Διπλωματική Εργασία

ΑΞΙΟΛΟΓΗΣΗ ΠΡΟΣΤΑΣΙΑΣ ΜΝΗΜΗΣ ΣΕ ΣΥΣΤΗΜΑΤΑ ΑΠΟΘΗΚΕΥΣΗΣ ΔΕΔΟΜΕΝΩΝ ΕΠΙΠΕΔΟΥ ΧΡΗΣΤΗ ΓΙΑ ΜΗ-ΠΤΗΤΙΚΗ ΜΝΗΜΗ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΖΗΝΩΝΟΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΑΙΟΣ 2023

Επίβλεψη: Καθηγητής Χάρης Βώλος

Διπλωματική που υποβλήθηκε σε μερική εκπλήρωση των απαιτήσεων για την απόκτηση πτυχίου Bachelor στο τμήμα πληροφορικής στο Πανεπιστήμιο Κύπρου.

ΕΥΧΑΡΙΣΤΙΕΣ

Σε αυτό το σημείο, θα ήθελα να εκφράσω την εγκάρδια ευγνωμοσύνη μου στον επιβλέποντα καθηγητή μου, Δρ. Χάρη Βώλο, για την αταλάντευτη υποστήριξη και καθοδήγησή του καθ' όλη τη διάρκεια της εκπόνησης αυτής της διατριβής. Από την επιλογή του θέματος μέχρι την ολοκλήρωση του γραπτού κειμένου, ο Δρ. Βώλος έχει προσφέρει ανεκτίμητη βοήθεια και οι πολυάριθμες συζητήσεις μας έχουν συμβάλει καθοριστικά στη κατανόηση του θέματος της πτυχιακής μου και της προσέγγισής που επέλεξα.

Επιπλέον, είμαι βαθιά ευγνώμων σε όλο το διδακτικό προσωπικό του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου. Τα μαθήματα που παρείχαν με εξόπλισαν με τις απαραίτητες δεξιότητες και γνώσεις για να ολοκληρώσω με επιτυχία τις σπουδές μου στον τομέα της Πληροφορικής. Είμαι βέβαιος ότι οι πόροι και οι εμπειρίες που έχω αποκτήσει κατά τη διάρκεια των σπουδών μου στο Πανεπιστήμιο θα μου δώσουν τη δυνατότητα να ξεπεράσω τυχόν προκλήσεις που μπορεί να αντιμετωπίσω στο επαγγελματικό μου ταξίδι, ενώ παράλληλα θα θέσουν τις βάσεις για μια ανταποδοτική καριέρα σε αυτόν τον τομέα.

Επιπλέον, θα ήθελα να εκφράσω τη βαθιά μου εκτίμηση στην οικογένειά μου, η οποία υπήρξε μια σταθερή πηγή υποστήριξης και ενθάρρυνσης από τη στιγμή που επέλεξα να ακολουθήσω αυτόν τον τομέα σπουδών. Η ακλόνητη πίστη τους στις ικανότητές μου υπήρξε κινητήρια δύναμη καθόλη τη διάρκεια του ακαδημαϊκού μου ταξιδιού.

Τέλος, πρέπει να αναγνωρίσω ότι δεν θα μπορούσα ποτέ να επιτύχω αυτό το επίπεδο επιτυχίας χωρίς την υποστήριξη και την ενθάρρυνση των συμφοιτητών και των φίλων μου, οι οποίοι μου παρείχαν κίνητρα και διορατική ανατροφοδότηση. Η συμβολή τους στην ακαδημαϊκή και την προσωπική μου ανάπτυξη δεν μπορεί να υπερεκτιμηθεί.

ΠΕΡΙΛΗΨΗ

Στην παρούσα διπλωματική εργασία, πρωταρχικός στόχος είναι η εις βάθος αξιολόγηση των μηχανισμών προστασίας και ασφάλειας σε συστήματα αποθήκευσης δεδομένων λειτουργίας χρήστη για μη πτητική μνήμη. Συγκεκριμένα, ο σκοπός είναι να προσδιοριστεί η πιο αποτελεσματική τεχνολογία για την παροχή προστασίας μνήμης που μπορεί να υιοθετηθεί από την Optane.

Η μόνιμη μνήμη σημαίνει ότι οι ανησυχίες σχετικά με την απώλεια δεδομένων λόγω δυσλειτουργιών του υπολογιστή μετριάζονται, καθώς οι αποθηκευμένες πληροφορίες παραμένουν σταθερά διαθέσιμες, ανεξάρτητα από την κατάσταση του υπολογιστή. Η πρόσβαση στις πληροφορίες είναι δυνατή ανά πάσα στιγμή. Ωστόσο, αυτό το ιδανικό σενάριο σπάνια συναντάται στην πράξη, καθώς εμπλέκονται διάφοροι παράγοντες όπως σφάλματα, κακόβουλος κώδικας και η ανάγκη προστασίας εμπιστευτικών δεδομένων. Δυστυχώς, η προστασία μνήμης στηρίζεται σε εικονική μνήμη, κάτι που επιφέρει κόστος επίδοσης στην διαχείριση του πίνακα σελίδων. Επιπλέον αυτή η μέθοδος προστασίας δεν είναι αρκετά ικανοποιητική για το σενάριο μας καθότι εμφανίζει περιορισμούς επίδοσης και ευπάθειες.

Ενώ η χρήση δικαιωμάτων πρόσβασης μπορεί να είναι η πιο απλή προσέγγιση για τη διασφάλιση της προστασίας, είναι συχνά ανεπαρκής. Κατά συνέπεια, αυτή η διπλωματική διερευνά την εναλλακτική λύση της χρήσης προστασίας μνήμης αντί για δικαιώματα αρχείων. Η εστίαση θα είναι στη διαχείριση ενός χώρου αποθήκευσης κλειδιού-τιμής, διασφαλίζοντας ότι τα ζεύγη κλειδιών προστατεύονται με ελάχιστη επιβάρυνση. Ένα αξιοσημείωτο πλεονέκτημα των συστημάτων αποθήκευσης κλειδιού-τιμής είναι η ταχύτητα πρόσβασης στα δεδομένα. Επειδή το κλειδί λειτουργεί ως μοναδικός αναγνωριστικός δείκτης, η αναζήτηση και η πρόσβαση σε μια συγκεκριμένη τιμή είναι γρήγορη και αποτελεσματική.

Επιπλέον, θα συζητηθούν οι προκλήσεις που αντιμετωπίστηκαν κατά τη διάρκεια της ερευνητικής διαδικασίας, καθώς και οι μεθοδολογίες που χρησιμοποιήθηκαν για την

αντιμετώπισή τους. Τέλος, θα αξιολογηθεί η επίδοση με την χρήση μηχανισμών για εξασφάλιση της ασφάλειας του key-value store καθώς και χωρίς με αποκορύφωμα τη διατύπωση οριστικών συμπερασμάτων.

ΠΕΡΙΕΧΟΜΕΝΑ

Λίστα από φιγούρες

Κεφάλαιο 1

Εισαγωγή

1.1 Περιγραφή προβλήματος

Μη πτητική μνήμη και η σημασία της

Στο πεδίο εφαρμογής της αναδυόμενης τεχνολογίας μνήμης, η επίμονη μνήμη, που συχνά χαρακτηρίζεται ως μη πτητική κύρια μνήμη (NVMM), έχει αποκτήσει εξέχουσα θέση. Αυτή η καινοτόμος τεχνολογία συνδυάζει τη διευθυνσιοδότηση byte της μνήμης τυχαίας προσπέλασης (RAM) με την ανθεκτικότητα των δίσκων. Ως εκ τούτου, παρουσιάζει μια βιώσιμη λύση για εφαρμογές που απαιτούν ανακτήσιμη μνήμη υψηλής απόδοσης, συμπεριλαμβανομένων βάσεων δεδομένων, εκτεταμένων επιστημονικών υπολογισμών και υπηρεσιών cloud.

Η λήψη αποφάσεων βάσει δεδομένων είναι υψίστης σημασίας για την επιτυχία ενός οργανισμού. Δεδομένης της αναμενόμενης αύξησης του όγκου δεδομένων και της αντίστοιχης ζήτησης για επεξεργασία σε πραγματικό χρόνο, είναι σημαντικό για τις επιχειρήσεις να διαχειρίζονται αποτελεσματικά τις αυξανόμενες ποσότητες δεδομένων που απαιτούν άμεση αποθήκευση στον επεξεργαστή για ταχείες, αξιοποιήσιμες πληροφορίες. Από την άποψη της βαθμίδας δεδομένων, η αυξανόμενη ζήτηση για αναλύσεις σε πραγματικό χρόνο υποδηλώνει ότι η αύξηση του μεγάλου όγκου δεδομένων ενδέχεται να μην καλύπτεται από την υπάρχουσα υποδομή πληροφορικής.

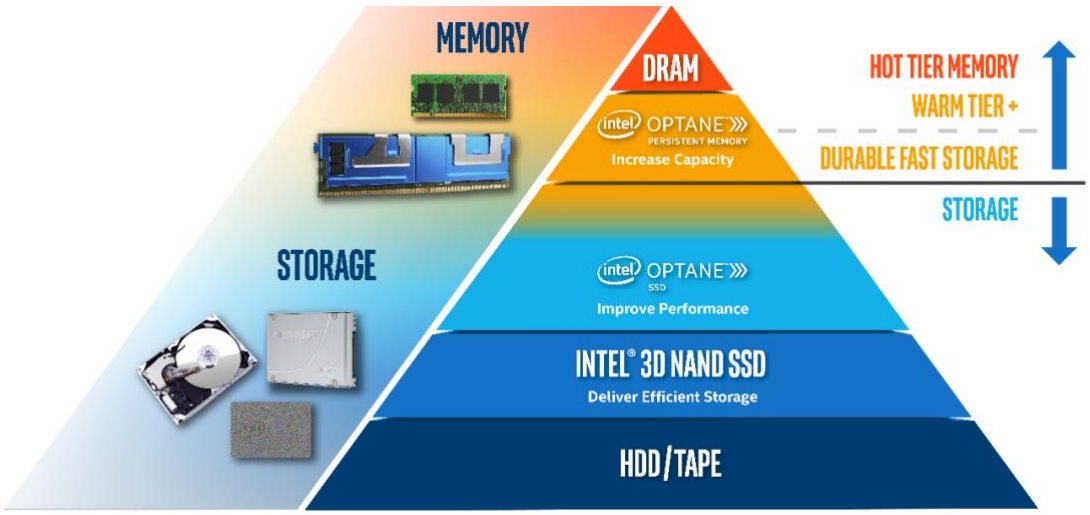

Figure 1Memory hierarchy abstracted

Η εξάλειψη των σημείων συμφόρησης και ο συγχρονισμός των τεχνολογιών για την ενίσχυση συγκεκριμένων φόρτων εργασίας είναι κρίσιμα βήματα για τους οργανισμούς να εκτελέσουν με επιτυχία καινοτόμες στρατηγικές δεδομένων. Ενώ τα συστήματα που βασίζονται σε DRAM προσφέρουν μνήμη με υψηλό κόστος, η χωρητικότητά τους δεν κλιμακώνεται αρκετά γρήγορα για να φιλοξενήσει τον αυξανόμενο όγκο των hot data. Παρόλο που οι SSD NAND διαθέτουν τη χωρητικότητα και τη δομή κόστους για κλιμάκωση, η υποκείμενη τεχνολογία τους έχει σχεδιαστεί ως αποθήκευση με δυνατότητα διεύθυνσης μπλοκ, καθιστώντας τους ακατάλληλους για λειτουργίες μνήμης.

Παρόλα αυτά, υπάρχει μια τεχνολογία που μη πτητική μνήμη με χαμηλότερο κόστος χωρητικότητας σε σύγκριση με τη DRAM, διατηρώντας παράλληλα χαμηλότερες καθυστερήσεις και υψηλότερη αντοχή από τη NAND. Η τεχνολογία Intel Optane, κατασκευασμένη από έναν ξεχωριστό συνδυασμό υλικών και χρησιμοποιούμενη σε μονάδες solid state Intel Optane, έχει ήδη δημιουργήσει ένα επίπεδο αποθήκευσης υψηλού εύρους ζώνης και χαμηλής καθυστέρησης. Η μη πτητική μνήμη Intel Optane επαναπροσδιορίζει την παραδοσιακή αρχιτεκτονική προσφέροντας ένα εκτεταμένο και

ανθεκτικό επίπεδο μνήμης με προσιτό κόστος, μεγιστοποιώντας τη χρήση των δεδομένων εργασίας. Αυτή η τεχνολογία διατίθεται σε χωρητικότητες έως 512 GB ανά μονάδα μνήμης και διαθέτει ασφάλεια υλικού και εταιρική (enterprise) αξιοπιστία. Ενσωματώνοντας υψηλή απόδοση, χαμηλό χρόνο απόκρισης (low latency) και μη πτητική με άμεση προσβασιμότητα στο δίαυλο μνήμης, η μη πτητική Intel Optane είναι συμβατή με συγκεκριμένους κλιμακούμενους επεξεργαστές Intel Xeon δεύτερης και τρίτης γενιάς.

Ωστόσο, η εγγύηση της ασφάλειας και της προστασίας των δεδομένων σε συστήματα αρχείων λειτουργίας χρήστη για μη πτητική παραμένει μια πρόκληση.

Η παρούσα διπλωματική εργασία επικεντρώνεται στην αξιολόγηση μηχανισμών προστασίας και ασφάλειας σε συστήματα αρχείων λειτουργίας χρήστη για επίμονη μνήμη, στοχεύοντας συγκεκριμένα στον εντοπισμό της πιο αποτελεσματικής τεχνολογίας για την παροχή προστασίας μνήμης που μπορεί να υιοθετηθεί από την Optane. Η περιγραφή του προβλήματος χωρίζεται στις ακόλουθες ενότητες:

1. Προκλήσεις στην μη πτητική προστασία μνήμης
2. Υφιστάμενοι μηχανισμοί ασφαλείας και οι περιορισμοί τους
3. Προτεινόμενη μεθοδολογία για την προστασία της μνήμης
4. Μετρήσεις αξιολόγησης και δοκιμές

1. Προκλήσεις στην επίμονη προστασία μνήμης

Η τεχνολογία μη πτητική μνήμης έχει κερδίσει σημαντική προσοχή λόγω της δυνατότητάς της να φέρει επανάσταση στον χώρο των υπολογιστών. Προσφέροντας μη πτητική αποθήκευση που διατηρεί δεδομένα ακόμα και μετά από διακοπή ρεύματος, η μη πτητική μνήμη γεφυρώνει το χάσμα μεταξύ της παραδοσιακής μνήμης και των συσκευών αποθήκευσης. Αυτή η καινοτόμος τεχνολογία προσφέρει πολλά οφέλη, όπως γρήγορη πρόσβαση σε δεδομένα, βελτιωμένη απόδοση συστήματος και μειωμένη κατανάλωση ενέργειας. Ωστόσο, όπως συμβαίνει με κάθε αναδυόμενη τεχνολογία, υπάρχουν προκλήσεις που σχετίζονται με τη διασφάλιση της επίμονης

μνήμης. Αυτή η πτυχιακή εργασία εμβαθύνει στις βασικές ανησυχίες σχετικά με την μη πτητική προστασία της μνήμης και επισημαίνει τους κύριους τομείς εστίασης.

1.1 Σφάλματα και αποτυχίες συστήματος:

Μια σημαντική πρόκληση για την προστασία της επίμονης μνήμης έγκειται στην αντιμετώπιση σφαλμάτων και αποτυχιών του συστήματος. Τα συστήματα μόνιμης μνήμης είναι εγγενώς ευαίσθητα σε ζητήματα που θα μπορούσαν να θέσουν σε κίνδυνο την ακεραιότητα και την προσβασιμότητα των δεδομένων. Αυτά τα προβλήματα μπορεί να προέρχονται από ανωμαλίες λογισμικού ή υλικού, με πιθανό αποτέλεσμα την καταστροφή ή την απώλεια δεδομένων.

Σε σενάρια που βασίζονται σε λογισμικό, σφάλματα μπορούν να εισαχθούν κατά τη διάρκεια της διαδικασίας ανάπτυξης ή να προκύψουν από ακούσιες αλληλεπιδράσεις μεταξύ components λογισμικού. Αυτό μπορεί να οδηγήσει σε μη πτητικά συστήματα μνήμης που συμπεριφέρονται απροσδόκητα, προκαλώντας καταστροφή δεδομένων και θέτοντας σε κίνδυνο τη σταθερότητα του συστήματος. Για να μετριάσουν αυτόν τον κίνδυνο, οι προγραμματιστές πρέπει να χρησιμοποιούν αυστηρές μεθόδους δοκιμών και επικύρωσης, διασφαλίζοντας ότι τα στοιχεία λογισμικού είναι πλήρως συμβατά με την τεχνολογία μόνιμης μνήμης που χρησιμοποιείται.

Τα ζητήματα που σχετίζονται με το υλικό μπορούν επίσης να επηρεάσουν τα μη πτητικά συστήματα μνήμης, καθώς μπορεί να περιλαμβάνουν κατασκευαστικά ελαττώματα, φθορά εξαρτημάτων ή ηλεκτρομαγνητικές παρεμβολές. Για την αντιμετώπιση αυτών των ανησυχιών, τα εξαρτήματα υλικού θα πρέπει να σχεδιάζονται έτσι ώστε να αντέχουν σε αντίξοες συνθήκες και να λειτουργούν αξιόπιστα σε διάφορα περιβάλλοντα. Επιπλέον, πρέπει να εφαρμοστούν εργαλεία παρακολούθησης και διάγνωσης για τον έγκαιρο εντοπισμό και επίλυση προβλημάτων υλικού, αποτρέποντας πιθανή απώλεια ή καταστροφή δεδομένων.

1.2 Κακόβουλος κώδικας και επιθέσεις:

Μια άλλη πρόκληση στην επίμονη προστασία της μνήμης είναι η προστασία των συστημάτων από κακόβουλο κώδικα και κυβερνοεπιθέσεις. Καθώς η επίμονη μνήμη γίνεται όλο και πιο διαδεδομένη, είναι πιθανό να γίνει πιο ελκυστικός στόχος για τους εγκληματίες του κυβερνοχώρου που επιδιώκουν να εκμεταλλευτούν τρωτά σημεία για μη εξουσιοδοτημένη πρόσβαση ή χειραγώγηση δεδομένων.

Οι κυβερνοεπιθέσεις σε επίμονα συστήματα μνήμης μπορούν να λάβουν διάφορες μορφές, συμπεριλαμβανομένων άμεσων επιθέσεων στο υλικό μνήμης, εισαγωγής κακόβουλου κώδικα ή εκμετάλλευσης τρωτών σημείων λογισμικού. Αυτές οι επιθέσεις μπορεί να έχουν σοβαρές συνέπειες, όπως παραβιάσεις δεδομένων, διαρροές ή μη εξουσιοδοτημένη πρόσβαση σε ευαίσθητες πληροφορίες.

Για την προστασία από αυτές τις απειλές, είναι απαραίτητο να χρησιμοποιούνται ισχυρά μέτρα ασφαλείας σε όλη τη διαδικασία σχεδιασμού και υλοποίησης του συστήματος. Αυτό περιλαμβάνει τη χρήση ασφαλών πρακτικών κωδικοποίησης, την εφαρμογή ισχυρών ελέγχων πρόσβασης και τη διεξαγωγή τακτικών αξιολογήσεων ευπάθειας. Επιπλέον, η χρήση τεχνικών κρυπτογράφησης και ελέγχου ταυτότητας μπορεί να προστατεύσει περαιτέρω τα δεδομένα που είναι αποθηκευμένα στη μόνιμη μνήμη, διασφαλίζοντας ότι μόνο εξουσιοδοτημένοι χρήστες έχουν πρόσβαση σε ευαίσθητες πληροφορίες.

1.3 Εμπιστευτικότητα και Ιδιωτικότητα:

Η προστασία της εμπιστευτικότητας και της ιδιωτικότητας των δεδομένων που αποθηκεύονται σε συστήματα μόνιμης μνήμης είναι υψίστης σημασίας, ιδίως δεδομένης της ευαίσθητης φύσης των πληροφοριών που αποθηκεύονται συχνά σε αυτές τις συσκευές. Η μη εξουσιοδοτημένη πρόσβαση σε εμπιστευτικά δεδομένα μπορεί να οδηγήσει σε οικονομικές απώλειες, ζημιά στη φήμη και πιθανές νομικές συνέπειες για τους οργανισμούς.

Για να διατηρηθεί η εμπιστευτικότητα και το απόρρητο των δεδομένων στη μόνιμη μνήμη, οι οργανισμοί πρέπει να εφαρμόζουν ολοκληρωμένα μέτρα ασφαλείας. Αυτό περιλαμβάνει μηχανισμούς ελέγχου πρόσβασης που περιορίζουν την πρόσβαση σε εξουσιοδοτημένους χρήστες και διασφαλίζουν ότι τα ευαίσθητα δεδομένα δεν εκτίθενται ακούσια. Θα πρέπει επίσης να χρησιμοποιείται κρυπτογράφηση δεδομένων, καθιστώντας τις πληροφορίες μη αναγνώσιμες σε μη εξουσιοδοτημένα άτομα, ακόμη και αν καταφέρουν να αποκτήσουν πρόσβαση στη συσκευή αποθήκευσης.

1.2 Σχετικές εργασίες

2. Υφιστάμενοι μηχανισμοί ασφαλείας και οι περιορισμοί τους

Καθώς η τεχνολογία μη πτητικής μνήμης κερδίζει έδαφος, η διασφάλιση των δεδομένων που είναι αποθηκευμένα σε αυτά τα συστήματα έχει γίνει μια κρίσιμη ανησυχία. Ενώ υπάρχουν διάφοροι μηχανισμοί ασφαλείας διαθέσιμοι για την προστασία των δεδομένων στη μόνιμη μνήμη, κάθε μέθοδος έχει τους περιορισμούς της. Σε αυτή την διπλωματική, θα συζητήσουμε τους υπάρχοντες μηχανισμούς ασφαλείας, τις αδυναμίες τους και τις προκλήσεις στην παροχή ολοκληρωμένης προστασίας για συστήματα μόνιμης μνήμης.

2.1 Δικαιώματα αρχείου:

Τα δικαιώματα αρχείων είναι ένας θεμελιώδης μηχανισμός ασφαλείας που χρησιμοποιείται σε διάφορα λειτουργικά συστήματα για τον έλεγχο της πρόσβασης σε αρχεία και καταλόγους. Καθορίζοντας τα δικαιώματα πρόσβασης για διαφορετικούς χρήστες και ομάδες, τα δικαιώματα αρχείων μπορούν να παρέχουν ένα βασικό επίπεδο προστασίας από μη εξουσιοδοτημένη πρόσβαση σε δεδομένα που είναι αποθηκευμένα σε συστήματα μη πτητικής μνήμης.

Ωστόσο, τα δικαιώματα αρχείων έχουν αρκετούς περιορισμούς:

- Βασίζονται στους μηχανισμούς ελέγχου πρόσβασης του λειτουργικού συστήματος, οι οποίοι ενδέχεται να μην είναι αρκετά ισχυροί για να αμυνθούν έναντι εξελιγμένων επιθέσεων ή τρωτών σημείων.
- Τα δικαιώματα αρχείων παρέχουν έλεγχο πρόσβασης μόνο σε επίπεδο αρχείου και ενδέχεται να μην είναι αρκετά λεπτομερή για την ασφάλεια ευαίσθητων δεδομένων σε μεμονωμένα αρχεία.
- Υπόκεινται σε εσφαλμένες διαμορφώσεις ή ανθρώπινα λάθη, εκθέτοντας ενδεχομένως ευαίσθητα δεδομένα σε μη εξουσιοδοτημένη πρόσβαση ακούσια.
- Τα δικαιώματα αρχείων δεν προστατεύουν τα δεδομένα όταν ένας εισβολέας αποκτά πρόσβαση στη φυσική συσκευή αποθήκευσης ή όταν το μέσο αποθήκευσης κλαπεί ή χαθεί.
- Ως αποτέλεσμα, ενώ τα δικαιώματα αρχείων μπορούν να προσφέρουν κάποια προστασία, συχνά δεν επαρκούν για την παροχή ολοκληρωμένης ασφάλειας για συστήματα μόνιμης μνήμης.

2.2 Κρυπτογράφηση:

Η κρυπτογράφηση είναι ένας άλλος ευρέως χρησιμοποιούμενος μηχανισμός ασφαλείας για την προστασία των δεδομένων σε συστήματα μόνιμης μνήμης. Κωδικοποιώντας τα δεδομένα χρησιμοποιώντας κρυπτογραφικούς αλγόριθμους, η κρυπτογράφηση καθιστά τις πληροφορίες μη αναγνώσιμες χωρίς το κατάλληλο κλειδί αποκρυπτογράφησης, προστατεύοντάς τις από μη εξουσιοδοτημένη πρόσβαση.

Ωστόσο, η κρυπτογράφηση έχει επίσης τους περιορισμούς της:

- **Επιβάρυνση απόδοσης:** Οι διαδικασίες κρυπτογράφησης και αποκρυπτογράφησης μπορούν να εισαγάγουν γενικά έξοδα απόδοσης, επηρεάζοντας τη συνολική απόδοση του συστήματος. Αυτό είναι ιδιαίτερα

προβληματικό σε συστήματα μη πτητικής μνήμης, όπου η χαμηλή καθυστέρηση και η πρόσβαση υψηλής ταχύτητας είναι απαραίτητες.

- **Διαχείριση κλειδιών:** Η κρυπτογράφηση βασίζεται στην ασφαλή διαχείριση κλειδιών, η εφαρμογή και η συντήρηση της οποίας μπορεί να είναι δύσκολη. Εάν η διαχείριση των κλειδιών κρυπτογράφησης δεν γίνεται σωστά, ενδέχεται να είναι ευάλωτα σε μη εξουσιοδοτημένη πρόσβαση ή παραβίαση.
- **Ελλιπής προστασία:** Ενώ η κρυπτογράφηση μπορεί να προστατεύσει την εμπιστευτικότητα των δεδομένων, ενδέχεται να μην προστατεύει πάντα από άλλους φορείς επίθεσης, όπως επιθέσεις ακεραιότητας δεδομένων ή μη εξουσιοδοτημένη τροποποίηση μεταδεδομένων.
- **Ευπάθειες:** Οι αλγόριθμοι κρυπτογράφησης και οι υλοποιήσεις ενδέχεται να έχουν ευπάθειες που μπορούν να εκμεταλλευτούν οι επιτιθέμενοι, υπονομεύοντας ενδεχομένως την ασφάλεια των κρυπτογραφημένων δεδομένων.

1.3 Το πεδίο μελέτης μας

3. Προτεινόμενη μεθοδολογία για την προστασία της μνήμης

Οι μη πτητικές τεχνολογίες μνήμης, όπως ή Optane της Intel, έχουν αναδειχθεί ως μια επαναστατική λύση για τη γεφύρωση του χάσματος μεταξύ της παραδοσιακής μνήμης και των συσκευών αποθήκευσης. Προσφέροντας μη πτητική αποθήκευση που διατηρεί δεδομένα ακόμη και μετά από απώλεια ηλεκτρισμού, αυτές οι τεχνολογίες παρέχουν γρήγορη πρόσβαση σε δεδομένα, βελτιωμένη απόδοση συστήματος και μειωμένη κατανάλωση ενέργειας. Ωστόσο, η ασφάλεια των δεδομένων σε συστήματα μόνιμης μνήμης παραμένει μια σημαντική πρόκληση. Η παρούσα διπλωματική εργασία επικεντρώνεται στις ακόλουθες βασικές πτυχές, συμπεριλαμβανομένης της διαχείρισης χώρου αποθήκευσης κλειδιού-τιμής, των μηχανισμών προστασίας μνήμης, της αντιμετώπισης προκλήσεων και μεθοδολογιών και της διατήρησης της επεκτασιμότητας και της απόδοσης για μη πτητική μνήμη (ειδικά για την Optane).

3.1 Διαχείριση χώρου αποθήκευσης κλειδιού-τιμής

Η προτεινόμενη μεθοδολογία στοχεύει στη διαχείριση ενός μεγάλου συστήματος αρχείων ως χώρου αποθήκευσης κλειδιού-τιμής, διασφαλίζοντας ότι τα ζεύγη κλειδιών προστατεύονται με ελάχιστη επιβάρυνση. Αυτή η προσέγγιση απλοποιεί τη διαδικασία διαχείρισης δεδομένων και επιτρέπει την αποτελεσματική πρόσβαση και ανάκτηση πληροφοριών. Με αυτό εννοούμε τα ακόλουθα οφέλη:

Πλεονεκτήματα της διαχείρισης χώρου αποθήκευσης κλειδιού-τιμής:

- **Απλοποιημένη διαχείριση δεδομένων:** Μια προσέγγιση που βασίζεται στο χώρο αποθήκευσης κλειδιού-τιμής μειώνει την πολυπλοκότητα της διαχείρισης δεδομένων, καθώς απαιτεί μόνο τη διατήρηση ζευγών κλειδιού-τιμής αντί για πιο περίπλοκες δομές αρχείων. Αυτή η απλοποίηση διευκολύνει την αποτελεσματική οργάνωση και ανάκτηση δεδομένων.

- **Ελάχιστη επιβάρυνση:** Εστιάζοντας σε ζεύγη κλειδιού-τιμής, η προτεινόμενη μεθοδολογία ελαχιστοποιεί τα γενικά έξοδα που σχετίζονται με τη διαχείριση

δεδομένων, διασφαλίζοντας ότι οι πόροι του συστήματος χρησιμοποιούνται αποτελεσματικά και αποφεύγοντας την περιττή υποβάθμιση της απόδοσης.

- **Ευέλικτη και επεκτάσιμη:** Η διαχείριση χώρου αποθήκευσης κλειδιού-τιμής είναι εγγενώς ευέλικτη και επεκτάσιμη, καθώς μπορεί εύκολα να φιλοξενήσει διάφορους τύπους δεδομένων και μεγέθη. Αυτή η προσαρμοστικότητα επιτρέπει στην προτεινόμενη μεθοδολογία να υποστηρίζει ένα ευρύ φάσμα εφαρμογών και φόρτου εργασίας.

3.2 Μηχανισμοί προστασίας μνήμης:

Οι παραδοσιακοί μηχανισμοί ασφαλείας, όπως τα δικαιώματα αρχείων, συχνά παρέχουν ανεπαρκή προστασία για μόνιμα συστήματα μνήμης καθότι εξαρτούνται από το λειτουργικό σύστημα το οποίο πιθανόν να παρουσιάζει ευπάθειες. Επιπλέον παρουσιάζουν κάποια έξοδα (overhead) απόδοσης. Αυτά περιλαμβάνουν τα γενικά έξοδα μετάφρασης, τα οποία είναι η ανάγκη αντιστοίχισης εικονικών διευθύνσεων σε φυσικές για κάθε πρόσβαση μνήμης, συχνά απαιτώντας αναζητήσεις στον πίνακα σελίδων. Η μνήμη που καταναλώνεται από τους ίδιους τους πίνακες σελίδων μπορεί να είναι σημαντική, ειδικά για συστήματα με μεγάλους χώρους διευθύνσεων ή ακόμα και για συστήματα που χρησιμοποιούν μικρά μεγέθη σελίδων. Επιπλέον, το λειτουργικό σύστημα πρέπει να διαχειρίζεται αυτούς τους πίνακες σελίδων. Τέλος, οι εναλλαγές περιβάλλοντος απαιτούν επίσης την εναλλαγή των σχετικών πινάκων σελίδων, οδηγώντας ενδεχομένως σε ακύρωση των προσωρινά αποθηκευμένων μεταφράσεων και αύξηση του κόστους μετάφρασης έως ότου συμπληρωθεί ξανά η προσωρινή μνήμη. Για την αντιμετώπιση των πιο πάνω, η προτεινόμενη μεθοδολογία διερευνά πιο ισχυρούς και αποδοτικούς μηχανισμούς προστασίας της μνήμης. Ένας τέτοιος μηχανισμός είναι τα κλειδιά προστασίας μνήμης (MPK) της Intel, τα οποία μπορούν να βοηθήσουν στην αποτροπή μη εξουσιοδοτημένης πρόσβασης, παραβίασης ή καταστροφής δεδομένων που είναι αποθηκευμένα στη μόνιμη μνήμη.

3.2.1 Κλειδιά προστασίας μνήμης Intel (MPK):

Τα κλειδιά προστασίας μνήμης (MPK) της Intel είναι μια δυνατότητα που βασίζεται σε υλικό και παρέχει λεπτομερή (fine-grained) έλεγχο των δικαιωμάτων πρόσβασης στη μνήμη. Τα MPK επιτρέπουν στο λειτουργικό σύστημα να συσχετίζει συγκεκριμένα δικαιώματα πρόσβασης με κάθε περιοχή μνήμης, προσφέροντας έτσι έναν πιο ισχυρό μηχανισμό ασφαλείας σε σύγκριση με τα παραδοσιακά δικαιώματα αρχείων.

3.2.2 Πλεονεκτήματα MPK:

- **Λεπτομερής (fine-grained) έλεγχος πρόσβασης:** Ο MPK μηχανισμός επιτρέπει τον ακριβή έλεγχο της πρόσβασης στη μνήμη, επιτρέποντας τον ορισμό συγκεκριμένων δικαιωμάτων για κάθε περιοχή μνήμης. Η λεπτομέρεια περιορίζεται από το μέγεθος των σελίδων μνήμης. Με άλλα λόγια, το MPK επιτρέπει να ελέγχετε την πρόσβαση ανά σελίδα, όχι ανά byte ή ανά λέξη. Αυτή η λεπτομέρεια διασφαλίζει ότι μόνο εξουσιοδοτημένες διεργασίες μπορούν να έχουν πρόσβαση ή να τροποποιούν ευαίσθητα δεδομένα που είναι αποθηκευμένα στη μόνιμη μνήμη.
- **Χαμηλό κόστος:** Ως μηχανισμός που βασίζεται σε υλικό, το MPK εισάγει ελάχιστη επιβάρυνση απόδοσης σε σύγκριση με τα μέτρα ασφαλείας που βασίζονται σε λογισμικό. Αυτό επιτρέπει στα μη πτητικά συστήματα μνήμης να διατηρούν υψηλή απόδοση, ενισχύοντας παράλληλα την ασφάλεια.
- **Δυναμική διαχείριση δικαιωμάτων:** Το MPK επιτρέπει τη δυναμική τροποποίηση των δικαιωμάτων πρόσβασης στη μνήμη κατά το χρόνο εκτέλεσης. Αυτή η ευελιξία επιτρέπει στο λειτουργικό σύστημα να προσαρμόζεται στις μεταβαλλόμενες απαιτήσεις ασφαλείας και να προστατεύει τα ευαίσθητα δεδομένα πιο αποτελεσματικά.
- **Ασφάλεια σε επίπεδο υλικού:** Εφαρμόζοντας ασφάλεια σε επίπεδο υλικού, το MPK παρέχει προστασία από ένα ευρύ φάσμα επιθέσεων που βασίζονται σε λογισμικό, όπως εκμεταλλεύσεις υπερχείλισης buffer ή έγχυση κώδικα (code injection).

3.2.3 Ενσωμάτωση με τη διαχείριση Key-Value store:

Η ενσωμάτωση των κλειδιών προστασίας μνήμης της Intel με την προτεινόμενη μεθοδολογία διαχείρισης χώρου αποθήκευσης κλειδιού-τιμής μπορεί να ενισχύσει περαιτέρω την ασφάλεια των συστημάτων μόνιμης μνήμης. Εκχωρώντας μοναδικά δικαιώματα πρόσβασης σε κάθε ζεύγος κλειδιού-τιμής στο store, το MPK μπορεί να αποτρέψει τη μη εξουσιοδοτημένη πρόσβαση και την αλλοίωση δεδομένων πιο αποτελεσματικά από το να βασίζεται μόνο στα δικαιώματα αρχείων. Επιπλέον, ο συνδυασμός MPK και διαχείρισης χώρου αποθήκευσης κλειδιού-τιμής μπορεί να διατηρήσει υψηλή απόδοση και επεκτασιμότητα, διασφαλίζοντας ότι δεν διακυβεύονται τα οφέλη της μόνιμης μνήμης.

3.2.4 Προκλήσεις και εκτιμήσεις:

Ενώ το MPK προσφέρει πολλά πλεονεκτήματα για την εξασφάλιση συστημάτων μη πτητικής μνήμης, υπάρχουν προκλήσεις και ζητήματα που πρέπει να αντιμετωπιστούν:

- **Συμβατότητα:** Το MPK είναι μια δυνατότητα που βασίζεται σε υλικό ειδικά για επεξεργαστές Intel. Η διασφάλιση της συμβατότητας με άλλες αρχιτεκτονικές επεξεργαστών ή λύσεις μη πτητικής μνήμης που δεν είναι Optane ενδέχεται να απαιτεί πρόσθετη έρευνα και ανάπτυξη.
- **Διαχείριση κλειδιών:** Όταν χρησιμοποιείται το MPK σε συνδυασμό με τη διαχείριση καταστήματος κλειδιού-τιμής, η σωστή διαχείριση κλειδιών γίνεται κρίσιμη. Η αποτελεσματική αποθήκευση, ανάκτηση και ενημέρωση κλειδιών πρόσβασης με παράλληλη διατήρηση της ασφάλειας μπορεί να είναι μια πολύπλοκη πρόκληση.
- **Συνύπαρξη με άλλους μηχανισμούς ασφαλείας:** Η ενσωμάτωση του MPK με άλλους μηχανισμούς ασφαλείας, όπως η κρυπτογράφηση ή οι λίστες ελέγχου πρόσβασης, μπορεί να δημιουργήσει πολυπλοκότητες και πιθανές συγκρούσεις που απαιτούν προσεκτική εξέταση και σχεδιασμό.

4. Μετρήσεις αξιολόγησης και δοκιμές

Για την αξιολόγηση της αποτελεσματικότητας της προτεινόμενης μεθοδολογίας προστασίας της μνήμης, θα χρησιμοποιηθούν οι ακόλουθες μετρήσεις αξιολόγησης και προσεγγίσεις δοκιμών:

4.1 Ασφάλεια και ανθεκτικότητα:

Η προτεινόμενη μεθοδολογία θα αξιολογηθεί για την ικανότητά της να παρέχει ολοκληρωμένη ασφάλεια και προστασία από διάφορες επιθέσεις και τρωτά σημεία. Αυτή η αξιολόγηση θα περιλαμβάνει εξέταση της αντίστασης σε μη εξουσιοδοτημένη πρόσβαση, αλλοίωση δεδομένων και διαφθορά.

4.2 Απόδοση και κόστος:

Θα αναλυθεί ο αντίκτυπος των προτεινόμενων μηχανισμών προστασίας μνήμης στην απόδοση του συστήματος και το κόστος (γενικά). Αυτή η ανάλυση θα περιλαμβάνει τη συγκριτική αξιολόγηση της απόδοσης του συστήματος με και χωρίς τους μηχανισμούς προστασίας της μνήμης.

4.3 Επεκτασιμότητα και αποτελεσματικότητα:

Η επεκτασιμότητα και η αποτελεσματικότητα της προτεινόμενης μεθοδολογίας θα αξιολογηθούν στο πλαίσιο της διαχείρισης μεγάλων συστημάτων αρχείων και χώρων αποθήκευσης κλειδιών-τιμών. Αυτή η αξιολόγηση θα εξετάσει παράγοντες όπως η ευκολία εφαρμογής, η ικανότητα χειρισμού αυξανόμενων όγκων δεδομένων και η αποτελεσματικότητα των μηχανισμών προστασίας καθώς το σύστημα κλιμακώνεται.

4.4 Προγράμματα δοκιμών:

Θα εκτελεστούν διάφορα προγράμματα δοκιμών για την αξιολόγηση της εφαρμογής της προτεινόμενης μεθοδολογίας προστασίας μνήμης. Αυτές οι δοκιμές θα παρέχουν πληροφορίες σχετικά με την αποτελεσματικότητα, την ευρωστία και την απόδοση των μηχανισμών προστασίας της μνήμης σε σενάρια πραγματικού κόσμου.

Κεφάλαιο 2

Υπόβαθρο - Προγραμματισμός μη πτητικής μνήμης από τον Steve Scargall

2.1 Εισαγωγή

2.1.1 Μη πτητική τεχνολογία μνήμης

Η μη πτητική μνήμη είναι τεχνολογία αποθήκευσης που διατηρεί δεδομένα ακόμα και μετά την αφαίρεση της τροφοδοσίας. Έχει τη δυνατότητα να γεφυρώσει το χάσμα απόδοσης μεταξύ της παραδοσιακής πτητικής μνήμης (π.χ. DRAM) και των συσκευών αποθήκευσης (π.χ. HDD και SSD) προσφέροντας υψηλή απόδοση, διευθυνσιοδότηση byte και διατήρηση δεδομένων.

Οι εφαρμογές που βασίζονται στη μη πτητική μνήμη μπορούν να εκτελούνται ταχύτερα λόγω της μειωμένης μεταφοράς δεδομένων μεταξύ της CPU και των πιο αργών συσκευών αποθήκευσης. Οι προγραμματιστές θα πρέπει να αποφασίσουν τη βέλτιστη κατανομή δεδομένων μεταξύ DRAM, μόνιμης μνήμης και αποθήκευσης. Με μεγαλύτερη χωρητικότητα από τη μνήμη DRAM, η μόνιμη μνήμη επιτρέπει την επεξεργασία μεγαλύτερων συνόλων δεδομένων, μειώνοντας τις εισόδους/εξόδους δίσκου και βελτιώνοντας την απόδοση. Η επεξεργασία στη μνήμη με μόνιμη μνήμη

εξαλείφει την ανάγκη για δεδομένα σημείων ελέγχου ή σελιδοποίησης, βελτιώνοντας περαιτέρω την απόδοση.

2.1.2 Βασικά χαρακτηριστικά της μόνιμης μνήμης που πρέπει να λάβουν υπόψη οι προγραμματιστές κατά την αρχιτεκτονική και την ανάπτυξη λύσεων:

Απόδοση: Η μόνιμη μνήμη προσφέρει σημαντικά καλύτερη απόδοση, λανθάνοντα χρόνο και εύρος ζώνης από τις συσκευές αποθήκευσης που βασίζονται σε NAND, όπως οι μονάδες SSD. Ωστόσο, η απόδοσή του μπορεί να είναι ελαφρώς χαμηλότερη από την DRAM.

Ανθεκτικότητα: Σε αντίθεση με τη μνήμη DRAM, η μόνιμη μνήμη διατηρεί τα δεδομένα της ακόμα και μετά την αφαίρεση της τροφοδοσίας. Η αντοχή του ξεπερνά την αποθήκευση NAND και αναμένεται να διαρκέσει για όλη τη διάρκεια ζωής του διακομιστή χωρίς να φθείρεται.

Χωρητικότητα: Οι μόνιμες μονάδες μνήμης μπορούν να έχουν πολύ μεγαλύτερες χωρητικότητες από τις DIMM DRAM και μπορούν να συνυπάρχουν στα ίδια κανάλια μνήμης.

Επιτόπιες (in place) ενημερώσεις δεδομένων: Οι εφαρμογές μπορούν να ενημερώνουν δεδομένα απευθείας στη μόνιμη μνήμη χωρίς την ανάγκη σειριοποίησης και αποσειριοποίησης των δεδομένων.

Διευθυνσιοδότηση byte: Όπως και η DRAM, η μόνιμη μνήμη είναι διευθυνσιοδοτούμενη από byte, επιτρέποντας στις εφαρμογές να ενημερώνουν μόνο τα απαραίτητα δεδομένα χωρίς την επιβάρυνση των λειτουργιών ανάγνωσης-τροποποίησης-εγγραφής.

Συνοχή cache CPU: Τα δεδομένα στη μόνιμη μνήμη είναι συνεκτικά με τη μνήμη cache της CPU , διασφαλίζοντας ότι τα δεδομένα παραμένουν συνεπή σε όλες τις κρυφές μνήμες της CPU και τη μόνιμη μνήμη.

Λειτουργίες DMA και RDMA: Η μόνιμη μνήμη υποστηρίζει λειτουργίες άμεσης προσπέλασης μνήμης (DMA) και απομακρυσμένης άμεσης προσπέλασης μνήμης (RDMA), επιτρέποντας αποτελεσματικές μεταφορές δεδομένων μεταξύ μνήμης και συσκευών.

Διατήρηση δεδομένων: Τα δεδομένα που εγγράφονται στη μόνιμη μνήμη δεν χάνονται όταν αφαιρείται η τροφοδοσία, διασφαλίζοντας την ανθεκτικότητα των δεδομένων.

Άμεση πρόσβαση στο χώρο χρήστη: Μετά τους ελέγχους δικαιωμάτων, τα δεδομένα που είναι αποθηκευμένα στη μόνιμη μνήμη μπορούν να προσπελαστούν απευθείας από το χώρο χρήστη, παρακάμπτοντας τον κώδικα πυρήνα, τις κρυφές μνήμες σελίδων του συστήματος αρχείων και τις διακοπές.

Άμεση διαθεσιμότητα δεδομένων: Τα δεδομένα σχετικά με τη μόνιμη μνήμη είναι άμεσα διαθέσιμα μόλις ενεργοποιηθεί το σύστημα. Οι εφαρμογές δεν χρειάζεται να ξοδεύουν χρόνο για να “ζεστάνουν” τις κρυφές μνήμες και μπορούν να έχουν πρόσβαση στα δεδομένα αμέσως μετά τη χαρτογράφηση μνήμης.

Χωρίς αποτύπωμα DRAM: Τα δεδομένα που βρίσκονται στη μόνιμη μνήμη δεν καταλαμβάνουν χώρο DRAM, εκτός εάν η εφαρμογή αντιγράψει ρητά τα δεδομένα σε DRAM για ταχύτερη πρόσβαση.

Τοπική αποθήκευση δεδομένων: Τα δεδομένα που εγγράφονται σε μόνιμες μονάδες μνήμης είναι τοπικά στο σύστημα. Οι εφαρμογές είναι υπεύθυνες για την αναπαραγωγή δεδομένων σε πολλαπλά συστήματα, εάν είναι απαραίτητο.

2.1.3 Σημασία και εφαρμογές της μη πτητικής μνήμης

Η σημασία της μη πτητικής (persistent) μνήμης είναι εμφανής σε ένα ευρύ φάσμα εφαρμογών. Στα συστήματα βάσεων δεδομένων, μπορεί να επιταχύνει την επεξεργασία συναλλαγών και να μειώσει τους χρόνους ανάκτησης μετά από διακοπές ρεύματος. Η υπολογιστική υψηλών επιδόσεων (high performance computing) μπορεί να επωφεληθεί από την επίμονη μνήμη βελτιώνοντας τις ταχύτητες επεξεργασίας

δεδομένων και ενεργοποιώντας υπολογισμούς στη μνήμη. Η ανάλυση μεγάλων δεδομένων μπορεί να χρησιμοποιήσει αυτήν την τεχνολογία για να χειριστεί αποτελεσματικά μεγάλα σύνολα δεδομένων, ενώ οι εφαρμογές τεχνητής νοημοσύνης και μηχανικής μάθησης μπορούν να την αξιοποιήσουν για ταχύτερη εκπαίδευση μοντέλων και εξαγωγή συμπερασμάτων. Επιπλέον, το Internet of Things (IoT) μπορεί να επωφεληθεί από τα χαρακτηριστικά χαμηλής καθυστέρησης και ενεργειακής απόδοσης της επίμονης μνήμης.

2.2 Ιεραρχία μνήμης

2.2.1 Ιεραρχία προσωρινής μνήμης:

Οι λειτουργίες φόρτωσης και αποθήκευσης (load/store) χρησιμοποιούνται για ανάγνωση και εγγραφή σε μόνιμη μνήμη αντί για εισόδους/εξόδους που βασίζονται σε μπλοκ στην παραδοσιακή αποθήκευση.

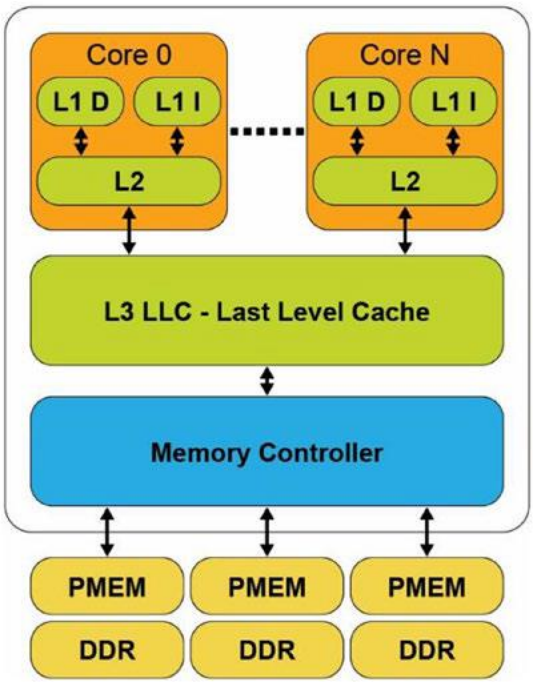

Figure 2 Προσωρινή μνήμη CPU και ιεραρχία μνήμης

Χρησιμοποιώντας την αρχιτεκτονική Intel ως παράδειγμα, η κρυφή μνήμη CPU έχει τρία επίπεδα: L1, L2 και L3. Αυτά τα επίπεδα αντιστοιχούν στην απόσταση από τον πυρήνα της CPU, την ταχύτητα και το μέγεθος της προσωρινής μνήμης. Η κρυφή μνήμη L1 είναι πιο κοντά στην CPU, με την υψηλότερη ταχύτητα αλλά το μικρότερο μέγεθος. Οι κρυφές μνήμες L2 και L3 έχουν μεγαλύτερες χωρητικότητες αλλά είναι πιο αργές.

Μια τυπική μικροαρχιτεκτονική CPU διαθέτει τρία επίπεδα προσωρινής μνήμης και έναν ελεγκτή μνήμης με τρία κανάλια μνήμης, καθένα από τα οποία έχει DRAM και μόνιμη μνήμη. Σε πλατφόρμες χωρίς προστατευμένο τομέα αποτυχίας ρεύματος για κρυφές μνήμες CPU, τα τροποποιημένα δεδομένα που δεν έχουν εγγραφεί στον δίσκο

(flush) θα χαθούν κατά τη διάρκεια απώλειας ισχύος ή διακοπής του συστήματος. Ωστόσο, οι πλατφόρμες με προστατευμένο τομέα διακοπής ρεύματος διασφαλίζουν ότι τα τροποποιημένα δεδομένα στις κρυφές μνήμες της CPU εγγράφονται (flush) σε μόνιμη μνήμη σε περίπτωση καταστροφή συστήματος ή απώλειας ισχύος.

2.2.2 Flushing, Ordering, and Fencing:

Οι επεξεργαστές Intel και AMD υποστηρίζουν οδηγίες συστήματος για μόνιμη μνήμη στο χώρο χρήστη. Η Intel υιοθέτησε το μοντέλο προγραμματισμού NVM SNIA (Storage Networking Industry Association), το οποίο επιτρέπει την άμεση πρόσβαση (DAX) χρησιμοποιώντας λειτουργίες με δυνατότητα διευθυνσιοδότησης byte (load/store). Η διατήρηση των δεδομένων στην προσωρινή μνήμη είναι εγγυημένη μόνο μετά την είσοδο στον τομέα της μη πτητικής μνήμης (persistence domain).

Η αρχιτεκτονική x86 προσφέρει οδηγίες για βελτιστοποιημένο flash γραμμής προσωρινής μνήμης. Παράλληλα με τις υπάρχουσες οδηγίες x86, προστέθηκαν δύο νέες: CLFLUSHOPT και CLWB, οι οποίες πρέπει να ακολουθηθούν από SFENCE για ολοκλήρωση του flash. Το flash γραμμών της cache και η χρήση non-temporal stores υποστηρίζονται στο χώρο χρήστη.

Τα non-temporal stores παρακάμπτουν τις κρυφές μνήμες της CPU, νοουμένου ότι τα γραπτά δεδομένα δεν θα διαβαστούν ξανά σύντομα. Το flush σε μόνιμη μνήμη απευθείας από το χώρο χρήστη είναι εξαιρετικά αποτελεσματικό και τεκμηριώνεται ως βελτιστοποιημένο flash στην προδιαγραφή μοντέλου προγραμματισμού μόνιμης μνήμης SNIA. Είναι σημαντικό για τις εφαρμογές να χρησιμοποιούν βελτιστοποιημένα flushes μόνο όταν το λειτουργικό σύστημα το κρίνει ασφαλές, καθώς το λειτουργικό σύστημα ενδέχεται να απαιτεί σημεία ελέγχου που παρέχονται από κλήσεις όπως msync() για αλλαγές μεταδεδομένων συστήματος αρχείων.

2.2.3 Υποστήριξη λειτουργικού συστήματος για μνήμη και αποθήκευση στο περιβάλλον της μόνιμης μνήμης:

Τα παραδοσιακά συστήματα υπολογιστών περιλαμβάνουν πτητική κύρια μνήμη συνδεδεμένη με την CPU μέσω ενός διαύλου μνήμης, ενώ οι συσκευές αποθήκευσης λειτουργούν σε χαμηλότερες ταχύτητες και συνδέονται μέσω ενός ελεγκτή εισόδου/εξόδου. Σε αυτό το μοντέλο, το λειτουργικό σύστημα διαχειρίζεται την αντιστοίχιση των περιοχών μνήμης στο χώρο διευθύνσεων της εφαρμογής και χειρίζεται την πρόσβαση αποθήκευσης μέσω device driver modules εντός του υποσυστήματος εισόδου/εξόδου.

Οι προγραμματιστές συνήθως εργάζονται με δομές δεδομένων στη μνήμη και χρησιμοποιούν τυποποιημένες κλήσεις συστήματος αρχείων (API) για την αποθήκευση δεδομένων σε συσκευές αποθήκευσης. Το λειτουργικό σύστημα εκτελεί λειτουργίες εισόδου/εξόδου για την εκτέλεση της εγγραφής, οι οποίες είναι γενικά χαμηλότερες από τις ταχύτητες της CPU, προκαλώντας την αναστολή της εφαρμογής μέχρι την ολοκλήρωση εισόδου/εξόδου.

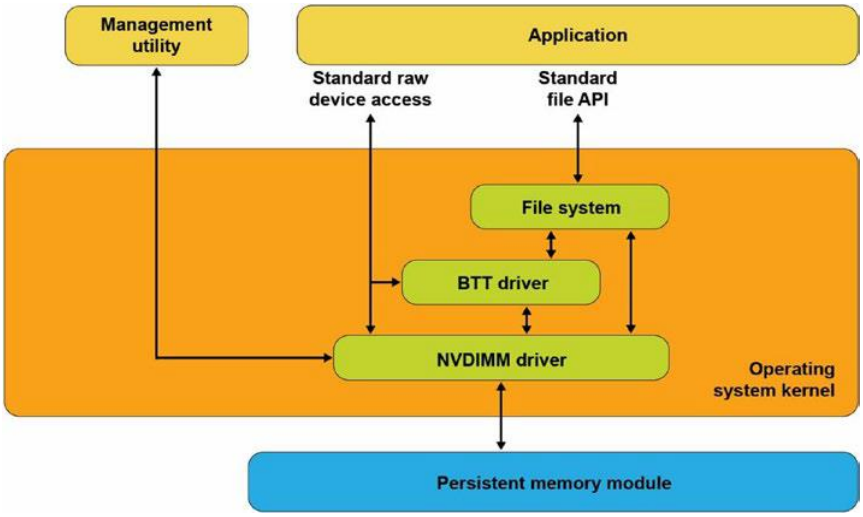

Figure 3 Μόνιμη μνήμη ως αποθήκευση μπλοκ

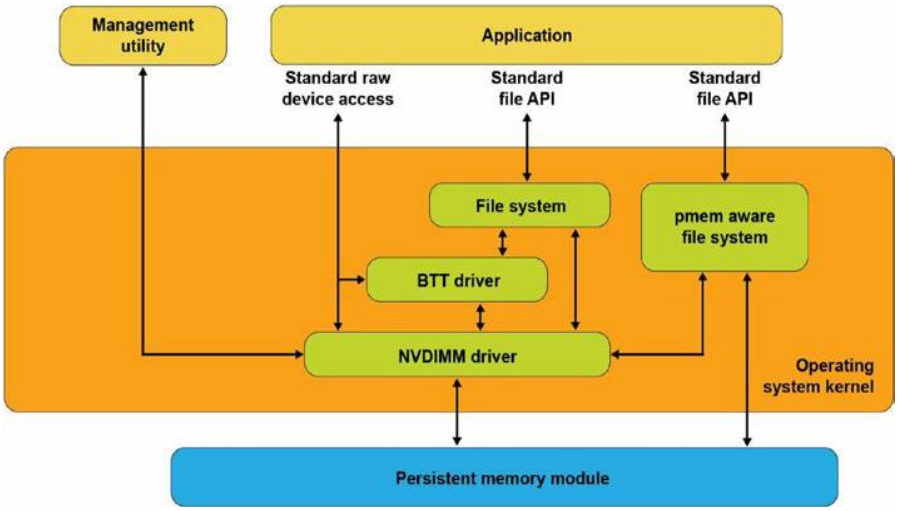

Figure 4 Μόνιμο σύστημα αρχείων με επίγνωση μνήμης

Η μόνιμη (persistent) μνήμη, ωστόσο, επιτρέπει ένα νέο μοντέλο προγραμματισμού που συνδυάζει την απόδοση της μνήμης και της διατήρησης (persistence) των μη πτητικών συσκευών αποθήκευσης. Οι σχεδιαστές λειτουργικών συστημάτων συνεργάστηκαν στο πλαίσιο του SNIA για να ορίσουν ένα κοινό μοντέλο

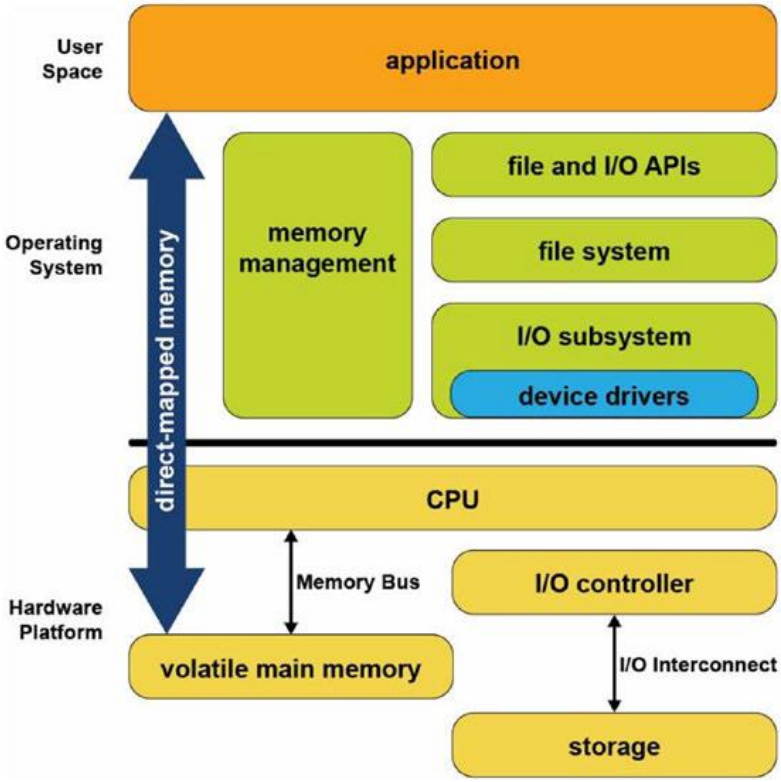

Figure 5 Αποθήκευση και πτητική μνήμη στο λειτουργικό σύστημα

προγραμματισμού για τη χρήση μόνιμης μνήμης, το οποίο υποστηρίζεται τόσο από τα Microsoft Windows όσο και από το Linux. Περισσότερες πληροφορίες μπορείτε να βρείτε στην προδιαγραφή μοντέλου προγραμματισμού SNIA NVM.

Direct Access (DAX) για Persistent μνήμη είναι μια λειτουργία που υπάρχει στα λειτουργικά συστήματα όπως το Linux και τα Windows, η οποία αξιοποιεί την εγγενή ικανότητα της persistent μνήμης να αποθηκεύει πληροφορίες και να λειτουργεί ως μνήμη. Με αυτόν τον τρόπο, το DAX καθιστά παρωχημένη την ανάγκη του λειτουργικού συστήματος να αποθηκεύει προσωρινά αρχεία στην πτητική κύρια μνήμη. Για την υλοποίηση του DAX, ένας διαχειριστής συστήματος δημιουργεί ένα σύστημα αρχείων στη λειτουργική μονάδα μόνιμης μνήμης και το ενσωματώνει στην ιεραρχία συστήματος αρχείων του λειτουργικού συστήματος. Ένα μόνιμο σύστημα αρχείων με επίγνωση μνήμης αναγνωρίζει αρχεία στη μόνιμη μνήμη και προγραμματίζει τη μονάδα διαχείρισης μνήμης (MMU) της CPU να αντιστοιχίσει τη μνήμη απευθείας στο χώρο διευθύνσεων της εφαρμογής. Αυτό εξαλείφει την ανάγκη για ένα αντίγραφο στη μνήμη πυρήνα ή λειτουργίες εισόδου/εξόδου για συγχρονισμό. Οι εφαρμογές μπορούν να χρησιμοποιήσουν δείκτες που επιστρέφονται από mmap() ή MapViewOfFile() για να λειτουργήσουν σε δεδομένα που βρίσκονται στη θέση τους απευθείας στη μόνιμη μνήμη. Αυτό επιτρέπει το χειρισμό μεγάλων αντικειμένων δεδομένων με υψηλότερη και πιο συνεπή απόδοση σε σύγκριση με την αποθήκευση με πρόσβαση εισόδου/εξόδου.

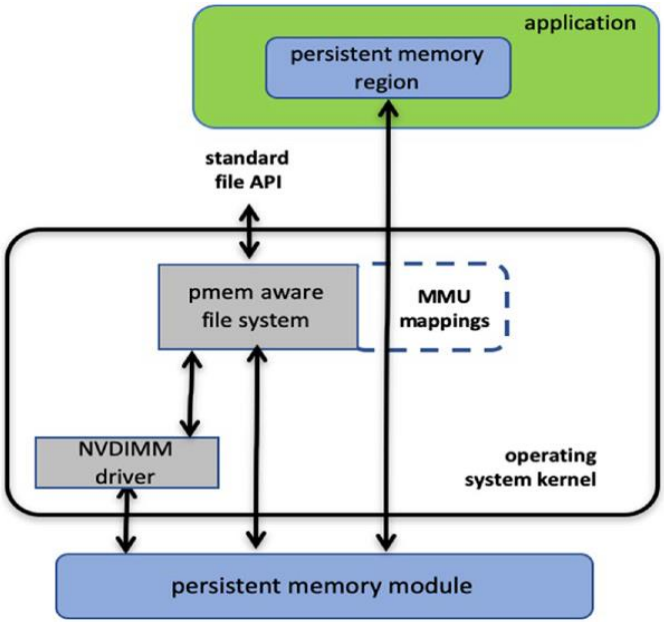

Figure 6 Εισόδοι/εξόδοι άμεσης πρόσβασης (DAX) και τυπικά μονοπάτια εισόδου εξόδου αρχείων μέσω κέρνελ

2.3 Συγκρίσεις

Metric	DRAM	Intel Optane PMem	Intel Optane SSD
Latency	10-20 ns	100-350 ns	~10 μs
Bandwidth	Up to 25.6 GB/s (DDR4)	~8.0 GB/s	Up to 2.4 GB/s (read)
			Up to 2.0 GB/s (write)
Capacity	Up to 32 GB/module	Up to 512 GB/module	Up to 1.5 TB/drive
Persistence	Volatile	Non-volatile	Non-volatile
Form Factor	DIMM	DIMM	U.2, AIC, M.2
Interface	DDR4-compatible	DDR4-compatible	NVMe

Πίνακας 1: Σύγκριση χαρακτηριστικών μνήμης

2.4 Συνέπεια, ατομικότητα δεδομένων και τεχνικό υπόβαθρο

Η διασφάλιση της συνέπειας και της ατομικότητας των δεδομένων στα συστήματα επίμονης μνήμης είναι ζωτικής σημασίας. Η συνέπεια δεδομένων αναφέρεται στην απαίτηση ότι τα δεδομένα παραμένουν σε έγκυρη κατάσταση ακόμη και μετά από βλάβες ή σφάλματα του συστήματος. Η ατομικότητα διασφαλίζει ότι οι λειτουργίες στα δεδομένα είτε ολοκληρώνονται εξ ολοκλήρου είτε καθόλου. Τεχνικές όπως το journaling, το shadow paging και write-ahead logging μπορούν να χρησιμοποιηθούν για τη διατήρηση της συνέπειας και της ατομικότητας των δεδομένων με μόνιμη μνήμη.

2.4.1 Μοντέλα προγραμματισμού και βιβλιοθήκες (NVM Programming Model, PMDK κ.λπ.)

Οι προγραμματιστές μπορούν να αξιοποιήσουν διάφορα μοντέλα προγραμματισμού και βιβλιοθήκες για να εργαστούν αποτελεσματικά με persistent μνήμη. Το μοντέλο προγραμματισμού μη πτητικής μνήμης (NVM) παρέχει οδηγίες για τη δημιουργία λογισμικού που μπορεί να χρησιμοποιήσει απευθείας τη μόνιμη μνήμη. Το Persistent Memory Development Kit (PMDK) είναι μια συλλογή βιβλιοθηκών και εργαλείων που απλοποιεί την ανάπτυξη εφαρμογών χρησιμοποιώντας persistent μνήμη. Άλλες βιβλιοθήκες, όπως το μοντέλο προγραμματισμού NVM της SNIA (Storage Networking Industry Association), παρέχουν επίσης υποστήριξη για persistent προγραμματισμό μνήμης. Σε αυτή τη διπλωματική εργασία χρησιμοποιούμε το PMDK για να δημιουργήσουμε ένα key-value store και τον MPK μηχανισμό για να βεβαιωθούμε ότι τα κλειδιά προστατεύονται.

Υπόβαθρο του κιτ ανάπτυξης μόνιμης μνήμης (PMDK).

Το Persistent Memory Development Kit (PMDK) είναι μια εκτενής συλλογή βιβλιοθηκών και εργαλείων ανοιχτού κώδικα που στοχεύουν να βοηθήσουν τους προγραμματιστές εφαρμογών και τους διαχειριστές συστημάτων να διαχειρίζονται και να έχουν πρόσβαση σε συσκευές μόνιμης μνήμης πιο αποτελεσματικά. Το PMDK έχει αναπτυχθεί σε συνδυασμό με την εξελισσόμενη υποστήριξη για persistent μνήμη σε λειτουργικά συστήματα, διασφαλίζοντας ότι οι βιβλιοθήκες αξιοποιούν πλήρως τα χαρακτηριστικά που προσφέρονται μέσω αυτών των διεπαφών.

Παρόλο που η Intel δημιούργησε το PMDK για να υποστηρίξει τα προϊόντα υλικού της, η εταιρεία είναι αφοσιωμένη στο να διασφαλίσει ότι οι βιβλιοθήκες και τα εργαλεία παραμένουν ανεξάρτητα από τον προμηθευτή και ουδέτερα ως προς την πλατφόρμα. Αυτό σημαίνει ότι το PMDK μπορεί να προσαρμοστεί ώστε να λειτουργεί με

οποιαδήποτε πλατφόρμα που εκθέτει τις απαραίτητες διεπαφές μέσω του λειτουργικού συστήματος, συμπεριλαμβανομένων των Linux και Microsoft Windows. Η Intel ενθαρρύνει τις συνεισφορές από ιδιώτες, προμηθευτές υλικού και ανεξάρτητους προμηθευτές λογισμικού (ISV).

Το PMDK είναι διαθέσιμο ως ανοικτό λογισμικό στο GitHub (https://github.com/pmem/pmdk) και διαθέτει ειδικό ιστότοπο στο https://pmem.io.

Επιλογή της κατάλληλης σημασιολογίας

Με το πλήθος των βιβλιοθηκών που είναι διαθέσιμες στο PMDK, είναι σημαντικό να αξιολογούνται προσεκτικά οι επιλογές.

Το PMDK παρέχει δύο κατηγορίες βιβλιοθηκών:

• Οι πτητικές βιβλιοθήκες (volatile) εξυπηρετούν περιπτώσεις χρήσης που εστιάζουν στην αξιοποίηση της χωρητικότητας της επίμονης μνήμης.

• Οι μόνιμες (persistent) βιβλιοθήκες προορίζονται για λογισμικό που στοχεύει στην υλοποίηση fail-safe αλγορίθμων μόνιμης μνήμης.

Οι προκλήσεις που παρουσιάζουν τα ασφαλή persistent προγράμματα διαφέρουν σημαντικά από τα πτητικά. Στην παρούσα διπλωματική εργασία επιλέξαμε την προσέγγιση της fail-safe persistence.

Πτητικές βιβλιοθήκες

Οι πτητικές βιβλιοθήκες προσφέρουν μια απλούστερη εμπειρία χρήστη, καθώς μπορούν να καταφύγουν στην δυναμική μνήμη τυχαίας προσπέλασης (DRAM) όταν η μόνιμη μνήμη δεν είναι διαθέσιμη, με αποτέλεσμα μια πιο άμεση υλοποίηση. Μπορεί

επίσης να παρουσιάζουν χαμηλότερα συνολικά γενικά έξοδα σε σύγκριση με παρόμοιες μόνιμες βιβλιοθήκες, ανάλογα με τον φόρτο εργασίας, καθώς δεν χρειάζεται να εγγυώνται τη συνέπεια των δεδομένων ενόψει αποτυχιών. Καθώς δεν θα χρησιμοποιήσουμε αυτές τις βιβλιοθήκες, δεν θα συζητηθούν περαιτέρω.

Μη πτητικές βιβλιοθήκες

Οι μη πτητικές βιβλιοθήκες έχουν σχεδιαστεί για να διασφαλίζουν τη συνέπεια των δεδομένων με ασφάλεια έναντι βλάβης για εφαρμογές που απαιτούν διατήρηση δεδομένων σε κύκλους ισχύος ή σφάλματα συστήματος. Αυτές οι βιβλιοθήκες συνήθως συνεπάγονται πρόσθετα γενικά έξοδα σε σύγκριση με τις ευμετάβλητες βιβλιοθήκες λόγω της εστίασής τους στη διατήρηση της συνέπειας των δεδομένων παρά τις αποτυχίες.

Παρακάτω, εξετάζουμε τις βιβλιοθήκες που είναι κατάλληλες για περιπτώσεις μόνιμης χρήσης σε εφαρμογές, συζητώντας το σκοπό κάθε βιβλιοθήκης και τα κατάλληλα σενάρια χρήσης. Ορισμένες βιβλιοθήκες ενδέχεται να έχουν επικαλυπτόμενες περιπτώσεις χρήσης.

libpmem

Η βιβλιοθήκη libpmem παρέχει υποστήριξη μόνιμης μνήμης χαμηλού επιπέδου, συμπεριλαμβανομένων των εισόδου/εξόδων αρχείων που έχουν αντιστοιχιστεί στη μνήμη, flush της προσωρινής μνήμης και των persistent -aware φρακτών. Είναι το θεμέλιο για άλλες βιβλιοθήκες persistent μνήμης στο PMDK.

libpmemobj

Η βιβλιοθήκη libpmemobj είναι ένας χώρος αποθήκευσης αντικειμένων συναλλαγών που παρέχει εκχώρηση μνήμης, συναλλαγές και δομές δεδομένων για μόνιμη μνήμη.

Είναι χτισμένο πάνω από το libpmem και προσφέρει υψηλότερο επίπεδο αφαίρεσης για προγραμματιστές. Αυτή η βιβλιοθήκη είναι κατάλληλη για τις περισσότερες εφαρμογές που απαιτούν εγγυήσεις επιμονής και συνέπειας, προσφέροντας έναν απλό και αποτελεσματικό τρόπο ανάπτυξης λογισμικού με επίγνωση της επίμονης μνήμης.

libpmemblk

Η βιβλιοθήκη libpmemblk παρέχει μια μόνιμη διασύνδεση αποθήκευσης persistent aware μπλοκ. Έχει σχεδιαστεί για περιπτώσεις χρήσης που απαιτούν ένα απλό σύστημα αποθήκευσης μπλοκ σταθερού μεγέθους. Είναι ιδανικό για εφαρμογές που δεν χρειάζονται την πολυπλοκότητα ενός πλήρους συστήματος αρχείων, αλλά εξακολουθούν να απαιτούν μια ελαφριά και αποτελεσματική λύση αποθήκευσης μπλοκ.

libpmemlog

Η βιβλιοθήκη libpmemlog παρέχει μια persistent aware μόνιμη διασύνδεση αποθήκευσης αρχείων καταγραφής. Έχει σχεδιαστεί για περιπτώσεις χρήσης που απαιτούν την προσάρτηση δεδομένων σε ένα αρχείο καταγραφής με ασφαλή τρόπο. Είναι ιδανικό για εφαρμογές που πρέπει να διατηρούν αρχείο καταγραφής συμβάντων ή συναλλαγών με εγγυήσεις επιμονής και συνέπειας.

2.4.2 Περιπτώσεις χρήσης σε βάσεις δεδομένων, HPC, ανάλυση μαζικών δεδομένων, AI και IoT

Η μόνιμη μνήμη έχει ένα ευρύ φάσμα περιπτώσεων χρήσης σε διάφορους τομείς:

Βάσεις δεδομένων: Η μόνιμη μνήμη μπορεί να βελτιώσει τις ταχύτητες επεξεργασίας συναλλαγών, να μειώσει τους χρόνους ανάκτησης μετά από διακοπές ρεύματος και να απλοποιήσει τη διαχείριση βάσεων δεδομένων.

Υπολογιστική υψηλών επιδόσεων (HPC): Η υπολογιστική στη μνήμη μπορεί να ενισχυθεί με μόνιμη μνήμη, επιτρέποντας ταχύτερη επεξεργασία μεγάλων συνόλων δεδομένων και πολύπλοκων προσομοιώσεων.

Big Data Analytics: Ο χαμηλός λανθάνων χρόνος και η υψηλή χωρητικότητα της επίμονης μνήμης την καθιστούν ιδανική για το χειρισμό μεγάλων συνόλων δεδομένων σε πραγματικό χρόνο, βελτιώνοντας την αποτελεσματικότητα των ροών εργασίας των analytics.

Τεχνητή νοημοσύνη (AI): Η επίμονη μνήμη μπορεί να επιταχύνει την εκπαίδευση μοντέλων μηχανικής μάθησης και την εξαγωγή συμπερασμάτων, παρέχοντας ταχύτερη πρόσβαση σε μεγάλα σύνολα δεδομένων.

Internet of Things (IoT): Οι συσκευές IoT μπορούν να επωφεληθούν από την ενεργειακή απόδοση και τη χαμηλή καθυστέρηση της μόνιμης μνήμης, επιτρέποντας την επεξεργασία και ανάλυση δεδομένων σε πραγματικό χρόνο στα άκρα.

2.5 Μελλοντικές τάσεις, προκλήσεις και συμπέρασμα

2.5.1 Ζητήματα επεκτασιμότητας, κατασκευής, ασφάλειας και απορρήτου

Καθώς η τεχνολογία της επίμονης μνήμης εξελίσσεται, πρέπει να αντιμετωπιστούν αρκετές προκλήσεις. Η επεκτασιμότητα και η δυνατότητα κατασκευής είναι ζωτικής σημασίας για να διασφαλιστεί ότι αυτές οι τεχνολογίες μπορούν να παραχθούν οικονομικά αποδοτικά και σε μεγάλες ποσότητες. Ανησυχίες σχετικά με την ασφάλεια και την προστασία της ιδιωτικής ζωής προκύπτουν επίσης λόγω της επίμονης φύσης των δεδομένων, η οποία απαιτεί ισχυρούς μηχανισμούς κρυπτογράφησης και ελέγχου πρόσβασης για την προστασία ευαίσθητων πληροφοριών.

2.5.2 Ανακεφαλαίωση της τεχνολογίας μόνιμης μνήμης και μελλοντικές βλέψεις

Συμπερασματικά, η τεχνολογία επίμονης μνήμης είναι ένα αναδυόμενο πεδίο που συνδυάζει τα οφέλη των πτητικών και μη πτητικών τύπων μνήμης. Προσφέροντας υψηλή απόδοση, χαμηλή καθυστέρηση και ανθεκτικότητα δεδομένων, έχει τη δυνατότητα να μεταμορφώσει διάφορες εφαρμογές, συμπεριλαμβανομένων των βάσεων δεδομένων, της υπολογιστικής υψηλών επιδόσεων, της ανάλυσης μεγάλων δεδομένων, της τεχνητής νοημοσύνης και του Διαδικτύου των πραγμάτων. Καθώς η τεχνολογία ωριμάζει και ξεπερνά τις προκλήσεις που σχετίζονται με την επεκτασιμότητα, τη δυνατότητα κατασκευής, την ασφάλεια και την προστασία της ιδιωτικής ζωής, αναμένεται να έχει σημαντικό αντίκτυπο στο μέλλον των υπολογιστικών συστημάτων.

Κεφάλαιο 3

Τεχνολογίες προστασίας μνήμης

3.1 Morello, η αρχιτεκτονική CPU ARMv8-A και το μοντέλο δυνατοτήτων CHERI

Επιδιώκοντας την ενίσχυση της προστασίας της μνήμης και την αντιμετώπιση των σύγχρονων προκλήσεων ασφάλειας, οι ερευνητές και οι μηχανικοί διερευνούν συνεχώς νέες τεχνολογίες και προσεγγίσεις. Μια τέτοια τεχνολογία είναι η CPU Morello, ένας επεξεργαστής βασισμένος στο ARMv8-A που συνδυάζει το μοντέλο δυνατοτήτων CHERI. Παρακάτω θα συζητηθεί μια εις βάθος περίληψη της τεχνολογίας Morello, των μηχανισμών της, του κόστους απόδοσης, των πτυχών ασφάλειας και των πιθανών τρωτών σημείων, όλα στο πλαίσιο του μοντέλου δυνατοτήτων CHERI.

3.1.1 Morello και η αρχιτεκτονική CPU ARMv8-A:

3.1.1.1 Έργο Morello:

Το Morello Project είναι ένα συνεργατικό ερευνητικό πρόγραμμα μεταξύ του ARM, του Πανεπιστημίου του Cambridge και άλλων εταίρων της βιομηχανίας. Το έργο στοχεύει στη δημιουργία μιας πρωτότυπης πλατφόρμας υλικού-λογισμικού για τη διερεύνηση και αξιολόγηση των πλεονεκτημάτων ασφάλειας από την ενσωμάτωση του μοντέλου δυνατοτήτων CHERI στην αρχιτεκτονική ARMv8-A. Η τεχνολογία Morello έχει τη

δυνατότητα να μεταμορφώσει την προστασία μνήμης και την ασφάλεια του συστήματος σε διάφορες εφαρμογές, συμπεριλαμβανομένων των ενσωματωμένων συστημάτων, των κινητών συσκευών και των διακομιστών.

3.1.1.2 Αρχιτεκτονική ARMv8-A:

Το ARMv8-A είναι η τελευταία γενιά της αρχιτεκτονικής επεξεργαστή 64-bit της ARM, σχεδιασμένη να παρέχει υψηλή απόδοση, ενεργειακή απόδοση και βελτιωμένα χαρακτηριστικά ασφαλείας. Η αρχιτεκτονική εισάγει αρκετές νέες βελτιώσεις ασφαλείας, συμπεριλαμβανομένου του ελέγχου ταυτότητας δείκτη (pointer) και των επεκτάσεων ετικετών μνήμης, επιπρόσθετα από τις επεκτάσεις Morello που ενσωματώνουν τις δυνατότητες CHERI.

3.1.2 Το μοντέλο δυνατοτήτων CHERI (capabilities):

Το μοντέλο δυνατοτήτων CHERI (Capability Hardware Enhanced RISC Instructions) είναι μια προσέγγιση υλικού-λογισμικού που αναπτύχθηκε από ερευνητές του Πανεπιστημίου του Cambridge. Στόχος του είναι να παρέχει λεπτομερή, αποτελεσματική προστασία μνήμης και κλιμακούμενη διαμερισματοποίηση (compartmentalization), διατηρώντας παράλληλα τη συμβατότητα με το υπάρχον λογισμικό.

3.1.3 Δυνατότητες CHERI:

Οι δυνατότητες CHERI είναι unforgeable tokens που αντιπροσωπεύουν την εξουσία πρόσβασης σε μια συγκεκριμένη περιοχή μνήμης. Αυτά τα tokens περιλαμβάνουν μεταδεδομένα, όπως η βασική διεύθυνση, το μήκος και τα δικαιώματα, τα οποία αποθηκεύονται μαζί με τις παραδοσιακές διευθύνσεις μνήμης στους καταχωρητές του επεξεργαστή. Επιβάλλοντας τη χρήση δυνατοτήτων για πρόσβαση στη μνήμη, η CHERI διασφαλίζει ότι μόνο εξουσιοδοτημένες διεργασίες μπορούν να έχουν πρόσβαση σε συγκεκριμένες περιοχές μνήμης, αποτρέποντας έτσι τη μη εξουσιοδοτημένη πρόσβαση, αλλοίωση ή καταστροφή δεδομένων.

3.1.4 CHERI και Morello:

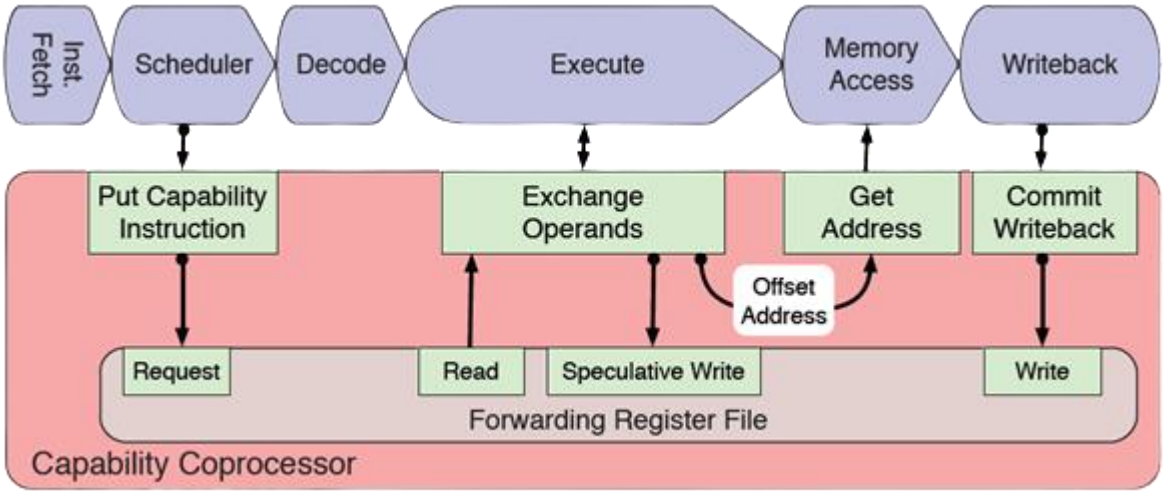

Figure 7 Beri pipeline με capability coprocessor

Ο επεξεργαστής Morello ενσωματώνει το μοντέλο δυνατοτήτων CHERI, επεκτείνοντας την αρχιτεκτονική ARMv8-A με νέες εντολές και καταχωρητές για την υποστήριξη προστασίας μνήμης βάσει δυνατοτήτων. Με την ενσωμάτωση των δυνατοτήτων CHERI στην αρχιτεκτονική ARMv8-A, το Morello μπορεί να παρέχει ισχυρότερες εγγυήσεις ασφαλείας και να προστατεύσει από ένα ευρύ φάσμα επιθέσεων που βασίζονται σε λογισμικό, όπως υπερχείλιση buffer, έγχυση κώδικα και ROP (return-oriented programming).

3.1.5 Γενικό κόστος απόδοσης και δοκιμές:

3.1.5.1 Κόστος απόδοσης:

Ενώ το μοντέλο δυνατοτήτων CHERI και οι επεκτάσεις Morello στοχεύουν στην παροχή βελτιωμένης ασφάλειας, εισάγουν επίσης κάποιο κόστος απόδοσης λόγω της πρόσθετης διαχείρισης μεταδεδομένων και ελέγχων των capabilities κατά τη διάρκεια προσβάσεων μνήμης. Ωστόσο, το γενικό κόστος είναι γενικά μέτριο και μπορούν να μετριαστεί μέσω διαφόρων τεχνικών βελτιστοποίησης, όπως συμπίεση δυνατοτήτων και αποτελεσματικές στρατηγικές προσωρινής αποθήκευσης.

3.1.5.2 Δοκιμές απόδοσης:

Για να αξιολογήσουν τον αντίκτυπο στην απόδοση των CHERI και Morello, οι ερευνητές διεξήγαγαν διάφορες δοκιμές απόδοσης χρησιμοποιώντας microbenchmarks, benchmarks εφαρμογών και στοίβες λογισμικού. Τα αποτελέσματα δείχνουν ότι τα γενικά έξοδα απόδοσης που εισάγονται από τις δυνατότητες CHERI είναι γενικά αποδεκτά, με τους περισσότερους φόρτους εργασίας να αντιμετωπίζουν μόνο μέτριο αντίκτυπο στην απόδοση. Σε ορισμένες περιπτώσεις, τα γενικά έξοδα απόδοσης μπορούν ακόμη και να αντισταθμιστούν από τα οφέλη ασφαλείας που παρέχονται από το μοντέλο δυνατοτήτων CHERI.

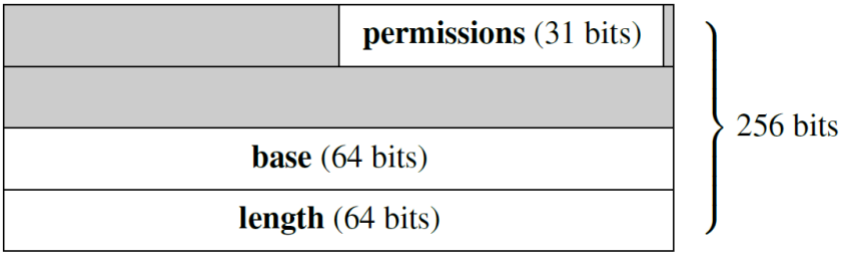

Figure 8Capabilities μνήμης

3.1.6 Ασφάλεια και ευπάθειες:

3.1.6.1 Οφέλη ασφαλείας:

Η ενσωμάτωση του μοντέλου δυνατοτήτων CHERI στον επεξεργαστή Morello προσφέρει σημαντικά οφέλη ασφάλειας, όπως:

- **Λεπτομερής (fine-grained) προστασία μνήμης:** Οι δυνατότητες CHERI παρέχουν ακριβή έλεγχο της πρόσβασης στη μνήμη, διασφαλίζοντας ότι μόνο εξουσιοδοτημένες διεργασίες μπορούν να έχουν πρόσβαση ή να τροποποιούν ευαίσθητα δεδομένα που είναι αποθηκευμένα στη μόνιμη μνήμη.
- **Κλιμακούμενη διαμερισματοποίηση:** Το CHERI επιτρέπει την αποτελεσματική διαμερισματοποίηση της μνήμης, επιτρέποντας την απομόνωση μεμονωμένων τμημάτων λογισμικού και μειώνοντας την περιοχή για επίθεση.
- **Μετριασμός κοινών επιθέσεων λογισμικού:** Με την επιβολή της χρήσης δυνατοτήτων για πρόσβαση στη μνήμη, οι CHERI και Morello μπορούν να προστατεύσουν από ένα ευρύ φάσμα επιθέσεων που βασίζονται σε λογισμικό, όπως υπερχείλιση buffer (buffer overflow)s, έγχυση κώδικα (code injection) και προγραμματισμό προσανατολισμένο στην επιστροφή (ROP).
- **Συμβατότητα με υπάρχον λογισμικό:** Το μοντέλο δυνατοτήτων CHERI διατηρεί τη συμβατότητα με τις υπάρχουσες στοίβες λογισμικού, επιτρέποντας την ομαλότερη μετάβαση στα βελτιωμένα χαρακτηριστικά ασφαλείας που παρέχονται από τις επεκτάσεις Morello και ARMv8-A.

3.1.6.2 Πιθανές ευπάθειες:

Ενώ το μοντέλο δυνατοτήτων CHERI και οι επεκτάσεις του Morello προσφέρουν σημαντικά οφέλη ασφαλείας, μπορεί επίσης να έχουν κάποιες πιθανές ευπάθειες ή περιορισμούς:

1. **Ευπάθειες σε επίπεδο υλικού:** Όπως συμβαίνει με οποιονδήποτε μηχανισμό ασφαλείας που βασίζεται σε υλικό, ενδέχεται να υπάρχουν ευπάθειες στην υλοποίηση του μοντέλου δυνατοτήτων CHERI ή των επεκτάσεων Morello που θα μπορούσαν να αξιοποιηθούν από επιτιθέμενους.
2. **Συνύπαρξη με άλλους μηχανισμούς ασφαλείας:** Η ενσωμάτωση των CHERI και Morello με άλλους μηχανισμούς ασφαλείας, όπως η κρυπτογράφηση ή οι λίστες ελέγχου πρόσβασης, μπορεί να εισαγάγει πολυπλοκότητες και πιθανές συγκρούσεις που απαιτούν προσεκτική εξέταση και σχεδιασμό.
3. **Διαρροή δυνατότητας:** Εάν ένας εισβολέας μπορεί να αποκτήσει ένα token δυνατότητας μέσω ευπαθειών λογισμικού ή εσφαλμένων ρυθμίσεων παραμέτρων, ενδέχεται να είναι σε θέση να παρακάμψει την προβλεπόμενη προστασία μνήμης.

3.1.7 Συμπέρασμα:

Η τεχνολογία Morello, βασισμένη στην αρχιτεκτονική CPU ARMv8-A και ενσωματώνοντας το μοντέλο δυνατοτήτων CHERI, προσφέρει πολλά υποσχόμενες εξελίξεις στην προστασία της μνήμης και την ασφάλεια του συστήματος. Παρέχοντας λεπτομερή έλεγχο πρόσβασης, κλιμακούμενη διαμερισματοποίηση και μετριασμό κοινών επιθέσεων λογισμικού, οι Morello και CHERI μπορούν να ενισχύσουν σημαντικά το κομμάτι ασφάλειας σε διάφορες εφαρμογές. Ωστόσο, είναι σημαντικό να αντιμετωπιστούν πιθανές ευπάθειες, ζητήματα σε επίπεδο υλικού και προκλήσεις ενοποίησης με άλλους μηχανισμούς ασφαλείας για να διασφαλιστεί η επιτυχής εφαρμογή και υιοθέτηση αυτής της τεχνολογίας. Μέσω της συνεχιζόμενης έρευνας, ανάπτυξης και συνεργασίας, το έργο Morello στοχεύει να μεταμορφώσει το μέλλον της προστασίας της μνήμης και της ασφάλειας στον συνεχώς εξελισσόμενο κόσμο της πληροφορικής.

3.2 Κλειδιά προστασίας μνήμης

Τα κλειδιά προστασίας μνήμης Intel (MPK) είναι μια τεχνολογία που βασίζεται σε υλικό και έχει σχεδιαστεί για να βελτιώνει την προστασία της μνήμης και τον έλεγχο πρόσβασης σε εφαρμογές χώρου χρήστη. Το MPK παρέχει έναν αποτελεσματικό και λεπτομερή μηχανισμό για τη διαχείριση των δικαιωμάτων πρόσβασης στη μνήμη, ο οποίος μπορεί να συμβάλει στη βελτίωση της ασφάλειας και της αξιοπιστίας των εφαρμογών χώρου χρήστη χωρίς να εισάγει σημαντικά γενικά κόστη απόδοσης.

3.2.1 Υπόβαθρο

Οι παραδοσιακοί μηχανισμοί προστασίας μνήμης, όπως οι πίνακες σελίδων και οι λίστες ελέγχου πρόσβασης (ACL-Access Control Lists), μπορεί να είναι περίπλοκοι και αργοί λόγω της εμπλοκής λειτουργιών πυρήνα του λειτουργικού συστήματος. Ο Intel MPK μηχανισμός προσφέρει μια πιο αποτελεσματική και ελαφριά εναλλακτική λύση, επιτρέποντας στις εφαρμογές χώρου χρήστη να διαχειρίζονται απευθείας τα δικαιώματα πρόσβασης στη μνήμη χωρίς την ανάγκη παρέμβασης πυρήνα.

3.2.2 Πώς λειτουργεί το Intel MPK

Ο Intel MPK μηχανισμός εισάγει ένα σύνολο νέων οδηγιών και αρχιτεκτονικών χαρακτηριστικών που επιτρέπουν τον λεπτομερή έλεγχο πρόσβασης μνήμης σε εφαρμογές χώρου χρήστη. Τα βασικά στοιχεία του Intel MPK είναι:

3.2.2.1 Κλειδιά προστασίας:

Το MPK εκχωρεί ένα κλειδί προστασίας (PKey) σε κάθε περιοχή μνήμης, το οποίο χρησιμεύει ως αναγνωριστικό για τα δικαιώματα πρόσβασης που σχετίζονται με αυτήν την περιοχή. Οι τιμές PKey μπορούν να χρησιμοποιηθούν για τον έλεγχο της πρόσβασης ανάγνωσης και εγγραφής στις περιοχές μνήμης με τις οποίες σχετίζονται.

3.2.2.2 PKey εντολές:

Κάθε διεργασία έχει ένα μητρώο PKey (PKRU), το οποίο αποθηκεύει τα τρέχοντα δικαιώματα πρόσβασης για κάθε PKey. Το PKRU είναι ένας καταχωρητής ανά νήμα,

επιτρέποντας σε διαφορετικά νήματα μέσα σε μια διεργασία να έχουν διαφορετικά δικαιώματα πρόσβασης στις περιοχές μνήμης.

3.2.2.3 Εντολές MPK:

Ο Intel MPK μηχανισμός παρέχει ένα σύνολο νέων εντολών για τη διαχείριση κλειδιών προστασίας και δικαιωμάτων πρόσβασης στη μνήμη. Αυτές οι εντολές περιλαμβάνουν:

WRPKRU: Γράψε το μητρώο PKey με τα καθορισμένα δικαιώματα πρόσβασης.

RDPKRU: Διάβασε τα τρέχοντα δικαιώματα πρόσβασης από το μητρώο PKey.

PKEY_SET: Όρισε το κλειδί προστασίας για μια περιοχή μνήμης.

PKEY_FREE: Ελευθέρωσε ένα κλειδί προστασίας που είχε εκχωρηθεί προηγουμένως.

Χρησιμοποιώντας αυτές τις οδηγίες, οι εφαρμογές χώρου χρήστη μπορούν να διαχειριστούν κλειδιά προστασίας μνήμης και δικαιώματα πρόσβασης χωρίς την ανάγκη κλήσεων συστήματος ή την παρέμβαση του πυρήνα.

3.2.3 Γενικά κόστη απόδοσης και δοκιμές

Ένα από τα βασικά πλεονεκτήματα του Intel MPK είναι η ελάχιστη επιβάρυνση απόδοσης σε σύγκριση με τους παραδοσιακούς μηχανισμούς προστασίας μνήμης. Το MPK επιτρέπει στις εφαρμογές χώρου χρήστη να διαχειρίζονται απευθείας τα δικαιώματα πρόσβασης στη μνήμη, αποφεύγοντας την επιβάρυνση της εμπλοκής του πυρήνα και των κλήσεων συστήματος. Επιπλέον, η ανά νήμα λειτουργία του καταχωρητή PKRU επιτρέπει τον αποτελεσματικό χειρισμό των δικαιωμάτων πρόσβασης μνήμης για εφαρμογές πολλαπλών νημάτων.

Αρκετές δοκιμές απόδοσης και σημεία αναφοράς έχουν αποδείξει την αποτελεσματικότητα του Intel MPK. Σε ένα επόμενο κεφάλαιο θα παρουσιαστεί ένα πλήρες πείραμα για τη δοκιμή του MPK για τη δοκιμή του Intel MPK με την παραδοσιακή προστασία μνήμης που βασίζεται σε πίνακα σελίδων. Άλλες μελέτες έχουν αναφέρει παρόμοια αποτελέσματα, με το MPK να ξεπερνά τους παραδοσιακούς μηχανισμούς όσον αφορά τόσο την καθυστέρηση όσο και την απόδοση.

3.2.4 Ασφάλεια και ευπάθειες

Ο Intel MPK μηχανισμός παρέχει ένα πρόσθετο επίπεδο ασφάλειας για εφαρμογές χώρου χρήστη, επιτρέποντας λεπτομερή (fine grained) έλεγχο πρόσβασης μνήμης. Αυτό μπορεί να βοηθήσει στην προστασία από διάφορες απειλές ασφαλείας, όπως υπερχειλίσεις buffer, ευπάθειες use-after-free και μη εξουσιοδοτημένη πρόσβαση στη μνήμη.

Ωστόσο, το Intel MPK δεν υπάρχει χωρίς τους δικούς του περιορισμούς και δικές του πιθανές ευπάθειες:

- **Ευπάθειες σε επίπεδο υλικού:** Ως τεχνολογία που βασίζεται σε υλικό, το Intel MPK υπόκειται σε πιθανές ευπάθειες στην υποκείμενη υλοποίηση υλικού. Τυχόν ελαττώματα στο σχεδιασμό ή την κατασκευή επεξεργαστών Intel θα μπορούσαν ενδεχομένως να επηρεάσουν την αποτελεσματικότητα του MPK.
- **Περιορισμοί στον βαθμό λεπτομέρειας (granularity):** Ο MPK παρέχει λεπτομερή έλεγχο πρόσβασης μνήμης, αλλά περιορίζεται από τον αριθμό των διαθέσιμων κλειδιών προστασίας. Σε σενάρια όπου μια εφαρμογή απαιτεί μεγάλο αριθμό διακριτών περιοχών μνήμης με διαφορετικά δικαιώματα πρόσβασης, ο βαθμός λεπτομέρειας του MPK ενδέχεται να είναι ανεπαρκής.
- **Συμβατότητα με άλλους μηχανισμούς ασφαλείας:** Η ενσωμάτωση του Intel MPK με άλλους μηχανισμούς ασφαλείας, όπως κρυπτογράφηση ή πρόσθετα μέτρα ελέγχου πρόσβασης, ενδέχεται να δημιουργήσει πολυπλοκότητες και πιθανές διενέξεις που απαιτούν προσεκτική εξέταση και σχεδιασμό.

3.2.5 Περιπτώσεις χρήσης και εφαρμογές του Intel MPK

Ο Intel MPK μηχανισμός μπορεί να χρησιμοποιηθεί σε διάφορες περιπτώσεις χρήσης και εφαρμογές για την ενίσχυση της προστασίας της μνήμης και του ελέγχου πρόσβασης στο χώρο χρήστη:

- **Απομόνωση ευαίσθητων δεδομένων:** Ο MPK μπορεί να χρησιμοποιηθεί για την απομόνωση ευαίσθητων δεδομένων σε μια διαδικασία, αποτρέποντας τη μη εξουσιοδοτημένη πρόσβαση ή την παραβίαση από άλλα μέρη της εφαρμογής.
- **Θωράκιση βιβλιοθηκών κρίσιμων για την ασφάλεια:** Με την εφαρμογή MPK σε βιβλιοθήκες κρίσιμες για την ασφάλεια, οι προγραμματιστές μπορούν να διασφαλίσουν ότι οι περιοχές μνήμης τους προστατεύονται από μη εξουσιοδοτημένη πρόσβαση, μειώνοντας τον κίνδυνο ευπαθειών.
- **Προστασία από αλλοίωση μνήμης:** Ο MPK μπορεί να βοηθήσει στην προστασία από επιθέσεις καταστροφής μνήμης, όπως υπερχειλίσεις buffer, περιορίζοντας την πρόσβαση εγγραφής σε συγκεκριμένες περιοχές μνήμης.
- **Μετριασμός ευπαθειών Use-After-Free:** Με τη διαχείριση των δικαιωμάτων πρόσβασης στη μνήμη με το MPK, οι προγραμματιστές μπορούν να αποτρέψουν την εκμετάλλευση των ευπαθειών use-after-free, οι οποίες προκύπτουν όταν ένα πρόγραμμα συνεχίζει να χρησιμοποιεί μια περιοχή μνήμης μετά την αποδέσμευσή του.

3.2.6 Προκλήσεις και μελλοντικές κατευθύνσεις

Παρά τα πλεονεκτήματα του Intel MPK, πρέπει να αντιμετωπιστούν αρκετές προκλήσεις και μελλοντικές κατευθύνσεις για την περαιτέρω βελτίωση της αποτελεσματικότητας και της υιοθέτησής του:

- **Επεκτασιμότητα και βαθμός λεπτομέρεια:** Καθώς ο αριθμός των κλειδιών προστασίας είναι περιορισμένος, η εξεύρεση τρόπων επέκτασης της επεκτασιμότητας και της λεπτομέρειας του Intel MPK θα είναι απαραίτητη για το χειρισμό πιο περίπλοκων σεναρίων και διατάξεων μνήμης.
- **Ενσωμάτωση με άλλους μηχανισμούς ασφαλείας:** Η ανάπτυξη μεθόδων για την ενσωμάτωση του Intel MPK με άλλους μηχανισμούς ασφαλείας, όπως η κρυπτογράφηση και ο έλεγχος πρόσβασης, θα είναι ζωτικής σημασίας για τη δημιουργία ολοκληρωμένων λύσεων προστασίας μνήμης.

- **Αντιμετώπιση τρωτών σημείων σε επίπεδο υλικού:** Καθώς τα τρωτά σημεία σε επίπεδο υλικού μπορούν δυνητικά να επηρεάσουν την αποτελεσματικότητα του MPK, απαιτούνται συνεχείς προσπάθειες έρευνας και ανάπτυξης για την αντιμετώπιση αυτών των προκλήσεων και τη διασφάλιση της ευρωστίας της τεχνολογίας.

3.2.7 Συμπέρασμα

Τα κλειδιά προστασίας μνήμης Intel (MPK) είναι μια υποσχόμενη τεχνολογία που παρέχει αποτελεσματικό και λεπτομερή έλεγχο πρόσβασης μνήμης για εφαρμογές χώρου χρήστη. Αξιοποιώντας μηχανισμούς που βασίζονται σε υλικό, το MPK μπορεί να ενισχύσει την ασφάλεια και την αξιοπιστία των εφαρμογών χωρίς να εισάγει σημαντικά γενικά κόστη απόδοσης. Παρόλο που υπάρχουν ορισμένοι περιορισμοί και πιθανές ευπάθειες που σχετίζονται με το Intel MPK, αντιπροσωπεύει ένα σημαντικό βήμα προς τα μπροστά στην προστασία της μνήμης και τον έλεγχο πρόσβασης, με πολλές πιθανές περιπτώσεις χρήσης και εφαρμογές. Οι μελλοντικές προσπάθειες έρευνας και ανάπτυξης θα είναι ζωτικής σημασίας για την αντιμετώπιση των υφιστάμενων προκλήσεων και την περαιτέρω βελτίωση των δυνατοτήτων του Intel MPK.

3.3 Hodor - Απομόνωση εντός της διεργασίας για βιβλιοθήκες επιπέδου δεδομένων υψηλής απόδοσης

Η αυξανόμενη ζήτηση για εφαρμογές υψηλής απόδοσης και υψηλής ρυθμάποδοσης έχει οδηγήσει στην ανάπτυξη βιβλιοθηκών επιπέδου δεδομένων (DPL-Data plane libraries) που επιτρέπουν στους προγραμματιστές να δημιουργούν αποτελεσματικές, κλιμακούμενες εφαρμογές. Ωστόσο, η διασφάλιση της απομόνωσης και της ασφάλειας στα DPL παραμένει μια πρόκληση. Η ερευνητική εργασία "Hodor: Intra-Process Isolation for High-Throughput Data Plane Libraries" αντιμετωπίζει αυτό το ζήτημα εισάγοντας έναν νέο μηχανισμό απομόνωσης εντός της διεργασίας που ονομάζεται Hodor. Παρακάτω σε αυτή τη πτυχιακή θα παρέχεται μια εις βάθος επισκόπηση της τεχνολογίας Hodor, των μηχανισμών της, του γενικού κόστους απόδοσης, της ασφάλειας και των πιθανών τρωτών σημείων στο πλαίσιο των DPL υψηλής απόδοσης.

3.3.1 Υπόβαθρο:

Οι παραδοσιακές λύσεις ασφάλειας που βασίζονται στον πυρήνα ενδέχεται να μην είναι κατάλληλες για DPL υψηλής απόδοσης λόγω των γενικών εξόδων που εισάγονται από κλήσεις συστήματος και διακόπτες περιβάλλοντος. Η Hodor παρουσιάζει έναν ελαφρύ μηχανισμό απομόνωσης εντός της διεργασίας που στοχεύει στην αντιμετώπιση αυτών των ζητημάτων, διατηρώντας παράλληλα υψηλή απόδοση και επεκτασιμότητα.

3.3.2 Βασικές έννοιες:

Η τεχνολογία Hodor αξιοποιεί τα κλειδιά προστασίας μνήμης (MPK) της Intel για να παρέχει απομόνωση μνήμης σε επίπεδο χρήστη, χωρίς την ανάγκη κλήσεων συστήματος ή εναλλαγών περιβάλλοντος. Ο προτεινόμενος σχεδιασμός αποτελείται από δύο κύρια στοιχεία: Lightweight Isolation Domains (LIDs) και το σύστημα χρόνου εκτέλεσης Hodor.

3.3.2.1 Ελαφριές περιοχές απομόνωσης (LID):

Τα LID είναι απομονωμένα περιβάλλοντα εκτέλεσης μέσα σε μία μόνο διαδικασία, που δημιουργήθηκαν από τη τεχνολογία Hodor για να παρέχει απομόνωση μνήμης. Σε κάθε

LID εκχωρείται ένας μοναδικός τομέας προστασίας MPK, διασφαλίζοντας ότι η πρόσβαση στη μνήμη περιορίζεται στο αντίστοιχο LID.

3.3.2.2 Σύστημα χρόνου εκτέλεσης Hodor:

Το σύστημα χρόνου εκτέλεσης Hodor είναι υπεύθυνο για τη διαχείριση των LID, το χειρισμό της επικοινωνίας μεταξύ των LID και την επιβολή πολιτικών απομόνωσης μνήμης. Το σύστημα χρόνου εκτέλεσης (runtime system) υλοποιείται ως βιβλιοθήκη σε επίπεδο χρήστη, επιτρέποντας την απρόσκοπτη ενσωμάτωση με υπάρχοντα DPL.

3.3.3 Πώς λειτουργεί το Hodor:

3.3.3.1 Δημιουργία και διαχείριση LID:

Η Hodor δημιουργεί LID εκχωρώντας μια ξεχωριστή περιοχή μνήμης για κάθε LID και ορίζοντας τον κατάλληλο τομέα προστασίας MPK. Το σύστημα χρόνου εκτέλεσης Hodor αρχικοποιεί κάθε LID με τα απαιτούμενα εξαρτήματα DPL και διαχειρίζεται τον κύκλο ζωής του LID.

3.3.3.2 Απομόνωση μνήμης και έλεγχος πρόσβασης:

Η Hodor επιβάλλει την απομόνωση μνήμης αξιοποιώντας τον μηχανισμό MPK της Intel, ο οποίος επιτρέπει λεπτομερή έλεγχο των δικαιωμάτων πρόσβασης στη μνήμη. Σε κάθε LID εκχωρείται ένας μοναδικός τομέας προστασίας και το σύστημα χρόνου εκτέλεσης Hodor διασφαλίζει ότι μόνο το LID ιδιοκτησίας μπορεί να έχει πρόσβαση στην προστατευμένη περιοχή μνήμης.

3.3.4 Επικοινωνία μεταξύ των LID:

Το Hodor παρέχει έναν ασφαλή μηχανισμό επικοινωνίας μεταξύ LID χρησιμοποιώντας περιοχές κοινόχρηστης μνήμης. Το σύστημα χρόνου εκτέλεσης Hodor μεσολαβεί στην πρόσβαση στην κοινόχρηστη μνήμη, διασφαλίζοντας ότι μόνο εξουσιοδοτημένα LID μπορούν να διαβάσουν ή να γράψουν τα δεδομένα.

3.3.4.1 Τραμπολίνα

Τα Τραμπολίνα αναφέρονται σε μια τεχνική που χρησιμοποιείται για την ασφαλή μετάβαση εκτέλεσης μεταξύ ελαφρών περιοχών απομόνωσης (LID) διατηρώντας παράλληλα την απομόνωση μνήμης και τον έλεγχο πρόσβασης. Τα Τραμπολίνα είναι ουσιαστικά κομμάτια κώδικα που χρησιμεύουν ως ασφαλής γέφυρα μεταξύ των LID για τη διευκόλυνση της επικοινωνίας, της κοινής χρήσης δεδομένων ή των κλήσεων λειτουργίας μεταξύ τους. Το σύστημα χρόνου εκτέλεσης (runtime) Hodor χρησιμοποιεί Τραμπολίνα για να διασφαλίσει ότι όταν η εκτέλεση μετακινείται από ένα LID σε άλλο, τα απαραίτητα δικαιώματα πρόσβασης μνήμης ρυθμίζονται σωστά και ενεργοποιείται το κλειδί προστασίας μνήμης (MPK) του LID προορισμού. Αυτό βοηθά στη διατήρηση αυστηρής απομόνωσης μνήμης και αποτρέπει τη μη εξουσιοδοτημένη πρόσβαση στις προστατευμένες περιοχές μνήμης.

3.3.4.2 Τα Τραμπολίνα στο Hodor λειτουργούν ως εξής:

Όταν ένα LID πρέπει να επικοινωνήσει με ένα άλλο LID ή να καλέσει μια λειτουργία σε άλλο LID, το κάνει μέσω μιας λειτουργίας τραμπολίνου που παρέχεται από το σύστημα χρόνου εκτέλεσης Hodor. Η λειτουργία τραμπολίνου είναι υπεύθυνη για τη ρύθμιση των απαραίτητων δικαιωμάτων πρόσβασης στη μνήμη και την ενεργοποίηση του MPK του στοχευόμενου LID. Μόλις ρυθμιστούν τα δικαιώματα μνήμης και ενεργοποιηθεί το MPK του LID προορισμού, η λειτουργία τραμπολίνου μεταφέρει τον έλεγχο εκτέλεσης στο LID προορισμού.

Όταν το LID προορισμού ολοκληρώσει την εκτέλεση της ζητούμενης λειτουργίας ή την επεξεργασία της επικοινωνίας, ο έλεγχος επιστρέφει στο αρχικό LID μέσω μιας άλλης λειτουργίας τραμπολίνου, η οποία επαναφέρει τα δικαιώματα πρόσβασης μνήμης και τις ρυθμίσεις MPK του αρχικού LID.

Χρησιμοποιώντας Τραμπολίνα, το σύστημα χρόνου εκτέλεσης Hodor μπορεί να διαχειριστεί με ασφάλεια την επικοινωνία μεταξύ LID και τις κλήσεις λειτουργίας, διατηρώντας παράλληλα αυστηρή απομόνωση μνήμης και τον έλεγχο πρόσβασης, ο

οποίος είναι ζωτικής σημασίας για τη διασφάλιση της ασφάλειας και της ακεραιότητας των βιβλιοθηκών επιπέδου δεδομένων υψηλής απόδοσης.

3.3.5 Γενικά κόστη απόδοσης και δοκιμές:

3.3.5.1 Γενικά κόστη απόδοσης:

Η Hodor εισάγει ελάχιστα γενικά κόστη απόδοσης λόγω της εφαρμογής σε επίπεδο χρήστη και της χρήσης του MPK της Intel για απομόνωση μνήμης. Αυτός ο ελαφρύς σχεδιασμός επιτρέπει στη Hodor να διατηρεί υψηλή απόδοση και επεκτασιμότητα σε DPL υψηλής απόδοσης.

3.3.5.2 Δοκιμές απόδοσης:

Το ερευνητικό άρθρο του Hodor παρουσιάζει διάφορες δοκιμές απόδοσης, συμπεριλαμβανομένων microbenchmarks αναφοράς και σημείων αναφοράς εφαρμογής, για την αξιολόγηση της απόδοσης και της επεκτασιμότητας του προτεινόμενου μηχανισμού απομόνωσης εντός της διαδικασίας. Τα αποτελέσματα δείχνουν ότι η Hodor εισάγει ελάχιστα γενικά έξοδα, παρέχοντας παράλληλα αποτελεσματική απομόνωση και ασφάλεια για DPL.

3.3.5.3 Ασφάλεια και ευπάθειες:

Ασφάλεια:

Η τεχνολογία Hodor παρέχει βελτιωμένη απομόνωση εντός της διαδικασίας και ασφάλεια για DPL υψηλής απόδοσης, αξιοποιώντας τον μηχανισμό MPK της Intel και εφαρμόζοντας ένα ελαφρύ σύστημα χρόνου εκτέλεσης για τη διαχείριση LID. Αυτή η προσέγγιση βοηθά στην ελαχιστοποίηση πιθανών επιφανειών επίθεσης, διατηρώντας παράλληλα υψηλή απόδοση και επεκτασιμότητα.

Πιθανές ευπάθειες:

Παρόλο που η Hodor προσφέρει σημαντικές βελτιώσεις στην απομόνωση και την ασφάλεια εντός της διαδικασίας, ενδέχεται να υπάρχουν πιθανές ευπάθειες ή περιορισμοί που σχετίζονται με αυτήν την προσέγγιση:

1. **Ευπάθειες σε επίπεδο υλικού:** Καθώς η Hodor βασίζεται στον μηχανισμό MPK της Intel, τυχόν ευπάθειες ή ελαττώματα στην εφαρμογή του MPK θα μπορούσαν ενδεχομένως να επηρεάσουν την αποτελεσματικότητα του Hodor στην παροχή απομόνωσης.

2. **Συμβατότητα με άλλους μηχανισμούς ασφαλείας:** Η ενσωμάτωση του Hodor με άλλους μηχανισμούς ασφαλείας, όπως κρυπτογράφηση ή πρόσθετα μέτρα ελέγχου πρόσβασης, μπορεί να δημιουργήσει πολυπλοκότητες και πιθανές συγκρούσεις που απαιτούν προσεκτική εξέταση και σχεδιασμό.

3. **Δυνατότητα εφαρμογής σε πλατφόρμες που δεν ανήκουν στην Intel:** Η Hodor αξιοποιεί τον MPK μηχανισμό της Intel για απομόνωση μνήμης και η εφαρμογή του σε πλατφόρμες που δεν ανήκουν στην Intel παραμένει ένα ανοιχτό ερώτημα. Ενδέχεται να χρειαστεί να διερευνηθούν εναλλακτικοί μηχανισμοί απομόνωσης βάσει υλικού για ευρύτερη εφαρμογή.

Προβλήματα που αντιμετωπίσαμε

Αντιμετωπίσαμε προβλήματα συμβατότητας κατά την προσπάθεια υιοθέτησης του Hodor. Έπρεπε να γίνουν τροποποιήσεις σε επίπεδο kernel, γι’ αυτό κατά τη διάρκεια της διπλωματικής αυτής, ξεκίνησα το έργο της δημιουργίας ενός προσαρμοσμένου πυρήνα για το λειτουργικό σύστημα. Ο στόχος ήταν να αποκτηθεί μια βαθύτερη κατανόηση της αρχιτεκτονικής του πυρήνα και των σχετικών ενοτήτων. Η διαδικασία περιελάμβανε τη λήψη και την εξαγωγή του πυρήνα Linux από τον επίσημο ιστότοπο, τη διαμόρφωση του πυρήνα, την επίλυση προκλήσεων που σχετίζονται με τη διαμόρφωση και, τέλος, την κατασκευή του πυρήνα. Πιο κάτω παρέχεται μια σύντομη επισκόπηση των βημάτων που απαιτούνται για τη ρύθμιση του πυρήνα,

επισημαίνονται τα προβλήματα ρύθμισης που αντιμετωπίστηκαν και αναφέρεται η γνώση που αποκτήθηκε από αυτή την προσπάθεια.

Βήματα:

Η δημιουργία ενός προσαρμοσμένου πυρήνα βασισμένου στην έκδοση βανίλιας Linux 4.15 απαιτούσε μια συστηματική προσέγγιση. Η διαδικασία ξεκίνησε με την απόκτηση του απαραίτητου πηγαίου κώδικα για την επιθυμητή έκδοση πυρήνα. Μετά την εξαγωγή του πηγαίου κώδικα, προχώρησα στη ρύθμιση του πυρήνα σύμφωνα με τις συγκεκριμένες απαιτήσεις της διπλωματικής μου για την υιοθέτηση του Hodor.

Κατά τη διάρκεια της φάσης διαμόρφωσης, αντιμετώπισα διάφορες προκλήσεις που σχετίζονται με συγκεκριμένες ρυθμίσεις. Αυτές οι προκλήσεις περιλάμβαναν εσφαλμένα καθορισμένες παραμέτρους, μη συμβατές ρυθμίσεις παραμέτρων και εξαρτήσεις που έλειπαν. Αυτά τα ζητήματα οδήγησαν σε σφάλματα μεταγλώττισης και εμπόδιζαν την επιτυχή δημιουργία μιας λειτουργικής εικόνας πυρήνα. Ωστόσο, μέσω επιμελούς έρευνας και αντιμετώπισης προβλημάτων, μπόρεσα να εντοπίσω και να επιλύσω αυτά τα προβλήματα διαμόρφωσης ένα προς ένα.

Ένα ζήτημα αφορούσε την έλλειψη modules, τα οποία ήταν κρίσιμα για τη σωστή λειτουργικότητα του πυρήνα.

Εντολές

Εγκατάσταση πυρήνα
wget https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.12.10.tar.xz
Εξαγωγή του πηγαίου κώδικα του πυρήνα
tar -xf linux-5.12.10.tar.xz
Πλοήγηση στον κατάλογο πυρήνα
cd linux-5.12.10

Εγκατάσταση απαιτούμενων εργαλείων
sudo apt update && sudo apt install build-essential
Καθαρισμός kernel source directory (προαιρετικό)
make mrproper
Ρύθμιση του πυρήνα
make defconfig
Προσαρμογή ρυθμίσεων πυρήνα
make menuconfig
Αποθήκευση του αρχείου ρύθμισης παραμέτρων
Εγκατάσταση απαιτούμενων βιβλιοθηκών
sudo apt-get install libncurses5-dev libncursesw5-dev
Δημιουργία πυρήνα
make -j4
Εγκατάσταση modules πυρήνα
make modules_install
Ενημέρωση διαμόρφωσης φορτωτή εκκίνησης (GRUB)
sudo update-grub
Επανεκκίνηση συστήματος
sudo reboot

Στο αρχείο διαμόρφωσης Hodor, η παράμετρος 'CONFIG_RANDOMIZE_BASE' ήταν μια πιθανή αιτία για το πρόβλημα που αντιμετωπίσαμε. Δοκιμάσαμε μια διαμόρφωση που απενεργοποιεί την τυχαιοποίηση βάσης ορίζοντας το 'CONFIG_RANDOMIZE_BASE=n'. Επιπλέον, καθορίσαμε άλλες επιλογές διαμόρφωσης, όπως 'CONFIG_LOCALVERSION', 'CONFIG_SYSTEM_TRUSTED_KEYS', 'CONFIG_PERCPU_PGTBL' και 'CONFIG_PERCPU_SCRATCH_PAGE'. Αυτές οι ρυθμίσεις τροποποιούν διάφορες πτυχές της διαμόρφωσης του πυρήνα, συμπεριλαμβανομένης της συμβολοσειράς τοπικής έκδοσης, των αξιόπιστων κλειδιών συστήματος και της διαχείρισης μνήμης ανά CPU.

Εξετάζοντας προσεκτικά τα μηνύματα σφάλματος και συμβουλευόμενος το documentation και το διαδίκτυο, απέκτησα γνώσεις σχετικά με τις σωστές επιλογές ρύθμισης για τον επιθυμητό πυρήνα. Έκανα τις απαραίτητες προσαρμογές, τροποποίησα τα αρχεία ρυθμίσεων και μεταγλώττισα ξανά τον πυρήνα μέχρι να επιλυθούν τα σφάλματα. Αυτή η επαναληπτική διαδικασία περιελάμβανε μια βαθιά κατανόηση της αρχιτεκτονικής του πυρήνα και των διαφόρων συστατικών του.

Παρόλο που έγινε με επιτυχία η κατασκευή του πυρήνα (Linux 4.15) δεν κατέστη δυνατόν να φορτωθεί στο μηχάνημα του εργαστηρίου. Επιλέξαμε όμως να ακολουθήσουμε τον μηχανισμό προστασίας που υιοθετεί ο Hodor (Memory protection keys) στην πιο βασική του μορφή.

Συμπέρασμα:

Η Hodor τεχνολογία παρουσιάζει έναν νέο μηχανισμό απομόνωσης εντός της διαδικασίας για βιβλιοθήκες επιπέδων δεδομένων υψηλής απόδοσης, αντιμετωπίζοντας τις προκλήσεις της παροχής αποτελεσματικής απομόνωσης και ασφάλειας χωρίς συμβιβασμούς στην απόδοση. Αξιοποιώντας τα κλειδιά προστασίας μνήμης της Intel και εφαρμόζοντας ένα ελαφρύ σύστημα χρόνου εκτέλεσης για τη διαχείριση ελαφρών τομέων απομόνωσης, η Hodor εξασφαλίζει αποτελεσματική απομόνωση μνήμης και έλεγχο πρόσβασης σε DPL.

Ενώ η Hodor προσφέρει σημαντικές βελτιώσεις στην απομόνωση και την ασφάλεια εντός της διαδικασίας, είναι σημαντικό να αντιμετωπιστούν πιθανές ευπάθειες, ζητήματα σε επίπεδο υλικού και προκλήσεις ενοποίησης με άλλους μηχανισμούς ασφαλείας. Η συνεχής έρευνα και ανάπτυξη στον τομέα της απομόνωσης εντός της διαδικασίας για βιβλιοθήκες δεδομένων υψηλής απόδοσης, όπως η Hodor, υπόσχονται την ενίσχυση της ασφάλειας και της αποτελεσματικότητας των σύγχρονων, υψηλής απόδοσης εφαρμογών.

Δύναται να τονιστεί όπως προαναφέραμε και πιο πριν πως στην προσπάθεια μας να υιοθετήσουμε αυτήν την τεχνολογία, δεν κατεστει δυνατό η εγκατάσταση και λειτουργία του Hodor.

Κεφάλαιο 4

Πειραματική σύγκριση τεχνολογιών προστασίας μνήμης

4.1 Περιγραφή πειράματος

Όπως αναλύθηκε στις προηγούμενες ενότητες της παρούσας διπλωματικής εργασίας, θα ξεκινήσουμε μια λεπτομερή εμπειρική έρευνα για να αναλύσουμε τη συγκριτική αποτελεσματικότητα δύο διακριτών τεχνολογιών που χρησιμοποιούνται για την προστασία της μνήμης: τη παραδοσιακή προσέγγιση mprotect και τη μεθοδολογία των κλειδιών προστασίας MPK. Η συγκριτική αξιολόγηση θα επικεντρωθεί στις μετρήσεις της ακρίβειας της μνήμης, της χρονικής αποτελεσματικότητας και του χωρικού χειρισμού. Το τελευταίο αναφέρεται στον τρόπο με τον οποίο αυτές οι δύο τεχνολογίες μπορούν να προκαλέσουν γενικά έξοδα με πιθανή σπατάλη μνήμης.

Το πείραμα θα χρησιμοποιήσει τη λειτουργία Linux pkeys, ένα ολοκληρωμένο εργαλείο συστήματος που επιτρέπει σε μια διαδικασία να ορίσει ή να αφαιρέσει περιορισμούς πρόσβασης σε ξεχωριστές περιοχές μνήμης. Για να το δείξουμε αυτό, στο **Παράρτημα Α - 1** βρίσκεται ο πηγαίος κώδικα για την εκτέλεση αυτού του πειράματος χρησιμοποιώντας την κλασσική προστασία και στο **παράρτημα Α - 2** με την χρήση των MPK. Αυτή η προσπάθεια στοχεύει να αναδείξει τις διαφορές στα γενικά έξοδα που

σχετίζονται με αυτές τις δύο διαφορετικές αλλά διαδεδομένες προσεγγίσεις προστασίας μνήμης, καθοδηγώντας τελικά τις μελλοντικές αποφάσεις στην επιλογή τεχνολογίας και τη βελτιστοποίηση του συστήματος.

4.2 Εξήγηση κώδικα Παραρτήματος Α-5

Αυτό το πρόγραμμα χρησιμεύει ως σημείο αναφοράς απόδοσης για τη λειτουργία των Linux pkeys, η οποία επιτρέπει σε μια διαδικασία να περιορίσει την πρόσβαση σε συγκεκριμένες περιοχές μνήμης. Αρχικά, το πρόγραμμα δεσμεύει ένα αρκετά μεγάλο buffer 4GB χρησιμοποιώντας το 'calloc()' και δημιουργεί ένα κλειδί προστασίας μέσω του 'pkey_alloc()'. Στη συνέχεια, μετακινείται σε μια σειρά από αντίγραφα ξεκινώντας από [3, 13, 50, 100, 512, 1024] και τα υπόλοιπα 20 μεγέθη υπολογίζονται αυξάνοντας το προηγούμενο μέγεθος κατά 4096.

Το πρόγραμμα κάνει κύκλους σε αυτά τα μεγέθη και για κάθε μέγεθος, εκτελεί 'pkey_mprotect()' σε ξεχωριστά κομμάτια του buffer. Κάθε μπλοκ που πρέπει να προστατευθεί έχει μέγεθος X byte, όπου X είναι το τρέχον μέγεθος αντιγραφής, με επιπλέον 4097 byte μεταξύ κάθε μπλοκ για να αποφευχθεί η επικάλυψη σελίδων. Οι κλήσεις 'pkey_mprotect()' αλλάζουν την προστασία μνήμης κάθε κομματιού για να επιτρέψουν την πρόσβαση ανάγνωσης/εγγραφής.

Το πρόγραμμα μετρά με ακρίβεια το χρόνο που απαιτείται για την εκτέλεση αυτών των λειτουργιών 'pkey_mprotect()' για κάθε μέγεθος αντιγράφου. Υπολογίζει επίσης τον αριθμό των πράξεων «pkey_mprotect()» που εκτελέστηκαν. Αυτές οι μετρήσεις εξάγονται στη συνέχεια, παρέχοντας τον συνολικό χρόνο, τον αριθμό των λειτουργιών και τον μέσο χρόνο ανά λειτουργία για κάθε μέγεθος αντιγράφου.

Ξεχωριστά, το πρόγραμμα εκτελεί μια σειρά από 2000 λειτουργίες «pkey_set()», που εναλλάσσονται μεταξύ της ρύθμισης του κλειδιού προστασίας σε 0 και «PKEY_DISABLE_ACCESS». Ο χρόνος που απαιτείται για την ολοκλήρωση αυτών των 2000 πράξεων μετράται και διαιρείται διά του 2000 για να βρεθεί ο μέσος χρόνος ανά πράξη «pkey_set()».

Τέλος, η μνήμη buffer δεν απελευθερώνεται ρητά από το πρόγραμμα. Αυτό γίνεται αυτόματα από το λειτουργικό σύστημα κατά την έξοδο του προγράμματος. Το κλειδί προστασίας απελευθερώνεται χρησιμοποιώντας το 'pkey_free()'. Παρακάτω παρατίθεται μια γραφική απεικόνιση της πιο πάνω περιγραφής.

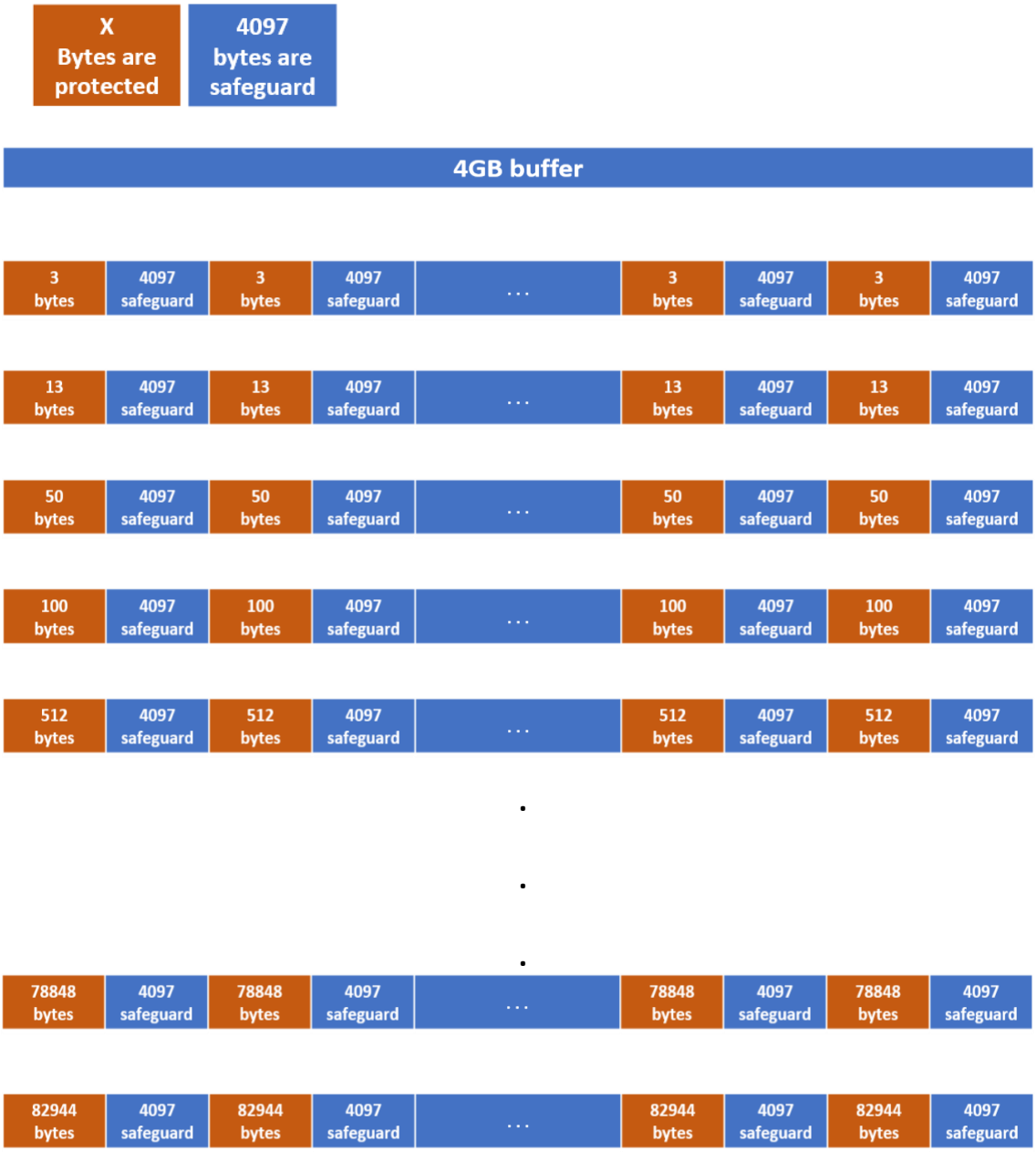

Figure 9 Επαναληπτική προστασία buffer

4.3 Υποθέσεις

Πριν την έναρξη της πειραματικής μελέτης διατυπώσαμε τις ακόλουθες υποθέσεις:

Σταθερότητα του χρόνου ρύθμισης κλειδιού: Αναμένουμε ότι ο χρόνος ρύθμισης του κλειδιού προστασίας θα παραμείνει σταθερός ανεξάρτητα από το μέγεθος ή τον αριθμό των περιοχών μνήμης που προστατεύονται.

Καλύτερη απόδοση με χρήση κλειδιών προστασίας μνήμης: Υποθέτουμε ότι η προστασία της μνήμης χρησιμοποιώντας κλειδιά προστασίας μνήμης θα παρέχει καλύτερη απόδοση — όσον αφορά τον ταχύτερο χρόνο εκτέλεσης — σε σύγκριση με την παραδοσιακή μέθοδο mprotect.

Επιβάρυνση για μικρά buffer: Αναμένουμε ότι τόσο με την χρήση mprotect όσο και την χρήση κλειδιών προστασίας θα εμφανίζεται επιβάρυνση κατά την προστασία μικρών buffer. Αυτή η επιβάρυνση μπορεί να προκύψει από σταθερό κόστος που σχετίζεται με κάθε λειτουργία προστασίας, το οποίο μπορεί να είναι σημαντικό όταν η περιοχή προστατευμένης μνήμης είναι μικρή.

Επιβάρυνση του mprotect με MAP_POPULATE: Αναμένουμε ότι το mprotect με MAP_POPULATE, το οποίο αναγκάζει τη δημιουργία καταχωρίσεων πίνακα σελίδων για μια περιοχή μνήμης, θα επιφέρει υψηλότερη επιβάρυνση από τα κλειδιά προστασίας μνήμης.

Αυτές οι υποθέσεις αντικατοπτρίζουν τις προσδοκίες μας με βάση τα εγγενή χαρακτηριστικά των κλειδιών προστασίας μνήμης και των κλήσεων λειτουργιών mprotect, μαζί με τη συμπεριφορά του υποσυστήματος διαχείρισης μνήμης του Linux.

4.4 Συμπεράσματα

Τα πειραματικά αποτελέσματα επικυρώνουν ορισμένες υποθέσεις, παρέχοντας παράλληλα νέες ιδέες που βελτιώνουν την κατανόησή μας για τα χαρακτηριστικά απόδοσης των κλειδιών προστασίας μνήμης και της λειτουργίας mprotect.

Figure 10 Χρόνος για ορισμό κλειδιού χωρίς το αρχικό ζέσταμα

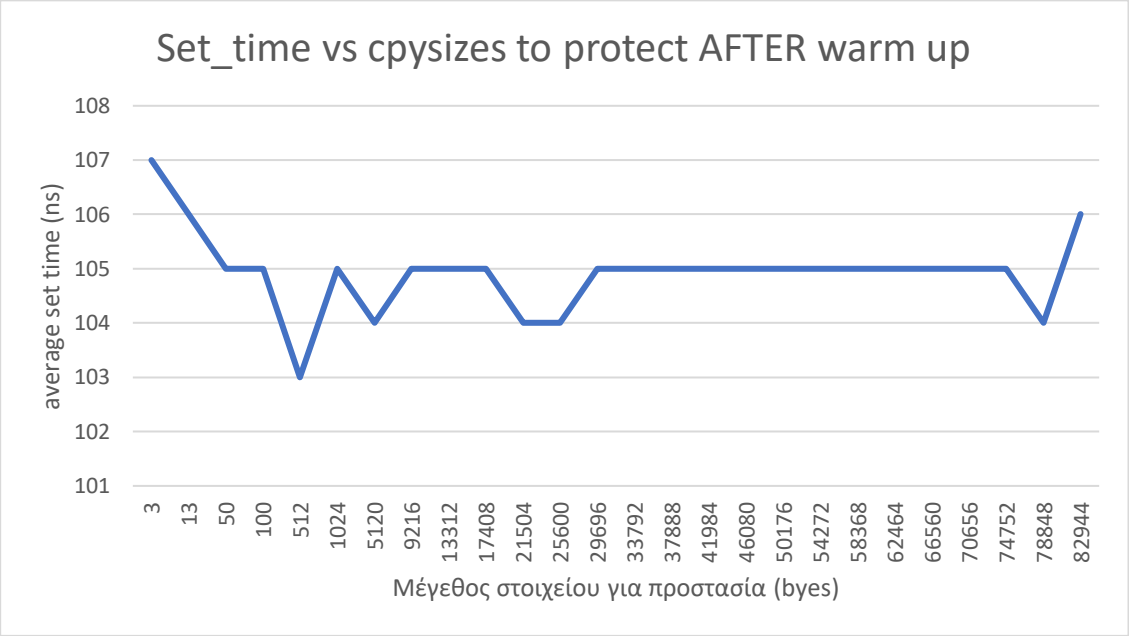

Figure 11 Χρόνος για ορισμό κλειδιού με το αρχικό ζέσταμα

Χρόνος ρύθμισης κλειδιών

Τα αποτελέσματα αποκαλύπτουν ότι η πρώτη κλήση για τη ρύθμιση του κλειδιού προστασίας έχει πράγματι κόστος, πιθανώς λόγω της ανάγκης προετοιμασίας του κλειδιού. Ωστόσο, οι επόμενες κλήσεις για τον καθορισμό του κλειδιού παρουσίασαν σταθερή απόδοση, σύμφωνα με την πρώτη μας υπόθεση.

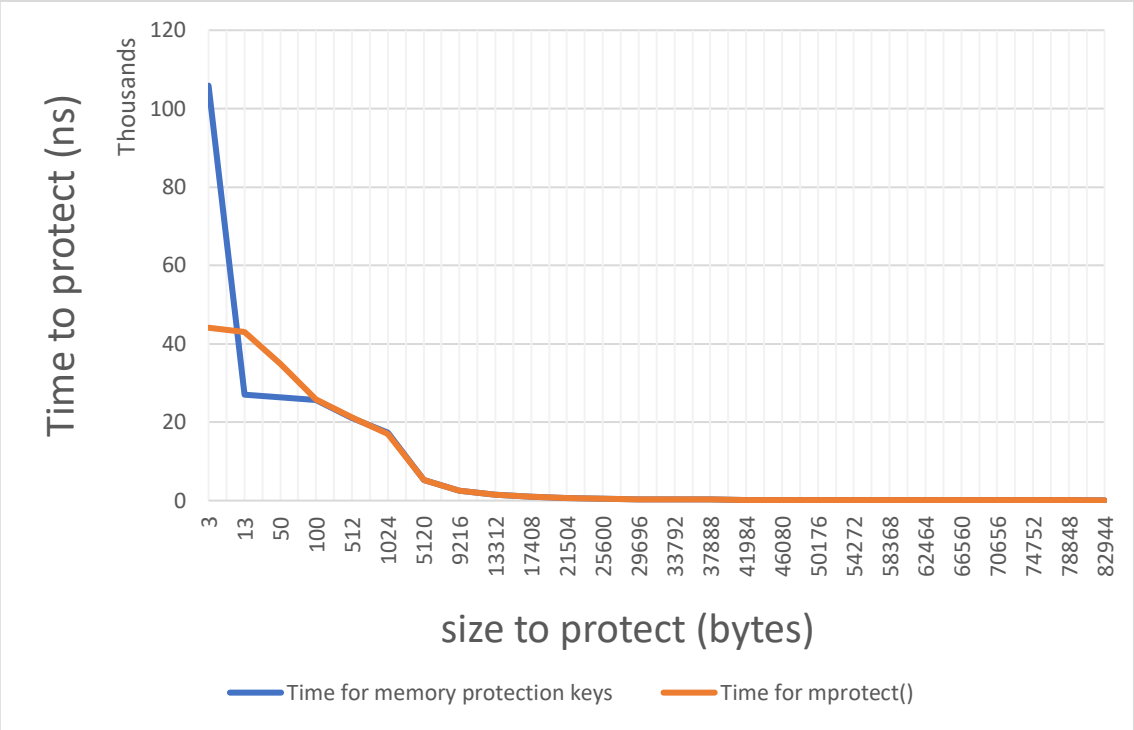

Figure 12 Χρόνος για προστασία X bytes με χρήση MPK, mprotect

Χρόνος προστασίας για buffer

Σε αντίθεση με τη δεύτερη υπόθεσή μας, τα πειραματικά αποτελέσματα έδειξαν ότι ο χρόνος που απαιτείται για την προστασία ενός buffer είναι πολύ παρόμοιος όταν χρησιμοποιούνται είτε κλειδιά προστασίας μνήμης είτε η λειτουργία mprotect. Αυτό επιφανειακά υποδηλώνει ότι και οι δύο τεχνολογίες έχουν συγκρίσιμες επιδόσεις όσον

αφορά την προστασία buffers μεγαλύτερων μεγεθών. Παρόλα αυτά πιο κάτω προχωρήσαμε σε περαιτέρω αξιολόγηση του φαινομένου αυτού. (στην περίπτωση αυτή οι πίνακες σελίδων δεν γίνονταν invalidate. Σε μετέπειτα στάδιο τους αναγκάζαμε να ακυρώνονται στην αρχή αλλά και σε κάθε επανάληψη για πιο ποιοτικά αποτελέσματα)

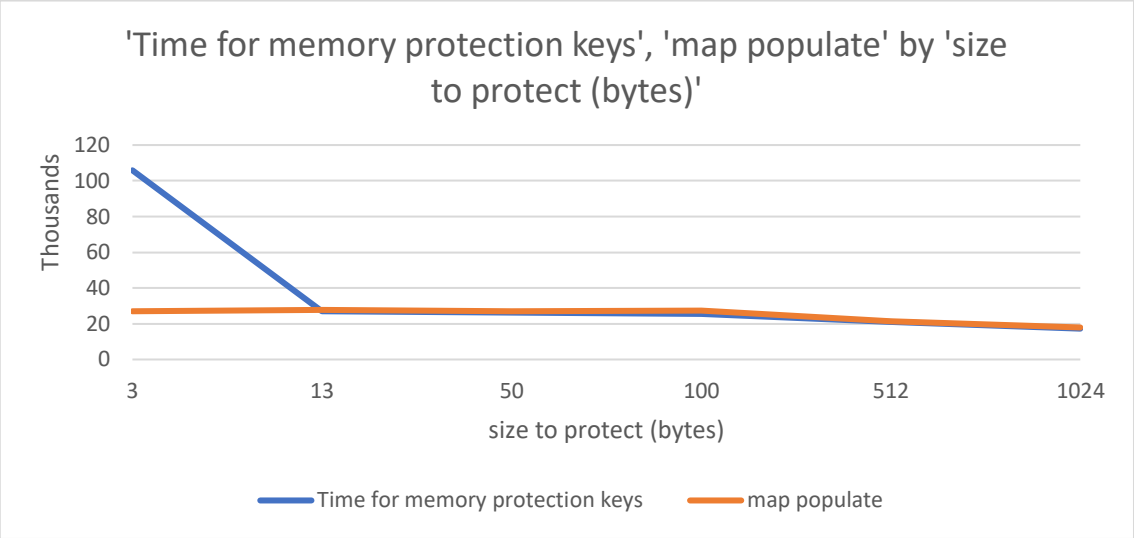

Figure 13 Χρόνος για προστασία μικρών buffer με χρήση mprotect με map populate στην αρχή vs MPK

Γενικά έξοδα για μικρά buffers

Σύμφωνα με την τρίτη μας υπόθεση, τα πειράματά μας επιβεβαίωσαν την ύπαρξη γενικών εξόδων τόσο στα κλειδιά προστασίας μνήμης όσο και στο mprotect κατά την προστασία μικρών περιοχών μνήμης. Είναι ενδιαφέρον ότι τα κλειδιά προστασίας μνήμης φαίνεται να χειρίζονται αυτές τις μικρές περιοχές μνήμης πιο αποτελεσματικά από τη λειτουργία mprotect, υποδεικνύοντας ότι τα κλειδιά προστασίας μνήμης μπορεί να είναι μια καλύτερη επιλογή για εφαρμογές που συχνά προστατεύουν μικρές περιοχές μνήμης.

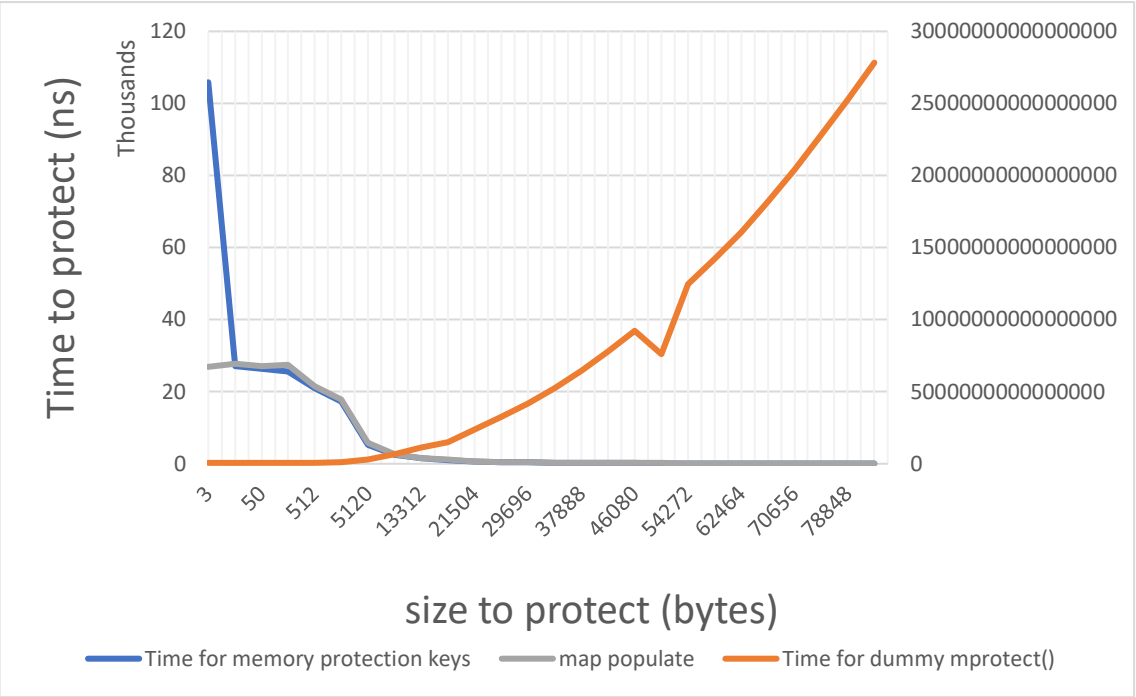

Figure 14 Χρόνος για προστασία X bytes με χρήση MPK, mprotect με map populate (στην αρχή), mprotect με map populate σε κάθε επανάληψη.

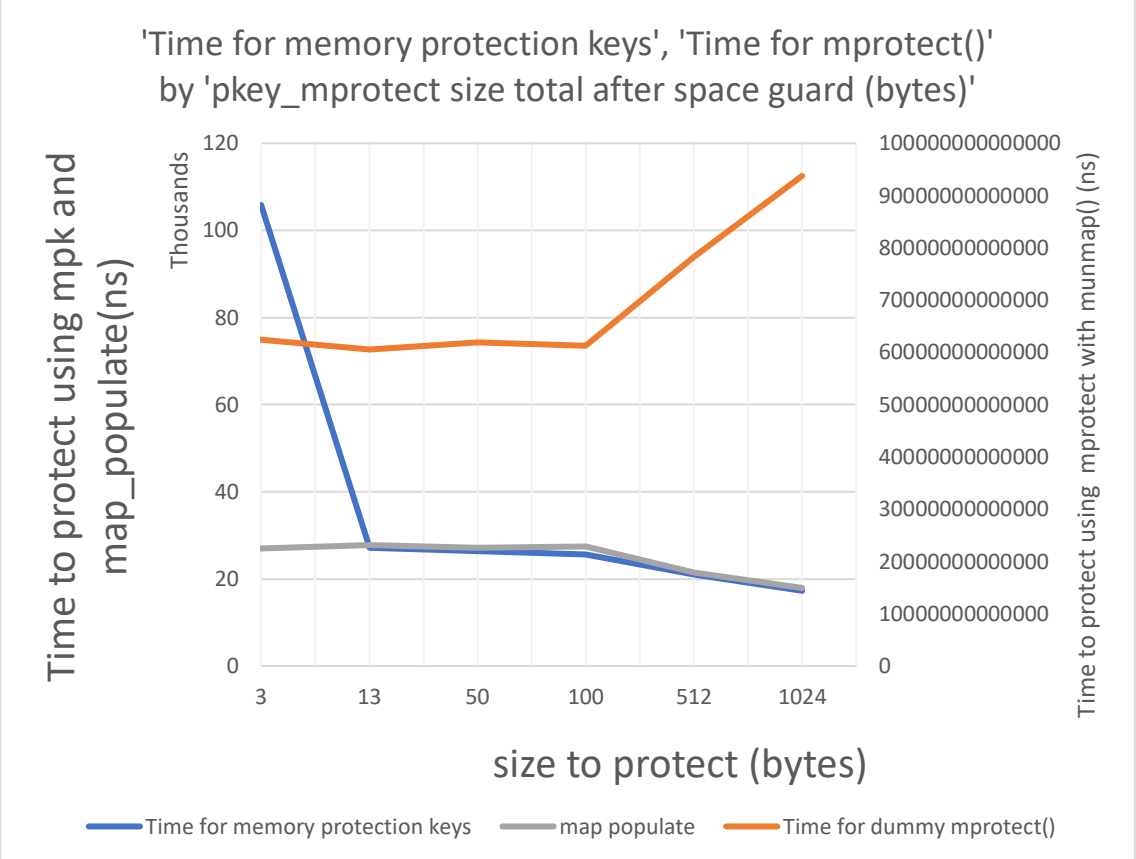

Figure 15 Χρόνος για προστασία μικρών buffers με χρήση MPK, mprotect με map populate (στην αρχή), mprotect με map populate σε κάθε επανάληψη.

Επιβάρυνση mprotect με MAP_POPULATE

Σύμφωνα με την τέταρτη υπόθεσή μας, τα αποτελέσματά μας αποκάλυψαν πράγματι μια αργή εκκίνηση και μια αύξηση του χρόνου κατά τη χρήση του MAP_POPULATE στο mprotect στην αρχή σε ένα πείραμα και σε κάθε επανάληψη σε άλλο πείραμα (το οποίο ονομάσαμε dummy mprotect). Αυτό υποδηλώνει ότι ο εξαναγκασμός δημιουργίας καταχωρήσεων πίνακα σελίδων για μια περιοχή μνήμης συνεπάγεται πρόσθετη επιβάρυνση, καθιστώντας ενδεχομένως αυτήν την τεχνολογία μη αποδοτική καθότι οι πίνακες θα τυγχάνουν invalidation συχνά.

Κεφάλαιο 5

Υλοποίηση key-value store

5.1 Περιγραφή key-value store:

Όπως συζητήσαμε σε προηγούμενες ενότητες αυτής της διπλωματικής εργασίας, τα παραδοσιακά μοντέλα διαχείρισης μνήμης συχνά υστερούν όσον αφορά την αξιοπιστία, την απόδοση και την ασφάλεια. Ο λόγος πηγάζει από την εξάρτηση του μοντέλου από το λειτουργικού συστήματος για την εκχώρηση μνήμης και την απελευθέρωση μνήμης. Αυτή η εξάρτηση μπορεί να οδηγήσει σε ασυνέπειες, αργές λειτουργίες ανάγνωσης και εγγραφής και ευπάθειες που εκθέτουν το σύστημα σε πιθανές παραβιάσεις.

Ο απώτερος στόχος μας ήταν να προσφέρουμε μια βελτιωμένη εναλλακτική λύση που μπορεί να μετριάσει αυτά τα ζητήματα. Η στρατηγική μας περιελάμβανε την αξιοποίηση της δυνατότητας που προσφέρει η τεχνολογίας επίμονης μνήμης, την ενίσχυσή της με κλειδιά προστασίας μνήμης για την προστασία των αποθηκευμένων δεδομένων και τη δημιουργία key-value store.

Η επιλογή της τεχνολογίας επίμονης μνήμης επηρεάστηκε από τα μοναδικά πλεονεκτήματά της έναντι της παραδοσιακής πτητικής μνήμης. Ενώ η πτητική μνήμη χάνει το περιεχόμενό της όταν διακόπτεται η τροφοδοσία, η επίμονη μνήμη διατηρεί

τα αποθηκευμένα δεδομένα, ακόμη και κατά την επανεκκίνηση του συστήματος ή την απώλεια ισχύος. Αυτός ο μη πτητικός χαρακτήρας διασφαλίζει την ανθεκτικότητα των δεδομένων και μπορεί να εξοικονομήσει πόρους που διαφορετικά θα χρησιμοποιούνταν στις διαδικασίες ανάκτησης δεδομένων. Ενσωματώσαμε το Persistent Memory Development Kit (PMDK) της Intel στην υλοποίησή μας. Όπως προαναφέραμε σε περισσότερη λεπτομέρεια σε προηγούμενο κεφάλαιο το PMDK είναι μια συλλογή βιβλιοθηκών ειδικά σχεδιασμένων για προγραμματισμό επίμονης μνήμης. Παρέχει ένα επίπεδο αφαίρεσης πάνω από το υλικό, επιτρέποντάς μας να χειριστούμε την επίμονη μνήμη ακριβώς όπως η παραδοσιακή μνήμη σωρού. Αυτό απλοποιεί σημαντικά τη διαδικασία υλοποίησης και ενισχύει την αναγνωσιμότητα και τη συντηρησιμότητα του κώδικά μας.

Για να δομήσουμε τα δεδομένα μας, επιλέξαμε ένα μοντέλο ζεύγους κλειδιού-τιμής. Κάθε στοιχείο δεδομένων στο store μας είναι ένα ζεύγος κλειδιού-τιμής, όπου τόσο το κλειδί όσο και η τιμή είναι συστοιχίες χαρακτήρων σταθερών μεγεθών. Αυτή η δομή είναι συμπαγής, εύκολη στο χειρισμό και κατάλληλη για διάφορες εφαρμογές. Επιπλέον, έχουμε θέσει ένα ανώτατο όριο στον αριθμό των ζευγών κλειδιού-τιμής που μπορούν να αποθηκευτούν (για σκοπούς του πειράματος), βελτιστοποιώντας τη χρήση μνήμης και ενισχύοντας την απόδοση. Αλλά η απλή αποθήκευση δεδομένων σε μόνιμη μνήμη δεν αρκεί. Η μνήμη που φιλοξενεί αυτά τα ζεύγη κλειδιού-τιμής πρέπει να προστατεύεται από μη εξουσιοδοτημένη ή ακούσια πρόσβαση για να διατηρηθεί η ακεραιότητα και η εμπιστευτικότητα των δεδομένων. Εδώ εφαρμόζονται στην πράξη τα κλειδιά προστασίας μνήμης (MPK). Διαθέσιμο σε σύγχρονους επεξεργαστές Intel, το PKU προσφέρει ελέγχους πρόσβασης που επιβάλλονται από το υλικό σε συγκεκριμένες περιοχές μνήμης.

Στο store κλειδιών-τιμών, αξιοποιούμε τον MPK μηχανισμό για να ορίσουμε συγκεκριμένα δικαιώματα πρόσβασης για την περιοχή μνήμης που περιέχει τα δεδομένα μας. Αυτά τα δικαιώματα στη συνέχεια ενισχύονται από το υλικό, δημιουργώντας ένα ισχυρό εμπόδιο κατά της παράνομης πρόσβασης στη μνήμη. Το

store κλειδιού-τιμής προσφέρει τέσσερις κύριες λειτουργίες: εισαγωγή, διαγραφή, ενημέρωση και αναζήτηση. Κάθε λειτουργία ξεκινά με την κατάργηση της προστασίας της περιοχής μνήμης που περιέχει τα ζεύγη κλειδιού-τιμής, ακολουθούμενη από την εκτέλεση της απαιτούμενης λειτουργίας και, τέλος, την εκ νέου προστασία της μνήμης. Αυτή η ακολουθία λειτουργιών διασφαλίζει ότι τα δεδομένα μας είναι ασφαλή, ακόμη και όταν τροποποιούμε το χώρο αποθήκευσης κλειδιού-τιμής.

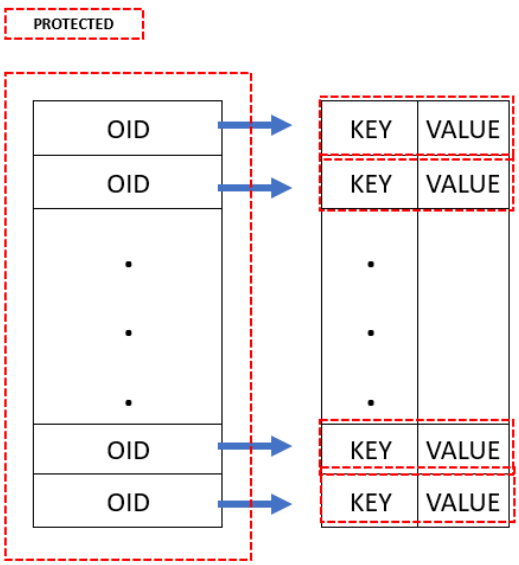

Figure 17 Προστασία key-value store με MPK - Αρχικός στόχος υλοποίησης

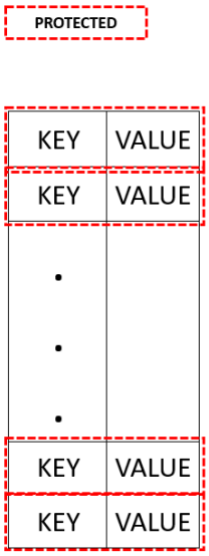

Figure 16 Προστασία key-value store με MPK - τελική υλοποίηση

Συμπερασματικά, η διπλωματική αυτή ήταν μια προσπάθεια να αλλάξουμε τον τρόπο που σκεφτόμαστε για τη διαχείριση της μνήμης. Ενσωματώνοντας τεχνολογία μόνιμης μνήμης, κλειδιά προστασίας μνήμης και χώρο αποθήκευσης κλειδιού-τιμής, δημιουργήσαμε μια λύση που όχι μόνο ξεπερνά τα μειονεκτήματα των παραδοσιακών μοντέλων μνήμης, αλλά προσφέρει επίσης βελτιωμένη αξιοπιστία, απόδοση και ασφάλεια. Αυτή η υλοποίηση αποτελεί απόδειξη των πρακτικών εφαρμογών αυτών των προηγμένων τεχνολογιών, προσφέροντας ένα μελλοντικό μοντέλο για ασφαλή και αποτελεσματική αποθήκευση δεδομένων.

5.2 Εξήγηση κώδικα key-value store σε persistent memory A- 11

1. **Αρχικοποίηση:** Ξεκινάμε ρυθμίζοντας το pool μνήμης χρησιμοποιώντας PMDK. Αυτό περιλαμβάνει τον καθορισμό της διάταξης του pool μας, όπου καθορίζουμε τη δομή των ζευγών κλειδιού-τιμής και τυχόν βοηθητικών δομών δεδομένων που χρειαζόμαστε. Μετά από αυτό, χρησιμοποιούμε τη λειτουργία 'pmemobj_create' του PMDK για να δημιουργήσουμε το pool, θέτοντας σαν όρισμα στην συνάρτηση το όνομα του αρχείου όπου θα αποθηκευτεί το pool μνήμης, η διάταξη και το μέγεθος του pool.

2. **Προστασία μνήμης:** Στη συνέχεια, ρυθμίζουμε τα κλειδιά προστασίας μνήμης χρησιμοποιώντας το PKU της Intel. Ξεκινάμε προσδιορίζοντας την περιοχή μνήμης που θέλουμε να προστατεύσουμε, η οποία σε αυτήν την περίπτωση είναι το μόνιμο pool. Στη συνέχεια, χρησιμοποιούμε την κλήση συστήματος «pkey_alloc» για να εκχωρήσουμε ένα νέο κλειδί προστασίας και τη λειτουργία «pkey_mprotect» για να εφαρμόσουμε αυτό το κλειδί στην περιοχή μνήμης, ορίζοντας έτσι τα δικαιώματα πρόσβασης για τα δεδομένα.

3**. Ρύθμιση δομής δεδομένων:** Στη συνέχεια, δημιουργούμε το store κλειδιού-τιμής. Στη διάταξή μας, κάθε ζεύγος κλειδιού-τιμής αντιπροσωπεύεται από μια δομή που περιέχει δύο πίνακες χαρακτήρων σταθερών μεγεθών: έναν για το κλειδί και έναν για την τιμή. Για να διαχειριστούμε αποτελεσματικά αυτές τις δομές, έχουμε επίσης ένα πίνακα για να λειτουργήσουμε ως το store. Αυτός ο πίνακας είναι γεμάτος με δείκτες στις δομές κλειδιού-τιμής.

4. **Λειτουργίες:** Υλοποιούμε τέσσερις κύριες λειτουργίες:

Εισαγωγή: Αυτή η λειτουργία περιλαμβάνει τη δημιουργία μιας νέας δομής κλειδιού-τιμής, την αντιγραφή του παρεχόμενου κλειδιού και τιμής στους πίνακες της νέας δομής και την προσθήκη ενός δείκτη σε αυτήν τη νέα δομή στον πίνακα του store. Η

προστασία μνήμης απενεργοποιείται προσωρινά για τη λειτουργία εισαγωγής και, στη συνέχεια, ενεργοποιείται ξανά.

Διαγραφή: Βρίσκουμε τη δομή κλειδιού-τιμής που πρέπει να διαγραφεί από τον πίνακα του store χρησιμοποιώντας το παρεχόμενο κλειδί και, στη συνέχεια, αφαιρούμε τον δείκτη σε αυτήν τη δομή από τον πίνακα. Και πάλι, απενεργοποιούμε προσωρινά και, στη συνέχεια, ενεργοποιούμε ξανά την προστασία μνήμης.

Ενημέρωση: Για την ενημέρωση μιας τιμής, βρίσκουμε (με μια πολύ βασική μέθοδο αναζήτησης) τη δομή κλειδιού-τιμής με το παρεχόμενο κλειδί και, στη συνέχεια, ενημερώνουμε την τιμή σε αυτήν τη δομή. Ο χειρισμός της προστασίας μνήμης γίνεται με τον ίδιο τρόπο όπως και για τις λειτουργίες εισαγωγής και διαγραφής.

Αναζήτηση: Αυτή η λειτουργία περιλαμβάνει απλώς σάρωση μέσω του πίνακα store για να βρούμε τη δομή κλειδιού-τιμής με το απαιτούμενο κλειδί και, στη συνέχεια, να επιστρέψουμε την αντίστοιχη τιμή.

Απελευθέρωση μνήμης: Τέλος, έχουμε τον κωδικό για το κλείσιμο του store κλειδιού-τιμής και την απελευθέρωση των πόρων μας. Αυτό περιλαμβάνει τη χρήση της λειτουργίας «pmemobj_close» του PMDK για να κλείσουμε το pool μνήμης μας και την κλήση συστήματος «pkey_free» για την απελευθέρωση του κλειδιού προστασίας μας.

Αυτή είναι μια απλοποιημένη έκδοση της περιγραφής του κώδικα.

Περισσότερα η εφαρμογή θα μπορούσε να περιλαμβάνει πρόσθετο έλεγχο σφαλμάτων, συνθήκες ορίων και διαδικασίες επικύρωσης δεδομένων για τη διασφάλιση ισχυρών και ασφαλών λειτουργιών.

5.3 Παρατηρήσεις-Γραφήματα

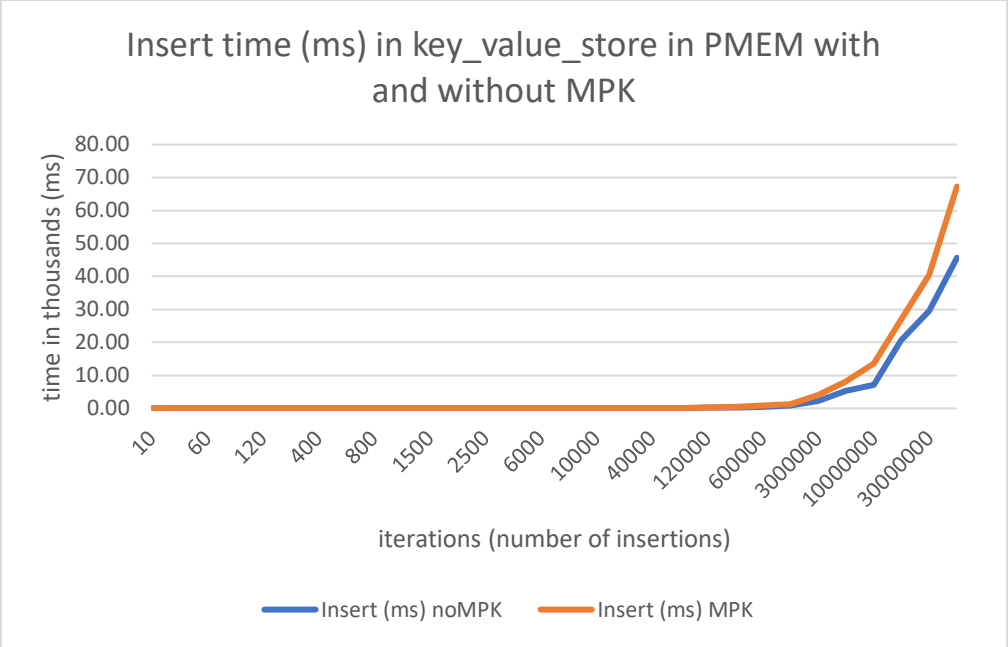

Figure 18Εισαγωγή κλειδιού σε PMEM με και χωρίς τον μηχανισμό MPK

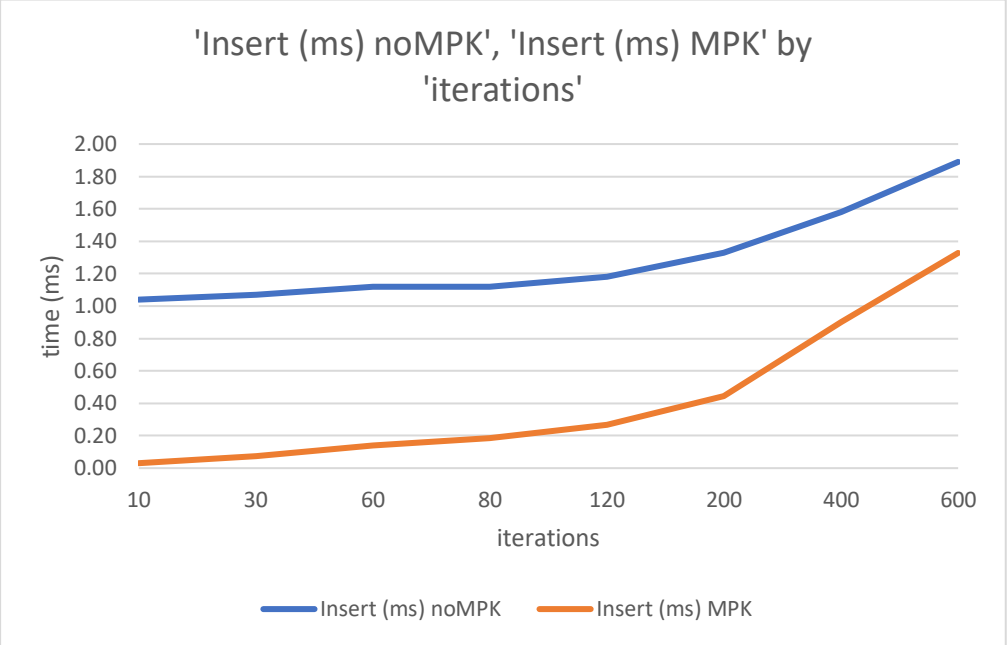

Figure 19 Εισαγωγή κλειδιού σε PMEM με και χωρίς τον μηχανισμό MPK για μικρό αριθμό επαναλήψεων

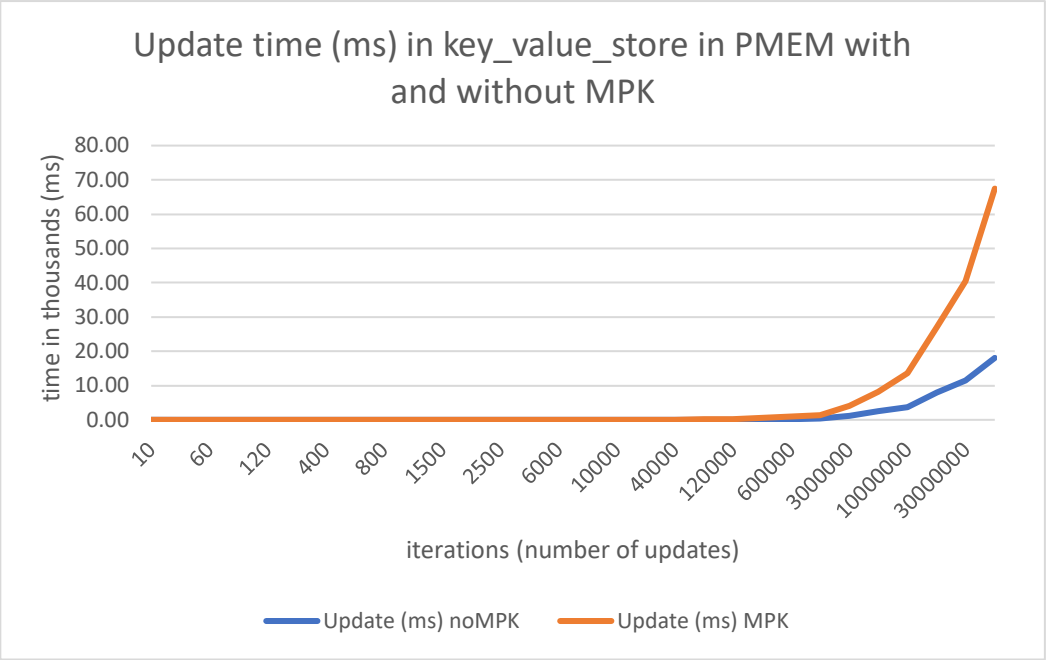

Figure 20 Ενημέρωση κλειδιού σε PMEM με και χωρίς τον μηχανισμό MPK

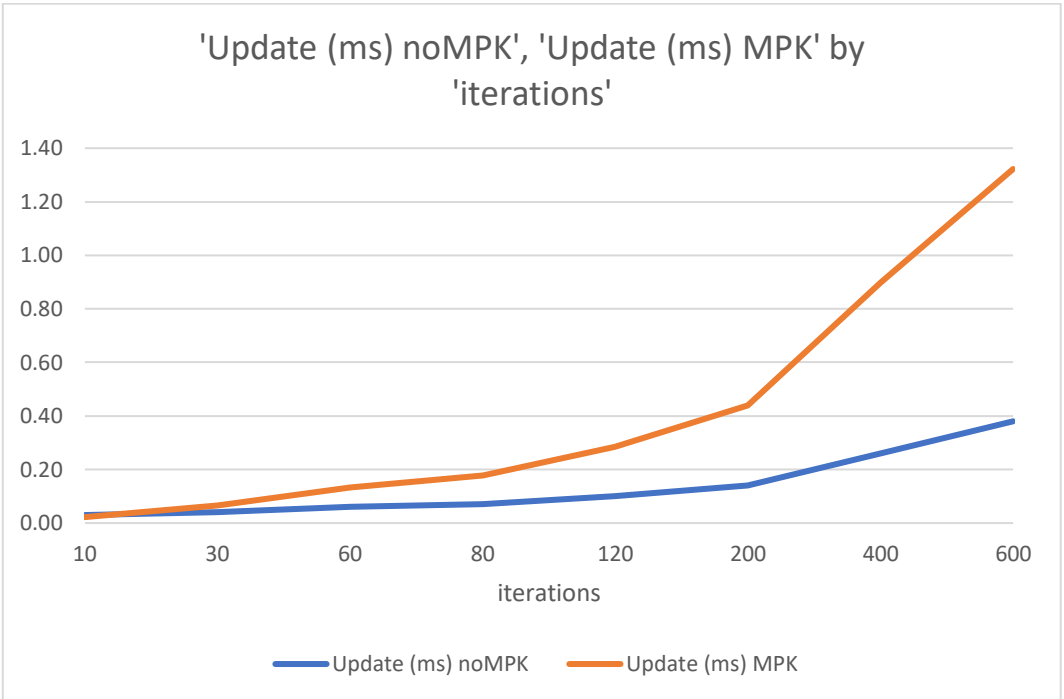

Figure 21 Ενημέρωση κλειδιού σε PMEM με και χωρίς τον μηχανισμό MPK για μικρό αριθμό επαναλήψεων

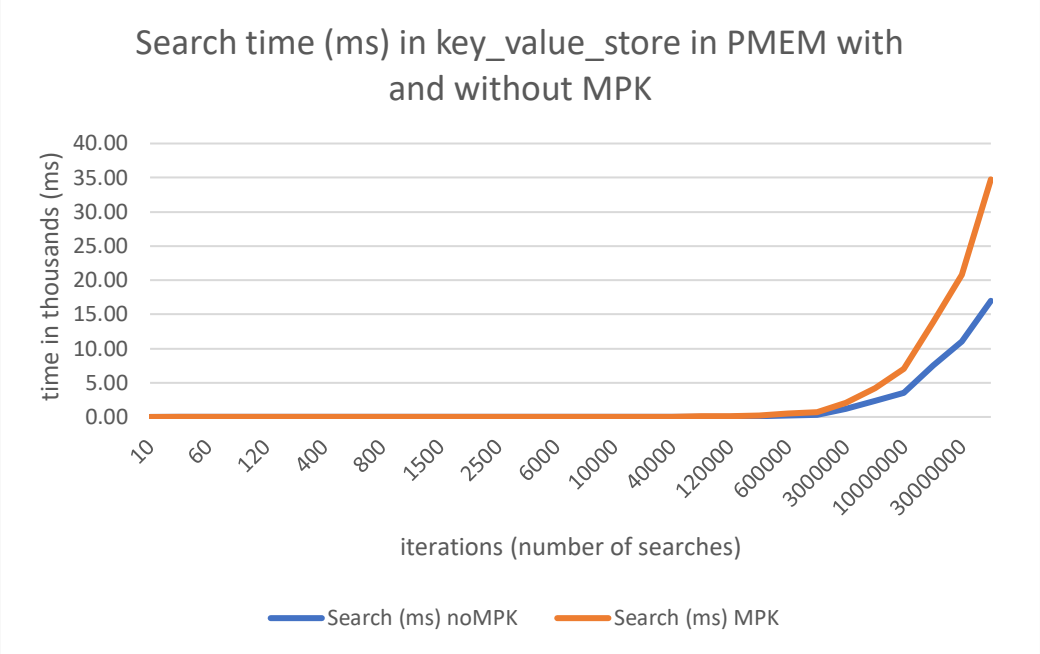

Figure 22 Αναζήτηση κλειδιού σε PMEM με και χωρίς τον μηχανισμό MPK

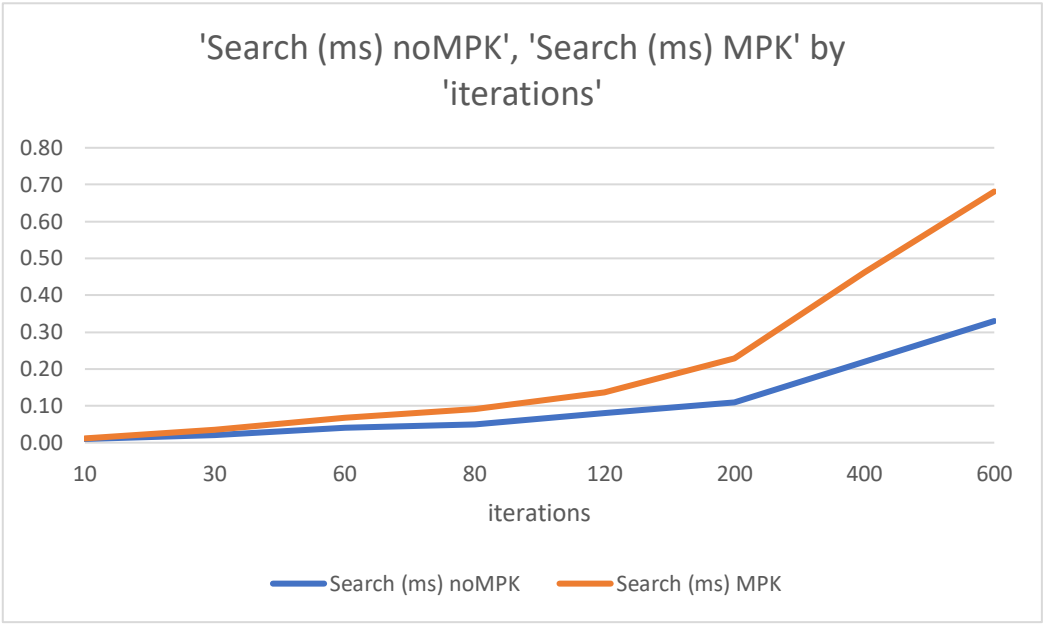

Figure 23 Αναζήτηση κλειδιού σε PMEM με και χωρίς τον μηχανισμό MPK για μικρό αριθμό επαναλήψεων

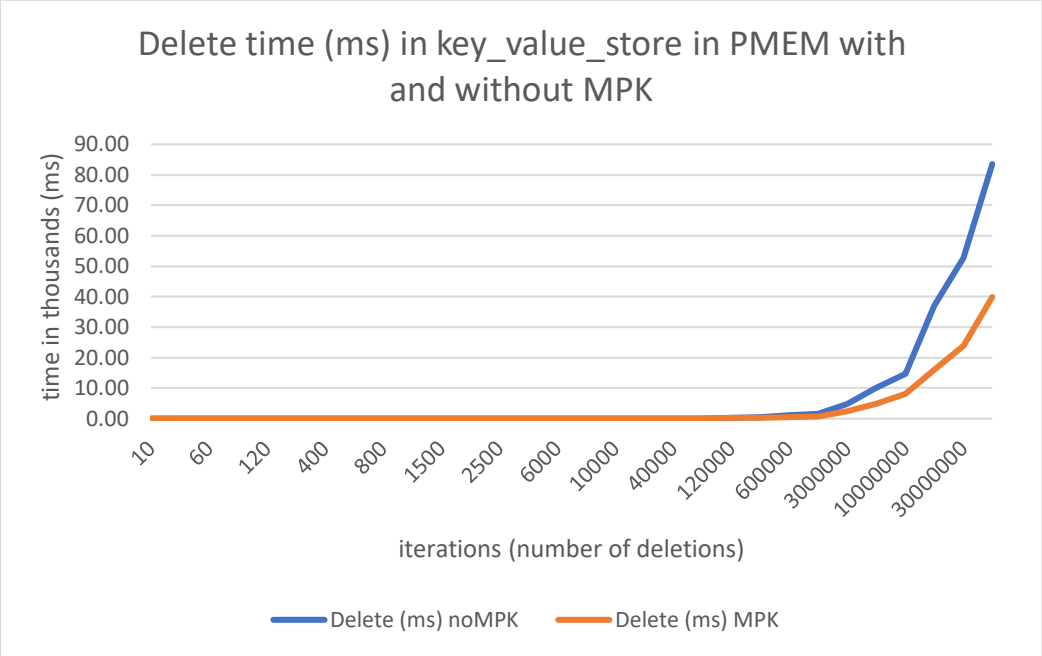

Figure 24 Διαγραφή κλειδιού σε PMEM με και χωρίς τον μηχανισμό MPK

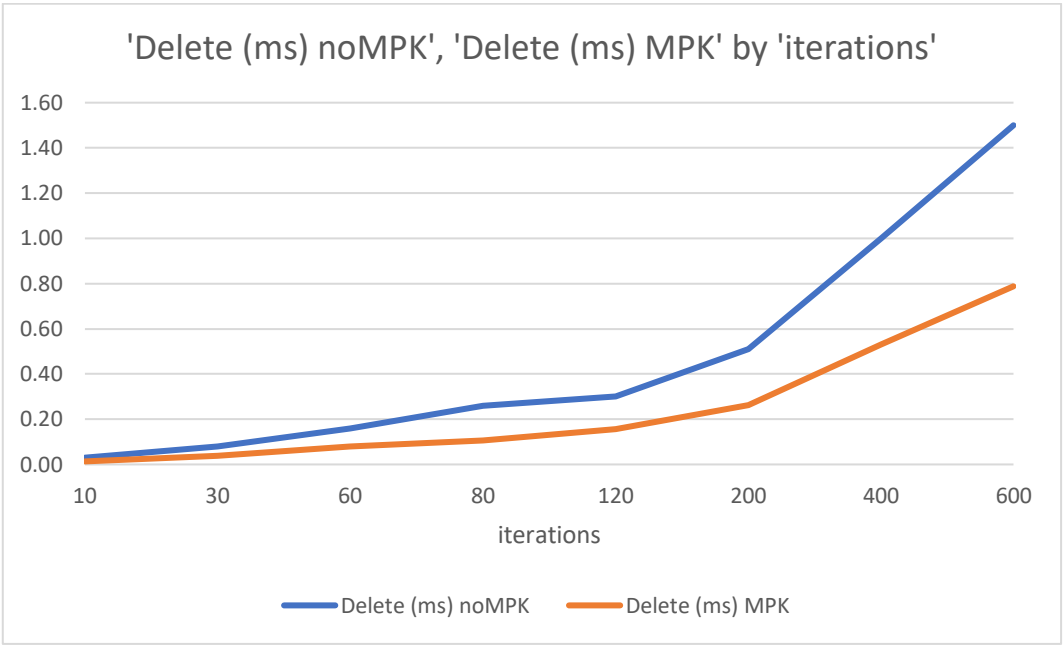

Figure 25 Διαγραφή κλειδιού σε PMEM με και χωρίς τον μηχανισμό MPK για μικρό αριθμό επαναλήψεων

Παραπάνω παραθέτουμε τα γραφήματα χρόνου απόδοσης για διαφορετικές επαναλήψεις εισαγωγής, αφαίρεσης, ενημέρωσης, αναζήτησης και διαγραφής ζεύγους τιμής-κλειδιού.

Αντίκτυπος της προστασίας στην απόδοση: Και στις τέσσερις λειτουργίες, η εφαρμογή προστασίας αυξάνει σημαντικά τον χρόνο που απαιτείται, ειδικά καθώς αυξάνεται το μέγεθος ή η πολυπλοκότητα των λειτουργιών. Αυτό υποδηλώνει ότι ενώ τα κλειδιά προστασίας μνήμης μπορούν να παρέχουν κρίσιμα οφέλη ασφαλείας, εισάγουν μια γενική επιβάρυνση απόδοσης.

Κλιμάκωση του αντίκτυπου απόδοσης: Η διαφορά μεταξύ προστατευμένων και μη προστατευμένων χρόνων λειτουργίας φαίνεται να αυξάνεται με μη γραμμικό τρόπο. Δηλαδή, καθώς οι λειτουργίες γίνονται πιο περίπλοκες ή μεγαλύτερες σε κλίμακα, τα γενικά έξοδα προστασίας αυξάνονται πιο γρήγορα. Αυτό είναι ιδιαίτερα εμφανές σε λειτουργίες όπως η Εισαγωγή και η Ενημέρωση, όπου για μεγαλύτερες λειτουργίες, ο χρόνος που απαιτείται με προστασία υψηλότερος από τον χρόνο που απαιτείται χωρίς προστασία.

Σχετική απόδοση λειτουργιών: Ανεξάρτητα από το αν χρησιμοποιείται προστασία, η λειτουργία διαγραφής διαρκεί σταθερά περισσότερο χρόνο από άλλες λειτουργίες (Εισαγωγή, Ενημέρωση, Αναζήτηση). Αυτό θα μπορούσε ενδεχομένως να υποδηλώνει ότι αυτή η λειτουργία έχει εγγενή πολυπλοκότητα ή απαιτεί περισσότερη υπολογιστική ισχύ.

Security vs Performance Trade-off: Τα δεδομένα αντικατοπτρίζουν έναν κλασικό συμβιβασμό μεταξύ ασφάλειας και απόδοσης. Ενώ η χρήση κλειδιών προστασίας ενισχύει σημαντικά την ασφάλεια αποτρέποντας τη μη εξουσιοδοτημένη πρόσβαση στη μνήμη, επιβάλλει επίσης κόστος απόδοσης. Αυτός ο συμβιβασμός πρέπει να εξεταστεί προσεκτικά με βάση τις ειδικές απαιτήσεις της αίτησης.Τονίζεται ότι ο κώδικας μας δεν είναι τόσο αποδοτικά σχεδιασμένος και αυτό επηρεάζει τα αποτελέσματα της αξιολόγησης.

5.4 Συμπεράσματα:

Η ολοκληρωμένη υλοποίηση μας, παρουσιάζει μια λύση στους περιορισμούς που είναι εμφανείς στα παραδοσιακά μοντέλα μνήμης. Έχουμε δημιουργήσει ένα αποτελεσματικό, ασφαλές και αξιόπιστο σύστημα αποθήκευσης κλειδιού-τιμής που συνδυάζει τα πλεονεκτήματα της μόνιμης μνήμης (PMEM) και των κλειδιών προστασίας μνήμης (MPK). Αυτή η προσέγγιση συνδυάζει ουσιαστικά τη μη πτητική φύση του PMEM με τις ενισχυμένες διατάξεις ασφαλείας του MPK, ξεπερνώντας τα γενικά έξοδα απόδοσης, την αναξιοπιστία και τα τρωτά σημεία που συχνά συνδέονται με τα παραδοσιακά συστήματα. Η άμεση πρόσβαση που επιτρέπει το PMEM στις load και store εντολές μειώνει σημαντικά το latency και βελτιώνει την απόδοση. Αυτή η άμεση και αποτελεσματική πρόσβαση στα δεδομένα είναι ιδιαίτερα επωφελής για το σύστημα αποθήκευσης κλειδιού-τιμής, όπου υπάρχει συχνή ανάκτηση και ενημέρωση δεδομένων.

Το σύστημά μας παρέχει θεμελιώδεις λειτουργίες, όπως εισαγωγή, διαγραφή, ενημέρωση και αναζήτηση, στο χώρο αποθήκευσης κλειδιού-τιμής, αντικατοπτρίζοντας αποτελεσματικά τη λειτουργικότητα που αναμένει κανείς από αυτό το είδος δομής δεδομένων. Οι μηχανισμοί προστασίας του MPK εφαρμόζονται με σύνεση κατά τη διάρκεια καθεμιάς από αυτές τις λειτουργίες, διασφαλίζοντας την ακεραιότητα και την ασφάλεια των δεδομένων καθ 'όλη τη διάρκεια του κύκλου ζωής τους (χρίζει περαιτέρω αξιολόγησης).

Παρά την τρέχουσα επιτυχή υλοποίηση, υπάρχει περιθώριο για μελλοντική εξερεύνηση. Η βελτιστοποίηση αυτού του μοντέλου για το χειρισμό μεγαλύτερων μεγεθών δεδομένων, η περαιτέρω βελτίωση της απόδοσης και η διερεύνηση πολύπλοκων δομών δεδομένων θα είναι ενδιαφέρουσα. Αυτή η δυνατότητα αποτελεί πράγματι ένα σημαντικό βήμα προόδου στον τομέα, ενθαρρύνοντας περαιτέρω την έρευνα και την καινοτομία σε ανθεκτικές δομές δεδομένων και τεχνολογίες προστασίας μνήμης.

Κεφάλαιο 6

Μελλοντική δουλειά

6.1 Μελλοντική δουλειά και βελτιώσεις

Περαιτέρω ανάλυση των τεχνολογιών προστασίας μνήμης: Καθώς το πρόσφατο πείραμα εξέτασε την απόδοση του mprotect και του pkey_mprotect όσον αφορά την ακρίβεια της μνήμης, το χειρισμό του χρόνου και του χώρου, υπάρχει περιθώριο επέκτασης αυτής της πειραματικής αξιολόγησης για να συμπεριλάβει και άλλες τεχνολογίες προστασίας μνήμης. Αυτό θα παρείχε μια πιο ολοκληρωμένη κατανόηση του τοπίου προστασίας της μνήμης.

Αντίκτυπος του PMDK σε άλλα συστήματα: Ενώ έχουμε μιλήσει για το ρόλο του PMDK στη διαχείριση της μόνιμης μνήμης στο υλικό της Intel, θα ήταν ενδιαφέρον να διερευνήσουμε πώς το PMDK θα μπορούσε να ενσωματωθεί σε συστήματα που δεν ανήκουν στην Intel. Καθώς το PMDK είναι ουδέτερο ως προς τον προμηθευτή και την πλατφόρμα, πρέπει να αναλυθεί ο αντίκτυπος και η απόδοσή του σε διάφορες πλατφόρμες.

Βελτιστοποίηση της προστασίας μνήμης: Υπάρχει περιθώριο για εμβάθυνση στη δημιουργία και τη δοκιμή τεχνικών βελτιστοποίησης για την προστασία της μνήμης με βάση τα ευρήματα του προαναφερθέντος πειράματος που συγκρίνει το mprotect και το pkey_mprotect. Επιπλέον θα μπορούσαμε να παραβιάσουμε την μνήμη για να μελετήσουμε την ασφάλεια του MPK μηχανισμού.

Αξιολόγηση τεχνικών Flushing, Ordering, and Fencing: Σε προηγούμενο κεφάλαιο έγινε αναφορά στη σημασία αυτών των τεχνικών στην τεχνολογία επίμονης μνήμης της Intel. Περισσότερη δουλειά μπορεί να γίνει για την αξιολόγηση αυτών των τεχνικών σε πραγματικές εφαρμογές και για την ανάπτυξη βέλτιστων πρακτικών για τη χρήση τους καθώς και η αποτελεσματικότητα τους σε θέματα ασφάλειας.

Χωρικό κόστος: Μελέτη της σπατάλης χώρου για προστασία μικρού μεγέθους μνήμης και τρόπος για επίλυση του ζητήματος αυτού.

Παραλληλία: Πρέπει να ελεγχθεί κατά πόσο η παραλληλία επηρεάζει την απόδοση και την ασφάλεια του key-value store.

Βιβλιογραφία

[1] “Stray-write protection for persistent memory” Ιστοσελίδα: https://lwn.net/Articles/883352/

[2] “Memory protection keys” Ιστοσελίδα: https://lwn.net/Articles/643797/

[3] “Persistent memory programming by Andy Rudoff” Ιστοσελίδα: https://www.usenix.org/system/files/login/articles/login_summer17_07_rudoff.pdf

[4] “Persistent memory development kit” Ιστοσελίδα: https://github.com/pmem/pmdk

[5] Βιβλίο “Programming Persistent Memory, A Comprehensive Guide for Developers

[6] By Steve Scargall” κεφάλαια 2,3,5,6,9,19.

[7] “What is Optane technology by Intel ” Ιστοσελίδα: https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/what-is-optane-technology-brief.pdf

[8] “PKU Pitfalls: Attacks on PKU-based Memory Isolation Systems” Ιστοσελίδα:

[9] https://www.usenix.org/conference/usenixsecurity20/presentation/connor

[10] “Survey of Persistent Memory Security” by Genoveva Fossas Ιστοσελίδα: https://paulgazzillo.com/files/fossas21survey.pdf

[11] “The CHERI capability model: Revisiting RISC in an age of risk” Ιστοσελίδα: https://www.cl.cam.ac.uk/research/security/ctsrd/pdfs/201406-isca2014-cheri.pdf

[12] “Verified security for the Morello capability-enhanced prototype Arm architecture” Ιστοσελίδα: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-959.pdf

[13] “Improved Architectur chitectures for Secur es for Secure Intra-process Isolation” Ιστοσελίδα: https://trace.tennessee.edu/cgi/viewcontent.cgi?article=8014&context=utk_graddiss

[14] “No Need to Hide: Protecting Safe Regions on Commodity Hardware” Ιστοσελίδα: https://scholar.google.com/citations?view_op=view_citation&hl=en&user=RLhxKecAAAAJ&citation_for_view=RLhxKecAAAAJ:O3NaXMp0MMsC

[15] “Hodor: Intra-Process Isolation for High-Throughput Data Plane Libraries”

[16] https://www.usenix.org/conference/atc19/presentation/hedayati-hodor

[17] https://kernelnewbies.org/Linux_4.15

ΠΑΡΑΡΤΗΜΑ Α

1- Κώδικας με χρήση παραδοσιακής προστασίας μνήμης

```
1  #include <stdio.h>
2  #include <sys/mman.h>
3  #include <time.h>
4  #include <stdlib.h>
5  #include <string.h>

6  #define PAGESIZE 4096 // define the page size to be 4096 b
   ytes
7  #define mask ~(PAGESIZE-1)

8  void errExit(const char* t) { // define a function to prin
   t error messages and exit the program
9  printf("\n%s\n", t);
10 exit(-1);
11 }

12 // define a function to get the current time in nanosecond
   s
13 static long get_nanos() {
14 struct timespec ts;
15 timespec_get(&ts, TIME_UTC);
16 return (long)ts.tv_sec * 1000000000L + ts.tv_nsec;
17 }

18 #define memSize 4294967296 // define the size of the buffe
   r to be 4GB
```

```
19 void main(void) {
20 int status, counter = 0; // declare variables for a counte
   r
21 void* buffer, *mprotectAddr; // declare pointers to the bu
   ffer and an address to be mprotected
22 void* tail; // declare a pointer to the end of the buffer
23 long timeBegin, timeEnd; // declare variables for measurin
   g time

24 int cpySize[26] = {3, 13, 50, 100, 512, 1024}; // define a
   n array of sizes for the mprotect operation
25 for (int i = 6; i < 26; i++) {
26 cpySize[i] = cpySize[i - 1] + 4096; // calculate the rest
   of the sizes
27 }

28 buffer = mmap(NULL, memSize, PROT_READ | PROT_WRITE, MAP_P
   RIVATE | MAP_ANONYMOUS, -1, 0); // allocate the buffer
29 if (buffer == MAP_FAILED) {
      a. errExit("mmap\n"); // check for allocation errors
30 }

31 tail = (void*)((unsigned long)buffer + memSize - 6245 - cp
   ySize[25]); // calculate the end of the buffer

32 printf("time(ns)\n");

33 timeBegin = get_nanos(); // get the current time
34 for (int i = 0; i < 2000; i++) { // loop 2000 times
35 counter++; // increment the counter
36 if (i % 2 == 0) {
```

```
37 mprotect(buffer, memSize, PROT_READ); // set the protectio
   n to read-only
38 } else {
39 mprotect(buffer, memSize, PROT_READ | PROT_WRITE); // set
   the protection to read/write
40 }
41 }
42 timeEnd = get_nanos(); // get the current time
43 printf("\tmprotect time %ld, set %d times, average set tim
   e %ld\n\n", timeEnd - timeBegin, counter, (timeEnd - timeB
   egin) / counter); // print the elapsed time, the number of
    times set, and the average time per set

44 // Iterate over 26 possible copy sizes, starting with 3 an
   d increasing by 4096 bytes for each iteration
45 for (int j = 0; j < 26; j++) {
46 int cmpSize = cpySize[j]; // get the current copy size fro
   m the cpySize array
47 counter = 0; // initialize counter variable to 0
48 mprotectAddr = buffer; // set the starting address for mpr
   otect() to the beginning of the buffer
49 timeBegin = get_nanos(); // get the current time in nanose
   conds

50 // Iterate over the buffer with the current copy size, plu
   s an additional 4097 bytes (to avoid overlapping pages)
51 for (; (unsigned long)mprotectAddr <= (unsigned long)tail;
    mprotectAddr = mprotectAddr + cmpSize + 4097) {
52 //unsigned long long mask = ~(PAGESIZE - 1);
53 // call mprotect() on the current memory page with the spe
   cified copy size and read/write access
```

```
54 if (mprotect((void*)((unsigned long long)mprotectAddr & ma
   sk), cmpSize, PROT_READ | PROT_WRITE) == -1) {
55 printf("mprotect error\n"); // print error message if mpro
   tect() fails
56 }
57 counter++; // increment the counter for the number of page
   s that have been protected
58 }

59 timeEnd = get_nanos(); //get the current time in nanosecon
   ds

60 // print the time taken, number of pages protected, averag
   e time per page, and current copy size
61 printf("\tmprotect consume time %10ld, ", timeEnd - timeBe
   gin);
62 printf("mprotect %10d times, ", counter);
63 printf("average set time %6ld, ", (timeEnd - timeBegin) /
   counter);
64 printf("mprotect size %7d\n", cmpSize);
65 }

66 // free the buffer memory
67 //free(buffer);
68 munmap(buffer, memSize);
69 return;
70 }
```

2- Κώδικας για προστασία μνήμης με χρήση MPK:

```
1  #include <stdio.h>
2  #include <sys/mman.h>
3  #include <time.h>
4  #include <stdlib.h>


5  #define PAGESIZE 4096 // define the page size to be 4096 b
   ytes

6  void errExit(const char* t) { // define a function to prin
   t error messages and exit the program
7  printf("\n%s\n", t);
8  exit(-1);
9  }

10 // define a function to get the current time in nanosecond
   s
11 static long get_nanos() {
12 struct timespec ts;
13 timespec_get(&ts, TIME_UTC);
14 return (long)ts.tv_sec * 1000000000L + ts.tv_nsec;
15 }

16 #define memSize 4294967296 // define the size of the buffe
   r to be 4GB


17 /**
18 @brief
19 This program benchmarks the performance of the Linux pkeys
    (protection keys)
20 feature, which allows a process to set and remove access r
   estrictions on
21 specific regions of memory.

22 The program first allocates a large buffer using calloc()
   and then allocates
23 a protection key using pkey_alloc(). It then iterates over
    a series of copy
24 sizes (3, 13, 50, 100, 512, 1024, 4108, 8216, etc.) and ca
   lls pkey_mprotect()
25 on the buffer, protecting each region of memory with the g
   iven copy size.
```

```
26 The program measures the time taken to perform the pkey_mp
   rotect() operation
27 and prints out the time taken and the number of times the
   operation was
28 performed for each copy size.

29 It also measures the time taken to perform pkey_set() and
   prints out the
30 time taken, the number of times the operation was performe
   d, and the average set time.

31 In summary, this program benchmarks the performance of the
    Linux pkeys feature,
32 specifically the pkey_mprotect() and pkey_set() operations
   , for a variety of
33 copy sizes. The output provides information on the time ta
   ken and the number
34 of times these operations were performed, as well as the a
   verage set time.
35 *
36 */
37 void main(void) {
38 int status, pkey, counter = 0; // declare variables for th
   e pkey and a counter
39 void* buffer, *mprotectAddr; // declare pointers to the bu
   ffer and an address to be mprotected
40 void* tail; // declare a pointer to the end of the buffer
41 long timeBegin, timeEnd; // declare variables for measurin
   g time
42 unsigned long long mask; // declare a mask to align addres
   ses

43 int cpySize[26] = {3, 13, 50, 100, 512, 1024}; // define a
   n array of sizes for the mprotect operation
44 for (int i = 6; i < 26; i++) {
45 cpySize[i] = cpySize[i - 1] + 4096; // calculate the rest
   of the sizes
46 }

47 mask = ~(4096 - 1); // calculate the mask to align address
   es
48 buffer = calloc(1, memSize); // allocate the buffer

49 if (buffer < 0)
50 errExit("malloc\n"); // check for allocation errors
```

```
51 pkey = pkey_alloc(0, PKEY_DISABLE_ACCESS); // allocate a p
   rotection key

52 if (pkey == -1)
53 errExit("pkey_alloc"); // check for errors in pkey allocat
   ion

54 tail = (void*)((unsigned long)buffer + memSize - 6245 - cp
   ySize[25]); // calculate the end of the buffer

55 printf("time(ns)\n");

56 timeBegin = get_nanos(); // get the current time

57 /**
58 @brief The purpose of this code is likely to test the beha
   vior of some
59 code or system under different protection key settings. By
    iterating
60 through the loop and changing the protection key setting o
   n each iteration,
61 the code can simulate different scenarios and observe how
   the system
62 behaves in each case. The variable "counter" may be used t
   o track the
63 number of iterations or to store some other kind of inform
   ation related
64 to the test.
65 *
66 */
67 for (int i = 0; i < 2000; i++) { // loop 2000 times
68 counter++; // increment the counter
69 if (i % 2 == 0) {
70 pkey_set(pkey, 0); // set the protection key to 0
71 } else {
72 pkey_set(pkey, PKEY_DISABLE_ACCESS); // set the protection
    key to PKEY_DISABLE_ACCESS
73 }
74 }

75 timeEnd = get_nanos(); // get the current time
76 printf("\tpkey_set time %ld, set %d times, average set tim
   e %ld\n\n", timeEnd - timeBegin, counter, (timeEnd - timeB
   egin) / counter); // print the elapsed time, the number of
    times set, and the average time per set

77 // Iterate over 26 possible copy sizes, starting with 3 an
   d increasing by 4096 bytes for each iteration
```

```
78 for(int j=0;j<26;j++){

79 int cmpSize = cpySize[j]; // get the current copy size fro
   m the cpySize array
80 counter = 0; // initialize counter variable to 0
81 mprotectAddr = buffer; // set the starting address for mpr
   otect() to the beginning of the buffer
82 timeBegin = get_nanos(); // get the current time in nanose
   conds

83 // Iterate over the buffer with the current copy size, plu
   s an additional 4097 bytes (to avoid overlapping pages)
84 for(;(unsigned long)mprotectAddr<=(unsigned long)tail;
85 mprotectAddr = mprotectAddr+cmpSize+4097){
86 // call pkey_mprotect() on the current memory page with th
   e specified copy size and read/write access
87 if( pkey_mprotect((void*)((unsigned long long)mprotectAddr
   &mask), cmpSize,
88 PROT_READ | PROT_WRITE, pkey)){
89 printf("pkey_mprotect error\n"); // print error message if
    pkey_mprotect() fails
90 }
91 counter++; // increment the counter for the number of page
   s that have been protected
92 }
93 timeEnd = get_nanos(); // get the current time in nanoseco
   nds
94 // print the time taken, number of pages protected, averag
   e time per page, and current copy size
95 printf("\tpkey_mprotect consume time %10ld, ", timeEnd-
   timeBegin);
96 printf("pkey_mprotect %10d times, ", counter);
97 printf("average set time %6ld, ", (timeEnd-
   timeBegin)/counter);
98 printf("pkey_mprotect size %7d\n", cmpSize);
99 }

100      // free the protection key
101      status = pkey_free(pkey);
102      if (status == -1)
103      errExit("pkey_free");

104      return ;
```

3- Dummy Mprotect

```c
#define _GNU_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <time.h>
#include <unistd.h>

#define PAGESIZE 4096
#define MEMSIZE 4294967296

void errExit(const char *msg) {
printf("\n%s\n", msg);
exit(EXIT_FAILURE);
}

static long get_nanos() {
struct timespec ts;
timespec_get(&ts, TIME_UTC);
return (long)ts.tv_sec * 1000000000L + ts.tv_nsec;
}

int main(void) {
int status, counter = 0;
void *buffer, *mprotect_addr;
void *tail;
unsigned long long mask;

int cpy_size[26] = {3, 13, 50, 100, 512, 1024};
for (int i = 6; i < 26; i++) {
cpy_size[i] = cpy_size[i - 1] + PAGESIZE;
}
mask = ~(4096 - 1);

buffer = mmap(NULL, MEMSIZE, PROT_READ | PROT_WRITE,
MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
if (buffer == MAP_FAILED)
errExit("mmap");

tail = (void *)((unsigned long)buffer + MEMSIZE - 6245 - cpy_size[25]);

printf("time(ns)\n");

for (int j = 0; j < 26; j++) {
```

```
36 int cmp_size = cpy_size[j];
37 counter = 0;
38 mprotect_addr = buffer;
39 for (; (unsigned long)mprotect_addr <= (unsigned long)tail;
40 mprotect_addr = mprotect_addr + cmp_size + PAGESIZE) {
41 if (mprotect((void *)((unsigned long long)mprotect_addr & mask), cm
   p_size,
42 PROT_READ | PROT_WRITE)) {
43 printf("mprotect error\n");
44 }
45 counter++;
46 }

47 // Unmap the buffer to release the virtual memory
48 munmap(buffer, MEMSIZE);

49 // Remap the buffer to start over
50 buffer = mmap(NULL, MEMSIZE, PROT_READ | PROT_WRITE,
51 MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
52 if (buffer == MAP_FAILED)
53 errExit("mmap");

54 // Get the new tail
55 tail = (void *)((unsigned long)buffer + MEMSIZE - 6245 - cpy_size[2
   5]);

56 printf("\tmprotect consume time %10ld, ", get_nanos());
57 printf("mprotect %10d times, ", counter);
58 printf("mprotect size %7d\n", cmp_size);
59 }

60 status = munmap(buffer, MEMSIZE);
61 if (status == -1)
62 errExit("munmap");

63 return EXIT_SUCCESS;
64 }
```

4- Κώδικα Key-value store σε persistent memory

```c
#define _GNU_SOURCE
#define _POSIX_C_SOURCE 200809L

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <libpmemobj.h>
#include <time.h>
#include <sys/mman.h>
#include <sys/resource.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/param.h>
#include <errno.h>

#define MAX_KEYS 100
#define KEY_SIZE 64
#define VALUE_SIZE 64
//#define ITERATIONS 800
#define LAYOUT_NAME "key_value_store"

typedef struct key_value_pair
{
char key[KEY_SIZE];
char value[VALUE_SIZE];
} key_value_pair_t;

typedef struct root
{
key_value_pair_t kvs[MAX_KEYS];
} root_t;

// Memory protection keys
int pkey;

// Function to allocate and protect memory
PMEMoid alloc_protected_memory(PMEMobjpool *pop, size_t size)
{
PMEMoid oid;

// Get the system's page size
long page_size = sysconf(_SC_PAGESIZE);

// Round up the size to the nearest multiple of the page size
```

```
37 size_t aligned_size = (size + page_size - 1) & ~(page_size - 1);

38 int ret = pmemobj_zalloc(pop, &oid, aligned_size, 0);
39 if (ret)
40 {
41 perror("pmemobj_zalloc");
42 return OID_NULL;
43 }

44 void *aligned_addr;
45 ret = posix_memalign(&aligned_addr, page_size, aligned_size);
46 if (ret)
47 {
48 perror("posix_memalign");
49 pmemobj_free(&oid);
50 return OID_NULL;
51 }

52 pkey = pkey_alloc(0, 0);
53 if (pkey == -1)
54 {
55 perror("pkey_alloc");
56 free(aligned_addr);
57 pmemobj_free(&oid);
58 return OID_NULL;
59 }

60 if (pkey_mprotect(aligned_addr, size, PROT_READ | PROT_WRITE, pkey)
   == -1)
61 {
62 perror("pkey_mprotect");
63 free(aligned_addr);
64 pmemobj_free(&oid);
65 return OID_NULL;
66 }

67 void *pmem_addr = pmemobj_direct(oid);
68 pmemobj_memcpy_persist(pop, pmem_addr, aligned_addr, aligned_size);

69 free(aligned_addr);

70 return oid;
71 }


72 // Function to deallocate and unprotect memory
73 void free_protected_memory(PMEMoid oid)
74 {
```

```
75 pkey_free(pkey);

76 pmemobj_free(&oid);
77 }


78 POBJ_LAYOUT_BEGIN(kv_store);
79 POBJ_LAYOUT_END(kv_store);

80 void insert_key_value(PMEMobjpool *pop, root_t *root_obj,
   key_value_pair_t *kvp)
81 {
82 // Unprotect the root memory before accessing it
83 if (pkey_mprotect(root_obj, sizeof(root_t), PROT_READ | PROT_WRITE,
   pkey) == -1)
84 {
85 perror("pkey_mprotect");
86 return;
87 }

88 for (int i = 0; i < MAX_KEYS; i++)
89 {
90 if (strlen(root_obj->kvs[i].key) == 0) // Check if the key string
   is empty
91 {
92 // Safely copy key and value strings using their lengths
93 size_t key_len = strlen(kvp->key) + 1;
94 size_t value_len = strlen(kvp->value) + 1;
95 strncpy(root_obj->kvs[i].key, kvp->key, key_len < KEY_SIZE - 1 ?
   key_len : KEY_SIZE - 1);
96 strncpy(root_obj->kvs[i].value, kvp->value, value_len < VALUE_SIZE
   - 1 ? value_len : VALUE_SIZE - 1);
97 break;
98 }
99 }

100       // Protect the root memory after the operation
101       if (pkey_mprotect(root_obj, sizeof(root_t), PROT_NONE, pkey)
   == -1)
102       {
103       perror("pkey_mprotect");
104       return;
105       }
106       }


107       const char *search_key(PMEMobjpool *pop, root_t *root_obj,
   const char *key)
```

```
108       {

109       // Unprotect the root memory before accessing it
110       if (pkey_mprotect(root_obj, sizeof(root_t), PROT_READ |
    PROT_WRITE, pkey) == -1)
111       {
112       perror("pkey_mprotect");
113       return;
114       }
115       for (int i = 0; i < MAX_KEYS; i++)
116       {
117       if (strlen(root_obj->kvs[i].key) != 0)
118       {
119       if (strcmp(root_obj->kvs[i].key, key) == 0)
120       {
121       return root_obj->kvs[i].value;
122       }
123       }
124       }

125       // Protect the root memory after the operation
126       if (pkey_mprotect(root_obj, sizeof(root_t), PROT_NONE, pkey)
    == -1)
127       {
128       perror("pkey_mprotect");
129       return;
130       }

131       return NULL;
132       }
133

134       void delete_key_value(PMEMobjpool *pop, root_t *root_obj,
    const char *key)
135       {

136       // Unprotect the root memory before accessing it
137       if (pkey_mprotect(root_obj, sizeof(root_t), PROT_READ |
    PROT_WRITE, pkey) == -1)
138       {
139       perror("pkey_mprotect");
140       return;
141       }



142       for (int i = 0; i < MAX_KEYS; i++)
143       {
144       if (strlen(root_obj->kvs[i].key) != 0)
```

```
145       {
146       if (strcmp(root_obj->kvs[i].key, key) == 0)
147       {
148       memset(&root_obj->kvs[i], 0, sizeof(key_value_pair_t)); //
   Reset the key-value pair
149       break;
150       }
151       }
152       }

153       // Protect the root memory after the operation
154       if (pkey_mprotect(root_obj, sizeof(root_t), PROT_NONE, pkey)
   == -1)
155       {
156       perror("pkey_mprotect");
157       return;
158       }
159       }

160       void update_key_value(PMEMobjpool *pop, root_t *root_obj,
   key_value_pair_t *kvp)
161       {
162       // Unprotect the root memory before accessing it
163       if (pkey_mprotect(root_obj, sizeof(root_t), PROT_READ |
   PROT_WRITE, pkey) == -1)
164       {
165       perror("pkey_mprotect");
166       return;
167       }

168       for (int i = 0; i < MAX_KEYS; i++)
169       {
170       if (strcmp(root_obj->kvs[i].key, kvp->key) == 0)
171       {
172       // Safely copy value string using its length
173       size_t value_len = strlen(kvp->value) + 1;
174       strncpy(root_obj->kvs[i].value, kvp->value, value_len <
   VALUE_SIZE - 1 ? value_len : VALUE_SIZE - 1);
175       break;
176       }
177       }

178       // Protect the root memory after the operation
179       if (pkey_mprotect(root_obj, sizeof(root_t), PROT_NONE, pkey)
   == -1)
180       {
181       perror("pkey_mprotect");
182       return;
```

```
183       }
184       }
185
186       int main(int argc, char *argv[])
187       {
188       if (argc != 3)
189       {
190       printf("Usage: %s <path_to_pool>\n", argv[0]);
191       return 1;
192       }
193       const char *pool_path = argv[1];
194       PMEMobjpool *pop = pmemobj_open(pool_path, LAYOUT_NAME);
195       if (pop == NULL)
196       {
197       perror("pmemobj_open");
198       exit(1);
199       }
200       const int iterations = atoi(argv[2]);
201       // Get the system's page size
202       long page_size = sysconf(_SC_PAGESIZE);
203       // Create the root object with an extra page size
204       PMEMoid root_oid = pmemobj_root(pop, sizeof(root_t)); //for
   memory alignment
205       if (OID_IS_NULL(root_oid))
206       {
207       perror("pmemobj_root");
208       pmemobj_close(pop);
209       return 1;
210       }
211       // Round up the root object address to the next page boundary
212       root_t *root_obj = (root_t
   *)(((uintptr_t)pmemobj_direct(root_oid) + page_size - 1) &
   ~(page_size - 1));
213       // Apply memory protection
214       int pkey = pkey_alloc(0, PKEY_DISABLE_ACCESS);
215       if (pkey == -1)
216       {
217       perror("pkey_alloc");
218       pmemobj_close(pop);
219       return 1;
```

```
220       }

221       if (pkey_mprotect(root_obj, sizeof(root_t), PROT_READ |
   PROT_WRITE, pkey) == -1)
222       {
223       perror("pkey_mprotect");
224       pmemobj_close(pop);
225       return 1;
226       }

227       struct timespec start, end;
228       double insert_time = 0, update_time = 0, delete_time = 0,
   search_time = 0;

229       PMEMoid kvp_oid = alloc_protected_memory(pop,
   sizeof(key_value_pair_t));

230       if (OID_IS_NULL(kvp_oid))
231       {
232       perror("alloc_protected_memory");
233       return 1;
234       }

235       key_value_pair_t *kvp_ptr = (key_value_pair_t
   *)pmemobj_direct(kvp_oid);

236       char key[KEY_SIZE], value[VALUE_SIZE];

237       for (int i = 0; i < iterations; i++)
238       {
239       snprintf(key, KEY_SIZE, "key_%d", i);
240       snprintf(value, VALUE_SIZE, "value_%d", i);

241       strncpy(kvp_ptr->key, key, KEY_SIZE);
242       strncpy(kvp_ptr->value, value, VALUE_SIZE);

243       // Insert
244       clock_gettime(CLOCK_MONOTONIC, &start);
245       insert_key_value(pop, root_obj, kvp_ptr);
246       clock_gettime(CLOCK_MONOTONIC, &end);
247       insert_time += (end.tv_sec - start.tv_sec) * 1e9 +
   (end.tv_nsec - start.tv_nsec);

248       // Update
249       snprintf(value, VALUE_SIZE, "updated_value_%d", i);
250       strncpy(kvp_ptr->value, value, VALUE_SIZE);
251       clock_gettime(CLOCK_MONOTONIC, &start);
252       update_key_value(root_obj, root_obj,kvp_ptr);
```

```
253       clock_gettime(CLOCK_MONOTONIC, &end);
254       update_time += (end.tv_sec - start.tv_sec) * 1e9 +
   (end.tv_nsec - start.tv_nsec);

255       // Search
256       clock_gettime(CLOCK_MONOTONIC, &start);
257       search_key(pop, root_obj, key);
258       clock_gettime(CLOCK_MONOTONIC, &end);
259       search_time += (end.tv_sec - start.tv_sec) * 1e9 +
   (end.tv_nsec - start.tv_nsec);

260       // Delete
261       clock_gettime(CLOCK_MONOTONIC, &start);
262       delete_key_value(pop, root_obj, key);
263       clock_gettime(CLOCK_MONOTONIC, &end);
264       delete_time += (end.tv_sec - start.tv_sec) * 1e9 +
   (end.tv_nsec - start.tv_nsec);
265       }

266       printf("Total time for %d iterations:\n", iterations);
267       printf("Insert time: %lf ms\n", insert_time / 1e6);
268       printf("Update time: %lf ms\n", update_time / 1e6);
269       printf("Search time: %lf ms\n", search_time / 1e6);
270       printf("Delete time: %lf ms\n", delete_time / 1e6);

271       free_protected_memory(kvp_oid);

272       pmemobj_close(pop);

273       return 0;
274       }
```