποίες αφορούν στους αντίστοιχους αλγορίθμους αντικατάστασης σελίδας, όσο και αυτό του μενού επιλογών της συνάρτησης `int main()`[35], έχουν ληφθεί από ιστοσελίδες στο Διαδίκτυο, έπειτα από ενδελεχή έρευνα. Έπρεπε όμως να γίνουν εκτεταμένες αλλαγές και επεξεργασία τους, προκειμένου να «δέσουν» αρμονικά με το συνολικό, πρωτότυπο πρόγραμμα του PRA.

30,31. https://www.programming9.com/programs/c-programs/285-page-replacement-programs-in-c.
32. https://educativesite.com/least-recently-used-lru-page-replacement-algorithm-in-c-and-c-program-code/.
33. https://raw.githubusercontent.com/zahan97/OS-Codes/master/LRU.c.
34. https://github.com/SydneyRaeBlackburn/PageReplacement/blob/master/page.c.
35. https://www.programming9.com/programs/c-programs/285-page-replacement-programs-in-c.

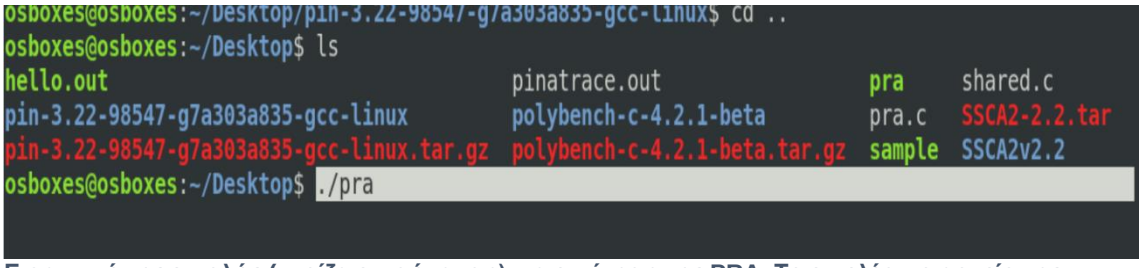
6 Εκτέλεση της εφαρμογής PRA και αποτελέσματα

6.1 Εισαγωγή

Το παρόν τμήμα της μεταπτυχιακής διατριβής αφορά στην εκτέλεση της εφαρμογής PRA, καθώς και στην παρουσίαση των αποτελεσμάτων που προκύπτουν από την εκτέλεση των αλγορίθμων αντικατάστασης σελίδας που περιλαμβάνονται σε αυτήν.

Η εκτέλεση της PRA λαμβάνει χώρα κάθε φορά που ο χρήστης εισάγει ως είσοδο το αρχείο εξόδου (pinatrace.out) του εργαλείου «pinatrace.cpp», που περιλαμβάνεται στο PIN της Intel. Ο χρήστης επιλέγει επίσης, τον αριθμό πλαισίων σελίδας που χρειάζονται για την εκτέλεση. Όπως έχει αναφερθεί ήδη στην ενότητα «Πραγματική εκτέλεση εφαρμογών μέσω του PIN»[36], το αρχείο εξόδου παράγεται για έτοιμα μετροπρογράμματα (benchmarks) που αφορούν σε regular (Polybench/C) και irregular memory access patterns (HPC Graph Analysis).

Προκειμένου να ξεκινήσει η εκτέλεση της PRA, θα πρέπει προφανώς να εισαχθεί στο τερματικό Linux η αντίστοιχη εντολή με το PRA εκτελέσιμο αρχείο, όπως φαίνεται στην ακόλουθη φωτογραφία:



Εισαγωγή της εντολής (γκρίζα επισήμανση) για εκκίνηση της PRA. Το εκτελέσιμο αρχείο της εφαρμογής βρίσκεται στο φάκελο "Desktop" των Linux, στην προκειμένη περίπτωση.

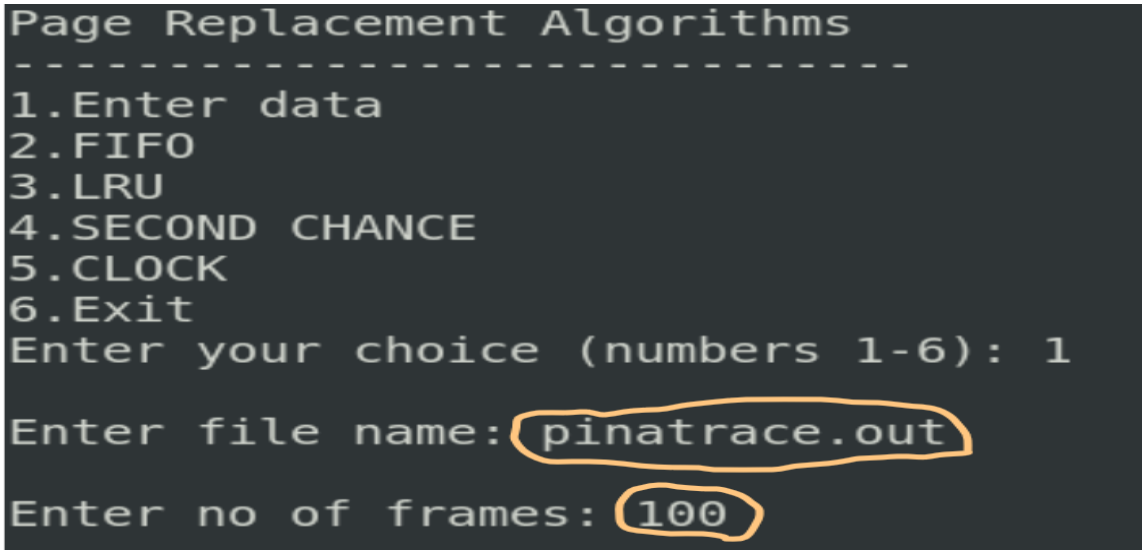
36. Ενότητα 4: «Πραγματική εκτέλεση εφαρμογών μέσω του PIN», σελ. 21.

6.2 Εκτέλεση της PRA για regular memory access patterns

Στην υποενότητα «Regular memory access patterns»[37], της ενότητας «Πραγματική εκτέλεση εφαρμογών μέσω του PIN», περιγράφεται αναλυτικά η διαδικασία δημιουργίας του εκτελέσιμου αρχείου για τα Polybench/C benchmarks, καθώς και η παραγωγή του αρχείου εξόδου τους, με τη χρήση του εργαλείου «pinatrace.cpp», του Intel PIN.

Για τις ανάγκες τις παρούσης εργασίας, παρήχθησαν αρχεία εξόδου από τρία Polybench/C benchmarks. Αυτά είναι κατά σειρά τα: «atax.c» (Matrix Transpose and Vector Multiplication), «bicg.c» (BiCG Sub Kernel of BiCGStab Linear Solver), «mvt.c» (Matrix Vector Product and Transpose) και «trisolv.c» (Triangular solver)[38].

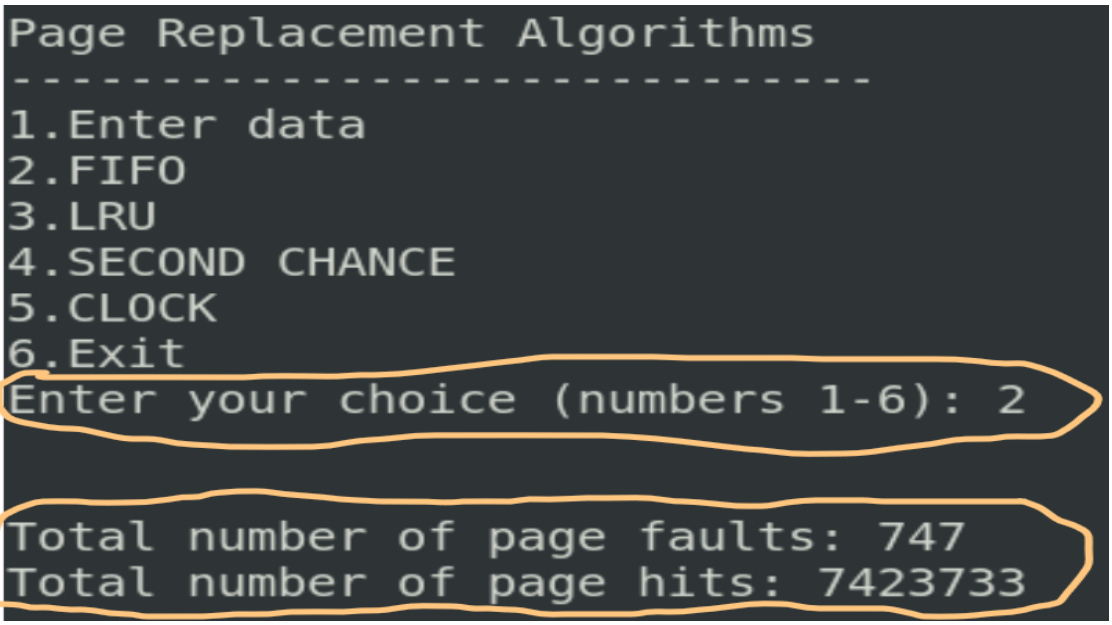
Το αρχείο εξόδου που δημιουργείται για το κάθε benchmark μπαίνει ως είσοδος στο πρόγραμμα PRA. Η εκτέλεση, σε όλες τις περιπτώσεις, γίνεται για πενήντα (50), εκατό (100), εκατόν πενήντα (150), διακόσια (200), διακόσια πενήντα (250), τριακόσια (300), τριακόσια πενήντα (350) και τετρακόσια (400) πλαίσια σελίδας.



Εκτέλεση της εφαρμογής PRA για το "atax.c" benchmark. Με κυκλική επισήμανση, το αρχείο εξόδου για το εκτελέσιμο "atax_base", καθώς και ο αριθμός πλαισίων σελίδας (100), στη συγκεκριμένη περίπτωση.

37. Υποενότητα 4.2: «Regular memory access patterns», σελ. 21.
38. Για τη συνοπτική περιγραφή των benchmarks: https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/, στην ενότητα «Available benchmarks (PolyBench/C version 3.2)».

Στο παράδειγμα της παραπάνω φωτογραφίας, η PRA εκτελείται για το αρχείο εξόδου, με είσοδο τα εκατό (100) πλαίσια σελίδας. Αυτό γίνεται για τον κάθε αλγόριθμο αντικατάστασης σελίδας, ξεχωριστά. Αφού εισαχθούν τα απαραίτητα δεδομένα (αρχείο εξόδου, αριθμοί πλαισίων), ο χρήστης επιλέγει, πατώντας τον αντίστοιχο αριθμό του μενού, για ποιόν αλγόριθμο θα γίνει η εκτέλεση. Με αυτόν τον τρόπο, εμφανίζεται ο αριθμός σφαλμάτων σελίδας (page faults) και ο αντίστοιχος των επιτυχιών (page hits) για κάθε έναν από τους αλγορίθμους της εφαρμογής (FIFO, LRU, Second Chance, Clock), για εκατό (100) πλαίσια σελίδας (page frames).



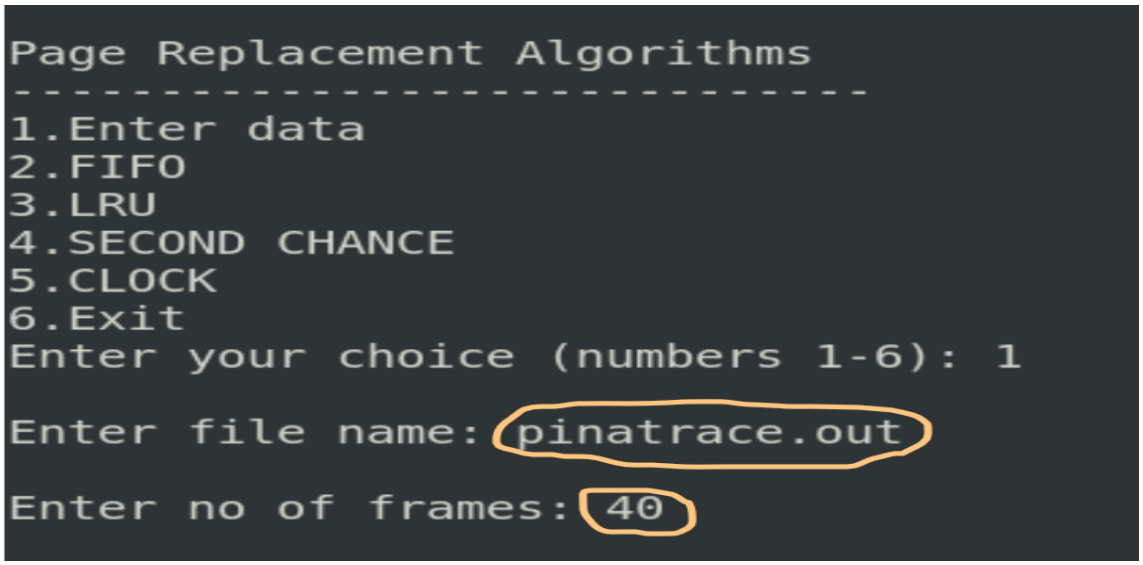
Εκτέλεση του αλγορίθμου αντικατάστασης σελίδας FIFO, της εφαρμογής PRA, για το αρχείο εξόδου του "atax_base". Ο αριθμός σφαλμάτων σελίδας (page faults) και επιτυχιών (page hits) αντιστοιχεί στην εισαγωγή, από τον χρήστη, εκατό (100) πλαισίων σελίδας (page frames).

Η ίδια διαδικασία επαναλαμβάνεται και για τους υπόλοιπους αριθμούς πλαισίων σελίδας, οι οποίοι έχουν αναφερθεί παραπάνω. Για την απεικόνιση των αποτελεσμάτων και των τεσσάρων (4) αλγορίθμων αντικατάστασης σελίδας, τόσο για regular, όσο και για irregular memory access patterns, δημιουργήθηκε λεπτομερές αρχείο, σε μορφή excel. Το αρχείο αποτελείται από πέντε (5) υπολογιστικά φύλλα (sheets). Τα τέσσερα αφορούν σε regular memory access patterns του «Polybench/C» (atax_base, bicg_base, mvt_base, trisolv_base), ενώ το ένα σχετίζεται με irregular memory access patterns του «HPC Graph Analysis» (συγκεκριμένα, το εκτελέσιμο του «SSCA#2 v2.2»).

6.3 Εκτέλεση της PRA για irregular memory access patterns

Στην υποενότητα «Irregular memory access patterns»[39], της ενότητας «Πραγματική εκτέλεση εφαρμογών μέσω του PIN», περιγράφεται αναλυτικά η διαδικασία δημιουργίας του εκτελέσιμου αρχείου για το HPC Graph Analysis benchmark, καθώς και η παραγωγή του αρχείου εξόδου του, με τη χρήση του εργαλείου «pinatrace.cpp», του Intel PIN.

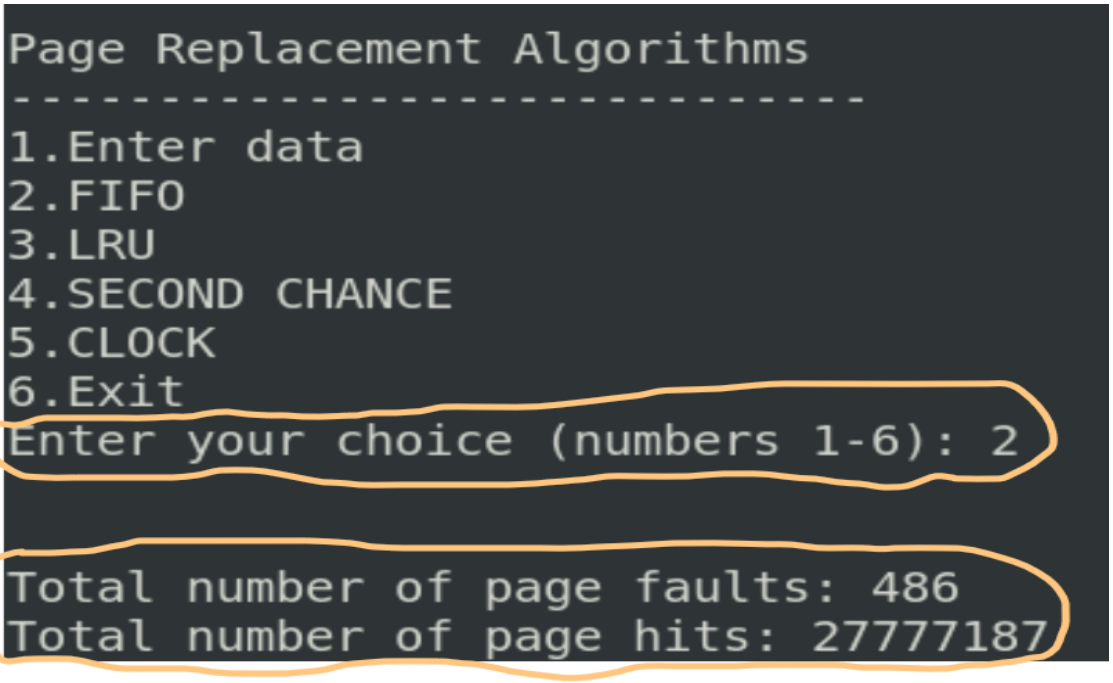
Για τη συγκεκριμένη εκτέλεση, παρήχθη αρχείο εξόδου για ένα μόνο benchmark, το «SSCA2.c» (Scalable Synthetic Compact Applications graph analysis benchmark). Το αρχείο εξόδου που δημιουργείται μπαίνει ως είσοδος στην εφαρμογή PRA, ενώ η εκτέλεση λαμβάνει χώρα για σαράντα (40), εξήντα (60), ογδόντα (80), εκατό (100) και εκατόν είκοσι (120) πλαίσια σελίδων, αντίστοιχα.



Εκτέλεση της εφαρμογής PRA για το "SSCA2.c" benchmark. Με κυκλική επισήμανση, το αρχείο εξόδου για το εκτελέσιμο "SSCA2 ", καθώς και ο αριθμός πλαισίων σελίδας (40), στη συγκεκριμένη περίπτωση.

Στο παράδειγμα της άνω φωτογραφίας, η εφαμοργή PRA εκτελείται για το αρχείο εξόδου, με είσοδο τα σαράντα (40) πλαίσια σελίδας. Αυτό γίνεται για τον κάθε αλγόριθμο αντικατάστασης σελίδας, ξεχωριστά. Αφού εισαχθούν τα απαραίτητα δεδομένα (αρχείο εξόδου, αριθμοί πλαισίων), ο χρήστης επιλέγει, πατώντας τον αντίστοιχο αριθμό του μενού, για ποιόν αλγόριθμο θα γίνει η εκτέλεση. Με αυτόν τον τρόπο, εμφανίζεται ο αριθμός σφαλμάτων σελίδας (page faults) και ο αντίστοιχος των επιτυχιών (page hits) για κάθε έναν από τους αλγορίθμους της εφαρμογής (FIFO, LRU, Second Chance, Clock), για σαράντα (40) πλαίσια σελίδας (page frames).

39. Υποενότητα 4.3: «Irregular memory access patterns», σελ. 24.



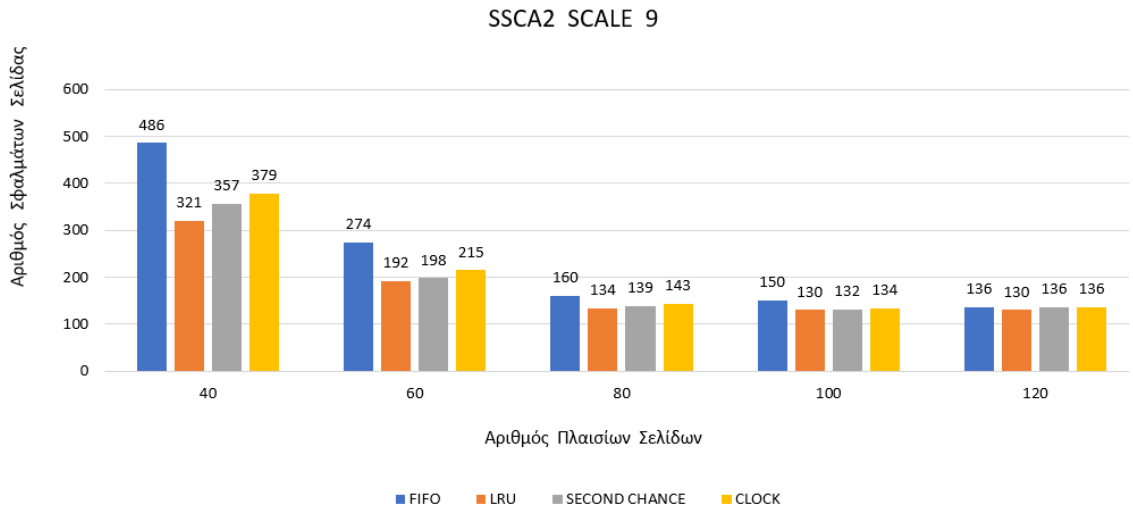
Εκτέλεση του αλγορίθμου αντικατάστασης σελίδας FIFO, της εφαρμογής PRA, για το αρχείο εξόδου του "SSCA2". Ο αριθμός σφαλμάτων σελίδας (page faults) και επιτυχιών (page hits) αντιστοιχεί στην εισαγωγή, από τον χρήστη, σαράντα (40) πλαισίων σελίδας (page frames).

Η ίδια διαδικασία επαναλαμβάνεται και για τους υπόλοιπους αριθμούς πλαισίων σελίδας, οι οποίοι έχουν αναφερθεί στην αρχή της υποενότητας.

6.4 Αποτελέσματα της εκτέλεσης της εφαρμογής PRA

Για την απεικόνιση των αποτελεσμάτων και των τεσσάρων (4) αλγορίθμων αντικατάστασης σελίδας, τόσο για regular, όσο και για irregular memory access patterns, δημιουργήθηκε λεπτομερές αρχείο, σε μορφή excel. Το αρχείο αποτελείται από πέντε (5) υπολογιστικά φύλλα (sheets). Τα τέσσερα αφορούν σε regular memory access patterns του «Polybench/C» (atax_base, bicg_base, mvt_base, trisolv_base), ενώ το ένα σχετίζεται με irregular memory access patterns του «HPC Graph Analysis», συγκεκριμένα το εκτελέσιμο του «SSCA#2 v2.2» (SSCA2).

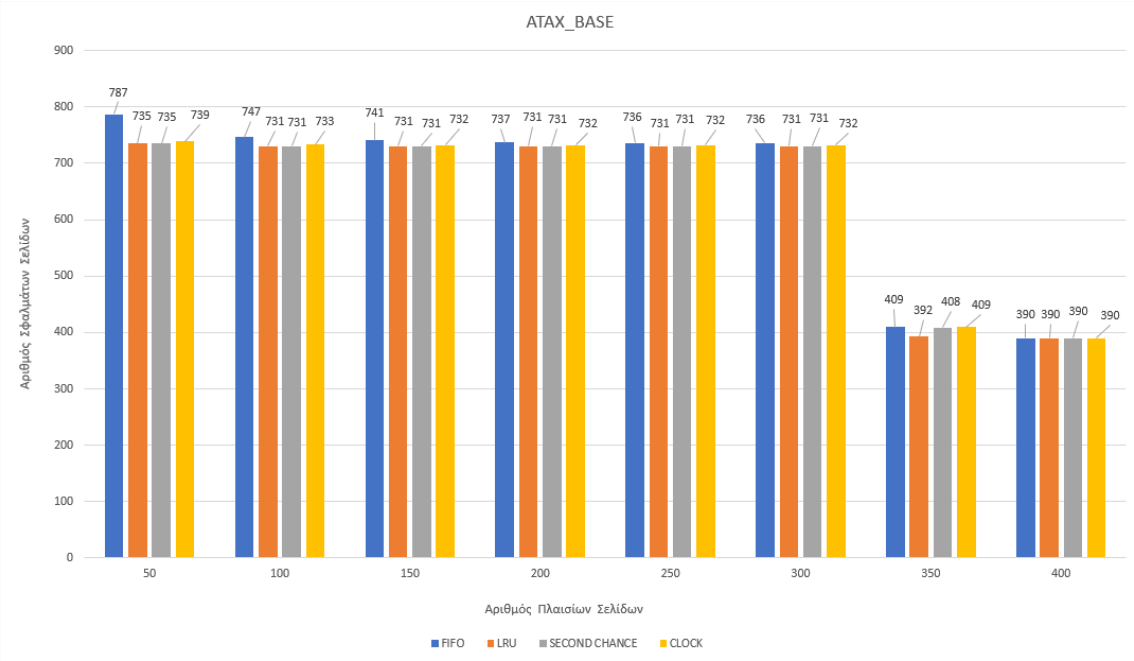
Στο κάθε φύλλο του αρχείου, υπάρχει μια γραφική παράσταση στηλών. Στον οριζόντιο άξονά της, εμφανίζονται οι αριθμοί πλαισίων που εισάγει ο χρήστης στην εφαρμογή PRA. Στον κάθετο άξονα, παρουσίαζονται οι αριθμοί σφαλμάτων σελίδας (page faults). Κάθε αλγόριθμος αντικατάστασης σελίδας εμφανίζεται με τη μορφή στήλης, διαφορετικού χρώματος. Πάνω από την κάθε στήλη, υπάρχει αριθμητική επισήμανση, η οποία αντιστοιχεί στον ακριβή αριθμό σφαλμάτων σελίδας του κάθε αλγορίθμου, για δεδομένο αριθμό πλαισίων σελίδας. Ένα παράδειγμα γραφικής παράστασης παρουσιάζεται στην παρακάτω φωτογραφία:



Παράδειγμα γραφικής παράστασης στηλών με αποτελέσματα της εκτέλεσης της εφαρμογής PRA.

Σε ό,τι αφορά την εκτέλεση της PRA για τα Polybench benchmarks, έχουν δημιουργηθεί τέσσερις γραφικές παραστάσεις. Αυτές αντιστοιχούν σε κάθε ένα από τα τέσσερα εκτελέσιμα των Polybench benchmarks, της παρούσης μεταπτυχιακής διατριβής: «atax_base», «bicg_base», «mvt_base» και «trisolv_base».

Η πρώτη από τις γραφικές παραστάσεις αναφέρεται στα αποτελέσματα του εκτελέσιμου «atax_base»:



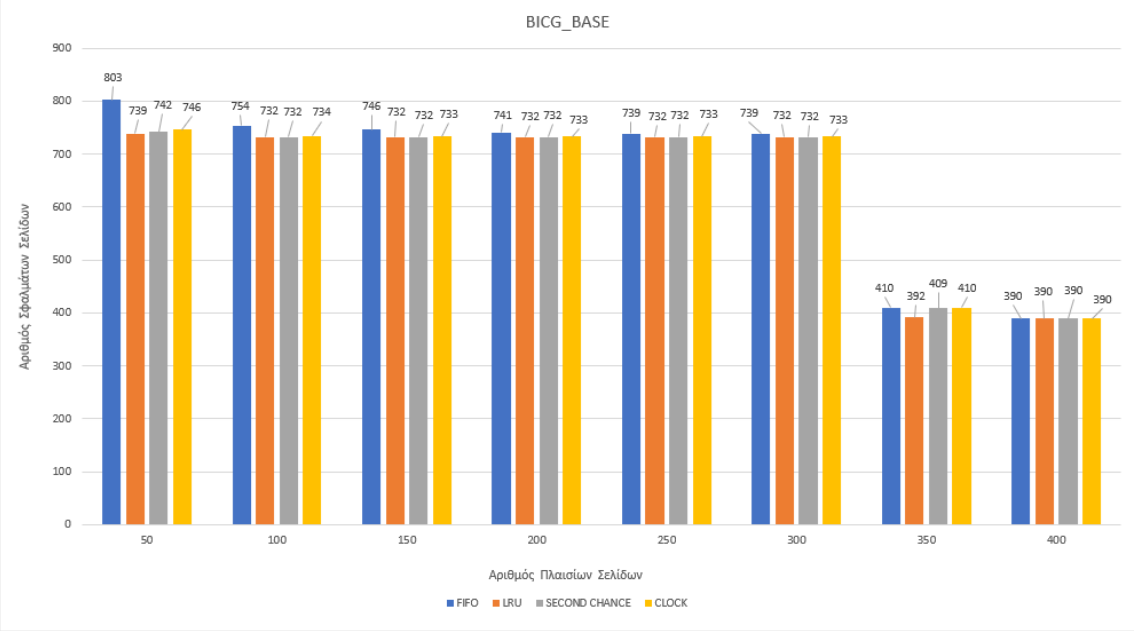
Η γραφική παράσταση με τα αποτελέσματα εκτέλεσης της PRA για το εκτελέσιμο "atax_base".

Όπως φαίνεται και από την παράσταση, στα πενήντα (50) πλαίσια σελίδων, ο αλγόριθμος FIFO παρουσιάζει έναν εμφανώς μεγαλύτερο αριθμό σφαλμάτων, επτακόσια ογδόντα επτά (787) τον αριθμό, έναντι των υπολοίπων. Ο LRU παρουσιάζει επτακόσια τριάντα πέντε (735) σφάλματα, όσα και ο Second Chance, ενώ ο αλγόριθμος Clock εμφανίζει επτακόσια τριάντα εννιά (739).

Από τα εκατό (100) μεχρι και τα τριακόσια (300) πλαίσια σελίδων, φαίνεται να υπάρχει μια ισορροπία στον αριθμό σφαλμάτων, μεταξύ των αλγορίθμων. Σε όλες τις περιπτώσεις όμως, ο FIFO παρουσιάζει, έστω και οριακά, μεγαλύτερο αριθμό, ενώ οι LRU και Second Chance ισορροπούν, έχοντας και τους μικρότερους δείκτες, με επτακόσια τριάντα ένα (731) σφάλματα.

Αξιοσημείωτη είναι η αισθητή πτώση, στον αριθμό σφαλμάτων, που παρουσιάζουν και οι τέσσερις αλγόριθμοι στα τριακόσια πενήντα (350) πλαίσια σελίδας. Υπάρχει μια σχετική ισορροπία στον αριθμό σφαλμάτων μεταξύ του FIFO με τετρακόσια εννιά (409), του Second Chance με τετρακόσια οκτώ (408) και του Clock με τετρακόσια εννιά (409), όσα και ο FIFO. Ο LRU εμφανίζει το μικρότερο αριθμό σφαλμάτων σελίδας, τριακόσια ενενήντα δύο (392).

Η δεύτερη γραφική παράσταση σχετίζεται με τα αποτελέσματα για το εκτελέσιμο «bicg_base»:



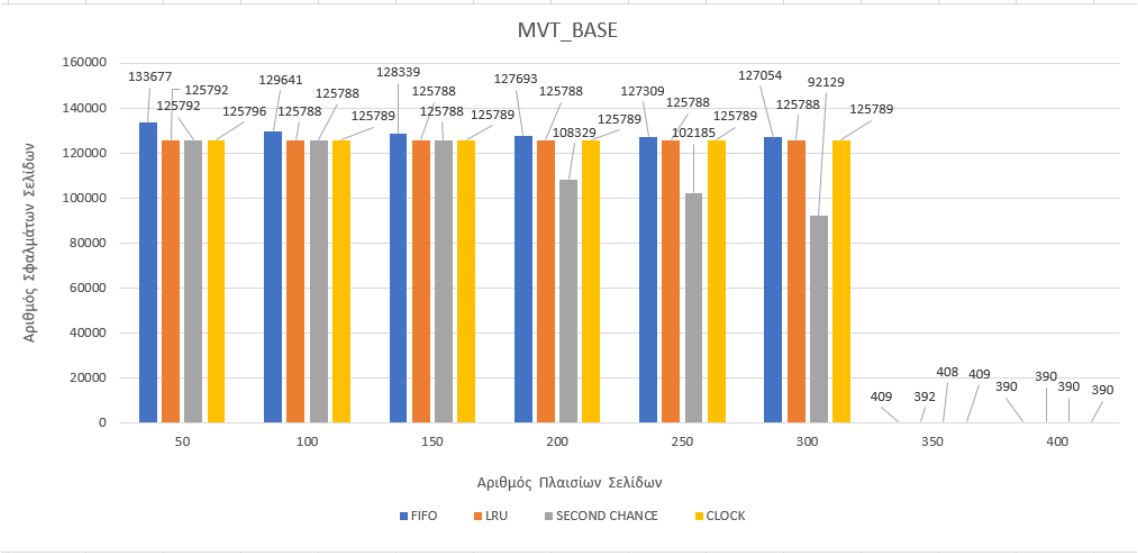
Η γραφική παράσταση με τα αποτελέσματα εκτέλεσης της PRA για το εκτελέσιμο "bicg_base".

Σύμφωνα και με αυτήν την παράσταση, στα πενήντα (50) πλαίσια σελίδων, ο αλγόριθμος FIFO παρουσιάζει ξάνα έναν αισθητά μεγαλύτερο αριθμό σφαλμάτων, οκτακόσια τρία (803) τον αριθμό, έναντι των υπολοίπων. Ο LRU εμφανίζει επτακόσια τριάντα εννιά (739) σφάλματα, ο Second Chance επτακόσια σαράντα δύο (742), ενώ ο αλγόριθμος Clock εμφανίζει επτακόσια σαράντα έξι (746).

Από τα εκατό (100) μέχρι και τα τριακόσια (300) πλαίσια σελίδων, φαίνεται να υπάρχει ξανά μια ισορροπία στον αριθμό σφαλμάτων, μεταξύ των αλγορίθμων. Σε όλες τις περιπτώσεις όμως, ο FIFO παρουσιάζει, έστω και οριακά πάλι, μεγαλύτερο αριθμό, ενώ οι LRU και Second Chance ισορροπούν, έχοντας και τους μικρότερους δείκτες με επτακόσια τριάντα δύο (732) σφάλματα.

Αξιοσημείωτη είναι και εδώ η αισθητή πτώση, στον αριθμό σφαλμάτων, που παρουσιάζουν και οι τέσσερις αλγόριθμοι στα τριακόσια πενήντα (350) πλαίσια σελίδας. Υπάρχει μια σχετική ισορροπία στον αριθμό σφαλμάτων μεταξύ του FIFO στα τετρακόσια δέκα (410), του Second Chance στα τετρακόσια εννιά (409) και του Clock στα τετρακόσια δέκα (410), όσα και ο FIFO. Ο LRU εμφανίζει και πάλι το μικρότερο αριθμό σφαλμάτων σελίδας, τριακόσια ενενήντα δύο (392).

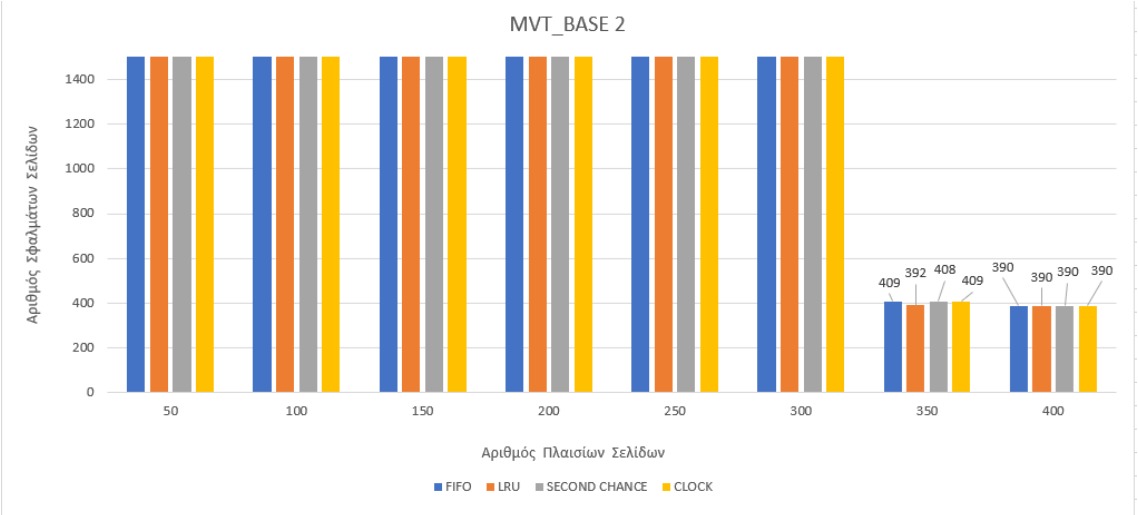
Η τρίτη, κατά σειρά, γραφική παράσταση αφορά στα αποτελέσματα για το εκτελέσιμο «mvt_base»:



Η πρώτη γραφική παράσταση με τα αποτελέσματα εκτέλεσης της PRA για το εκτελέσιμο "mvt_base".

Στην περίπτωση των αποτελεσμάτων για το «mvt_base», υπάρχει μια ιδιομορφία. ‘Οπως φαίνεται από την παραπάνω γραφική παράσταση, όλοι οι αλγόριθμοι αντικατάστασης σελίδας παρουσιάζουν μεγάλους αριθμούς σφαλμάτων σελίδας, από τα πενήντα (50) μέχρι και τα τριακόσια (300) πλαίσια σελίδας. Στα τριακόσια πενήντα (350) και τετρακόσια (400) πλαίσια σελίδας όμως, υπάρχει μια πολύ μεγάλη πτώση στα σφάλματα. Όλοι οι αλγόριθμοι παρουσιάζουν τριψήφια σφάλματα, όπως ακριβώς στις περιπτώσεις των αποτελεσμάτων των προηγούμενων εκτελέσιμων.

Προκειμένου να είναι ευκρινή όλα τα αποτελέσματα, αποφασίστηκε η παράσταση να παρουσιαστεί σε δύο τμήματα. Στο πρώτο τμήμα, εμφανίζονται τα αποτελέσματα ανάμεσα στα πενήντα (50) και τα τριακόσια (300) πλαίσια σελίδας. Στο δεύτερο τμήμα που ακολουθεί, παρουσιάζονται τα αποτελέσματα για τα τριακόσια πενήντα (350) και τα τετρακόσια (400) πλαίσια σελίδας, αντίστοιχα.

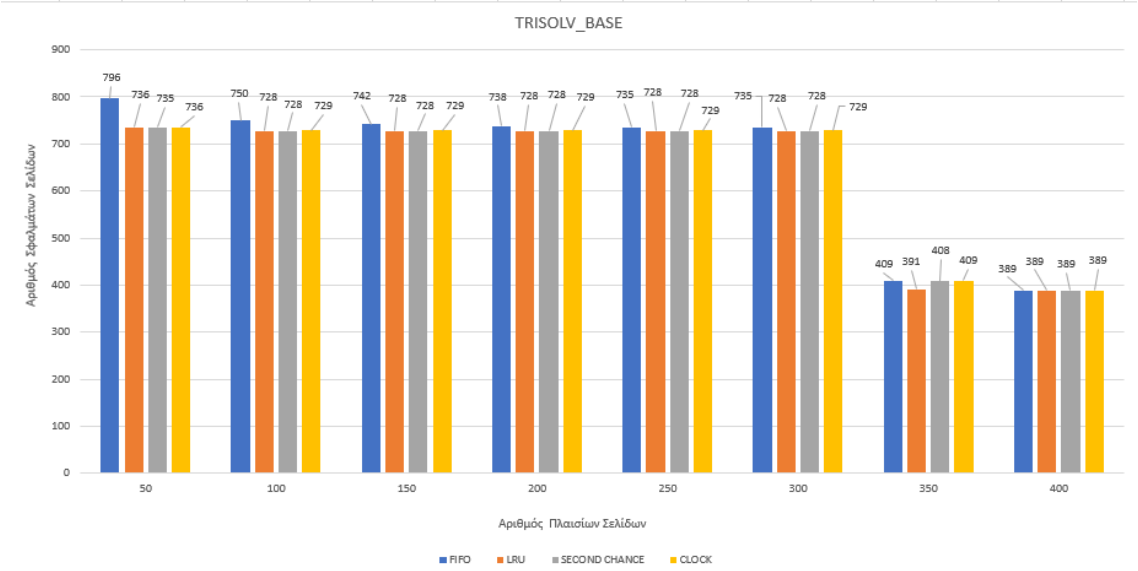


Η δεύτερη γραφική παράσταση με τα αποτελέσματα εκτέλεσης της PRA για το εκτελέσιμο "mvt_base". Εδώ, φαίνονται καθαρά τα αποτελέσματα στα τριακόσια πενήντα (350) και τα τετρακόσια (400) πλαίσια σελίδας, αντίστοιχα.

Τα στοιχεία για το «mvt_base» παρουσιάζουν ενδιαφέρον. Όπως φαίνεται και από τις γραφικές παραστάσεις, ο FIFO παραμένει ο αλγόριθμος με τα περισσότερα, έστω και οριακά, σφάλματα σελίδας, ανάμεσα στα πενήντα (50) και τα τριακόσια (300) πλαίσια σελίδας. Το νέο όμως εύρημα αποτελεί το γεγονός ότι, στο συγκεκριμένο εύρος και συγκεκριμένα, μεταξύ των διακοσίων (200) και τριακοσίων (300) πλαισίων, ξεχωρίζει αισθητά ο αλγόριθμος Second Chance, από άποψη αποδοτικότητας.

Φαίνεται ξεκάθαρα πως παρουσιάζει με διαφορά, το μικρότερο αριθμό σφαλμάτων σελίδας και αυτό ενώ, ανάμεσα στα πενήντα (50) και τα εκατόν πενήντα (150) πλαίσια, βρίσκεται σε ισορροπία με τον LRU. Στα τριακόσια πενήντα (350) πλαίσια σελίδας, τα αποτελέσματα είναι σχεδόν όμοια με αυτά των προηγούμεων εκτελέσιμων αρχείων. Ο FIFO εμφανίζει τον ίδιο αριθμό σφαλμάτων με τον Clock, τετρακόσια εννέα (409) τον αριθμό, ενώ ο LRU είναι ο πιο αποδοτικός αλγόριθμος, με τριακόσια ενενήντα δύο (392) σφάλματα σελίδας, όσα και στα δύο προηγούμενα εκτελέσιμα.

Ακολουθεί η τέταρτη γραφική παράσταση των αποτελεσμάτων για το εκτελέσιμο «trisolv_base», το τελευταίο που σχετίζεται με τα Polybench benchmarks:



Η γραφική παράσταση με τα αποτελέσματα εκτέλεσης της PRA για το εκτελέσιμο "trisolv_base".

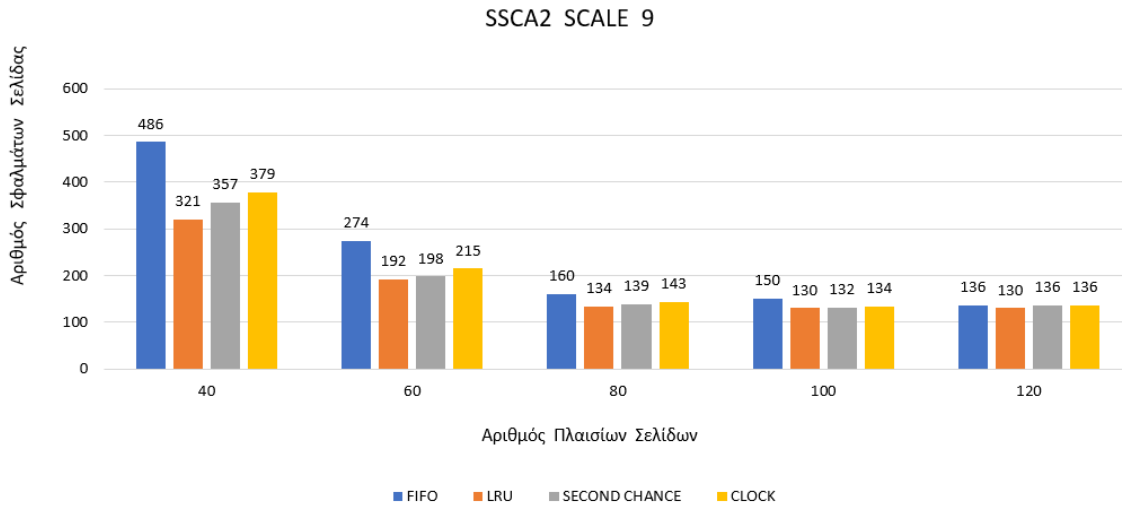
Και σε αυτήν την παράσταση, στα πενήντα (50) πλαίσια σελίδων, ο αλγόριθμος FIFO παρουσιάζει έναν εμφανώς μεγαλύτερο αριθμό σφαλμάτων, επτακόσια ενενήντα έξι (796) τον αριθμό, έναντι των υπολοίπων. Ο LRU εμφανίζει επτακόσια τριάντα έξι (736), όσα και ο αλγόριθμος Clock, με τον Second Chance να έχει ένα σφάλμα λιγότερο, στα επτακόσια τριάντα πέντε (735).

Από τα εκατό (100) μέχρι και τα τριακόσια (300) πλαίσια σελίδων, εμφανίζεται μια σχετική ισορροπία στον αριθμό σφαλμάτων, μεταξύ των αλγορίθμων. Σε όλες τις περιπτώσεις όμως, ο FIFO παρουσιάζει, έστω και οριακά, μεγαλύτερο αριθμό σφαλμάτων, ενώ οι LRU και Second Chance ισορροπούν, έχοντας και τους μικρότερους δείκτες, με επτακόσια είκοσι οκτώ (728) σφάλματα.

Εμφανής είναι και εδώ η αισθητή πτώση, στον αριθμό σφαλμάτων, που παρουσιάζουν και οι τέσσερις αλγόριθμοι στα τριακόσια πενήντα (350) πλαίσια σελίδας. Υπάρχει ισορροπία στον αριθμό σφαλμάτων μεταξύ του FIFO και του Clock με τετρακόσια εννιά (409) σφάλματα σελίδας, ενώ ο LRU είναι ο πιο αποδοτικός, με τριακόσια ενενήντα ένα (391).

Η παρουσίαση των γραφικών παραστάσεων, με τα αποτελέσματα των εκτελέσιμων για τα benchmarks του «Polybench/C», ολοκληρώνεται με μία σημαντική επισήμανση. Σε όλες ανεξαιρέτως τις παραπάνω παραστάσεις, στα τετρακόσια (400) πλαίσια σελίδας, εμφανίζεται απόλυτη ισορροπία μεταξύ των αλγορίθμων αντικατάστασης σελίδας, αναφορικά με τον αριθμό σφαλμάτων. Ο λόγος είναι το ότι, στη συγκεκριμένη περίπτωση, όλες οι εντολές και, κατά συνέπεια, ολόκληρο το αρχείου εξόδου (pinatrace.out), «χωράνε» στην κύρια, φυσική μνήμη.

Αναφορικά με την εκτέλεση της PRA για τα HPC Graph Analysis benchmarks, παρουσιάζεται μία γραφική παράσταση, η οποία αντιστοιχεί στο εκτελέσιμο «SSCA2». Τα αποτελέσματα ισχύουν για $2^9 = 512$ κόμβους:



Η γραφική παράσταση με τα αποτελέσματα εκτέλεσης της PRA για το εκτελέσιμο "SSCA2".

Όπως έχει αναφερθεί ήδη στην υποενότητα «Εκτέλεση της PRA για irregular memory access patterns», τα αποτελέσματα προέκυψαν για σαράντα (40), εξήντα (60), ογδόντα (80), εκατό (100) και εκατόν είκοσι (120) πλαίσια σελίδων, αντίστοιχα.

Από τη γραφική παράσταση, διαπιστώνει κανείς ότι ο αλγόριθμος FIFO είναι και σε αυτήν την περίπτωση, ο λιγότερο αποδοτικός. Συγκεκριμένα, για σαράντα (40) και εξήντα (60) πλαίσια σελίδων εμφανίζει αισθητά μεγαλύτερους αριθμούς σφαλμάτων σελίδας από τους υπόλοιπους, τετρακόσια ογδόντα έξι (486) και διακόσια εβδομήντα τέσσερα (274) σφάλματα, αντίστοιχα. Στα ογδόντα (80) και εκατό (100) πλαίσια σελίδας, παρουσιάζει μια οριακή αύξηση, με εκατόν εξήντα (160) και εκατόν πενήντα (150) σφάλματα στην κάθε περίπτωση, ενώ στα εκατόν είκοσι (120) πλαίσια, ισορροπεί με τους Second Chance και Clock, με εκατόν τριάντα έξι (136) σφάλματα σελίδας. Ο αποδοτικότερος αλγόριθμος είναι, όπως φαίνεται και στην παράσταση, ο LRU (εκατόν τριάντα (130) σφάλματα).

7 Συμπεράσματα

Τόσο κατά τη διάρκεια, όσο και μετά το τέλος της διαδικασίας υλοποίησης της παρούσης μεταπτυχιακής διατριβής, προέκυψαν ορισμένα ενδιαφέροντα ευρήματα. Αρχικά, αναδεικνύονται ορισμένα ενδιαφέροντα στοιχεία σχετικά με τα αρχεία εξόδου που δημιουργήθηκαν τόσο για τα Polybench benchmarks, όσο και για αυτό του «HPC Graph Analysis».

Συγκεκριμένα, παρουσιάζεται μια σχετική ομοιομορφία αναφορικά με το μέγεθος των .out αρχείων για τα εκτελέσιμα των τριών πρώτων benchmarks του «Polybench/C». Το αρχείο εξόδου για το «atax_base» έχει μέγεθος 193 MB, το αντίστοιχο του «bicg_base» 169.7 MB, ενώ αυτό του «mvt_base» ανέρχεται στα 200.7 MB. Επίσης ενδιαφέρον είναι το γεγονός ότι, το αρχείο εξόδου για το «trisolv_base» έχει μέγεθος μόλις 64.5 MB.

Εξίσου ενδιαφέρουσα είναι και η σύγκριση μεγεθών για τα αρχεία εξόδου συνολικά των Polybench benchmarks με αυτό του «HPC Graph Analysis». Το .out αρχείο του εκτελέσιμου «SSCA2» είναι με διαφορά ογκωδέστερο από τα τέσσερα παραπάνω. Με μέγεθος 722.2 MB, είναι μεγαλύτερο, όχι μόνο έναντι καθενός από τα αρχεία εξόδου των Polybench benchmarks, αλλά και από το άθροισμα των συνολικών μεγεθών τους (627.9 MB συνολικό μέγεθος έναντι 722.2 MB του αρχείου εξόδου για το «SSCA2»).

Χρήσιμα συμπεράσματα εξάγονται και από τις πληροφορίες των διαγραμμάτων στο αρχείο Excel και οι οποίες αφορούν στη συμπεριφορά των αλγορίθμων αντικατάστασης σελίδας της εφαρμογής PRA. Αναφορικά με την εκτέλεση της PRA για το «SSCA2», βλέπουμε μια σταδιακή μείωση του αριθμού σφαλμάτων σελίδας για τον κάθε αλγόριθμο, όσο αυξάνεται ο αριθμός πλαισίων σελίδας. Επιπλέον, συγκρίνοντας τους αλγορίθμους αντικατάστασης σελίδας μεταξύ τους, για κάθε αριθμό πλαισίων σελίδας, διαπιστώνουμε ότι, ο λιγότερο αποδοτικός αλγόριθμος είναι ο FIFO. Ειδικά στα σαράντα (40) και τα εξήντα (60) πλαίσια σελίδας, υστερεί με διαφορά έναντι των υπολοίπων αλγορίθμων και μόνο στα εκατόν είκοσι (120) πλαίσια βρίσκεται στα ίδια επίπεδα με τους Second Chance και Clock, αντίστοιχα. Από την άλλη μεριά, ο LRU εμφανίζεται ως ο πιο αποδοτικός σε όλες τις περιπτώσεις.

Σε ό,τι αφορά την εκτέλεση της PRA για τα Polybench benchmarks, η εικόνα είναι πιο σύνθετη και τα αποτελέσματα πιο ενδιαφέροντα. Αρχικά, αξιοσημείωτη και στις τέσσερις περιπτώσεις είναι η απότομη και όχι σταδιακή πτώση των σφαλμάτων σελίδας, για όλους ανεξαιρέτως τους αλγορίθμους, στα τριακόσια πενήντα (350) και τα τετρακόσια (400) πλαίσια σελίδας. Ειδικά στην περίπτωση του «mvt_base», το διάγραμμα έπρεπε να «σπάσει» σε δύο μέρη, ώστε να είναι ορατά όλα τα αποτελέσματα. Όμως, το εν γένει μοτίβο συμπεριφοράς όλων των αλγορίθμων για όλους τους αριθμούς σελίδας και για τα τέσσερα εκτελέσιμα είναι σχεδόν το ίδιο.

Στο κομμάτι της αποδοτικότητας των αλγορίθμων για τα benchmarks του «Polybench/C», ο FIFO είναι και πάλι ο λιγότερο αποδοτικός από τους τέσσερις. Εξαίρεση αποτελεί η περίπτωση των τετρακοσίων (400) πλαισίων σελίδας όπου, για όλα τα εκτελέσιμα, εμφανίζεται ο ίδιος αριθμός σφαλμάτων και για τους τέσσερις αλγορίθμους. Αυτό συμβαίνει επειδή, όπως αναφέρθηκε στην προηγούμενη υποενότητα, τα αρχεία εξόδου «χωράνε» ολόκληρα στην κύρια μνήμη.

Σε ό,τι έχει να κάνει όμως με τον πιο αποδοτικό αλγόριθμο, δεν υφίσταται το καθαρό προβάδισμα που είχε ο LRU στην περίπτωση του «SSCA2». Αντίθετα, ο Second Chance τον ανταγωνίζεται σε απόδοση. Στην περίπτωση μάλιστα, του «mvt_base» και συγκεκριμένα στο διάστημα ανάμεσα στα διακόσια (200) και τα τριακόσια (300) πλαίσια σελίδων, ο τελευταίος υπερέχει με διαφορά των υπολοίπων σε αριθμό σφαλμάτων σελίδας.

Βιβλιογραφία

- Andrew S. Tanenbaum, «Σύγχρονα Λειτουργικά Συστήματα», 3η Αμερικανική Έκδοση, επιστημονική επιμέλεια ελληνικής έκδοσης Δημήτρης Γκιζόπουλος, Εκδόσεις Κλειδάριθμος.
- Hitha Paulson, Dr. Rajesh Ramachandran, «Page Replacement Algorithms-Challenges and Trends». International Journal of Computer & Mathematical Sciences, Volume 6, Issue 9, September 2017.
- Peter J. Denning, «Third Generation Computer Systems», ACM Computing Surveys, Vol. 3, No. 4, December 1971.
- Alfred V. Aho, Peter J. Denning, Jeffrey D. Ullman, «Principles of Optimal Page Replacement», Journal of the ACM, Volume 18, Issue 1, January 1971.
- Kartik Moudgil, Anushka Ringshia, Harshal Bharatkumar Parekh, Ria Maheshwari, Sheetal Chaudhari, «Systematic Analysis and Simulation of Classical Page Replacement Algorithms: A Proposed Novel WSClock-Derivative», IEEE, 2017 2nd International Conference for Convergence in Technology (I2CT).
- Kiran Muhammad Saleem, Maria Iqbal, Hadia Saadi, Farheen Qazi, Dur-e-Shawar Agha, «Second Chance Page Replacement Algorithm with Optimal (SCAO)», IEEE, 2019 International Conference on Information Science and Communication Technology (ICISCT).
- «Pin-A Dynamic Binary Instrumentation Tool»: www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html.
- «Polybench/C-the Polyhedral Benchmark suite»: https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/.
- «HPC Graph Analysis»: http://www.graphanalysis.org/benchmark/.
- https://www.programming9.com/programs/c-programs/285-page-replacement-programs-in-c.
- https://educativesite.com/least-recently-used-lru-page-replacement-algorithm-in-c-and-c-program-code/.
- https://raw.githubusercontent.com/zahan97/OS-Codes/master/LRU.c.
- https://github.com/SydneyRaeBlackburn/PageReplacement/blob/master/page.c.