Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Προσαρμογή του Rumprun Unikernel για φιλοξενία στο Firecracker Virtual Machine Monitor

Διπλωματικη Εργασια

του

ΓΕΩΡΓΙΟΥ ΓΚΑΝΑ

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εργαστηριο Υπολογιστικων Συστηματων
Αθήνα, Οκτώβριος 2023

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

Προσαρμογή του Rumprun Unikernel για φιλοξενία στο Firecracker Virtual Machine Monitor

Διπλωματικη Εργασια

του

ΓΕΩΡΓΙΟΥ ΓΚΑΝΑ

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 20η Οκτωβρίου 2023.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Νεκτάριος Κοζύρης	Διονύσιος Πνευματικάτος	Γεώργιος Γκούμας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αναπληρωτής
		Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2023

(Υπογραφή)

.....

Γεωργιος Γκανας

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Ο σκοπός της διπλωματικής ήταν η τροποποίηση του Rumprun unikernel ώστε να μπορεί να εκκινήσει και να εκτελέσει τη λειτουργία του σε εικονικοποιημένο περιβάλλον με χρήση του Firecracker ως επόπτη εικονικής μηχανής.

Το Rumprun είναι υλοποιημένο για φιλοξενία στο QEMU ή στο Xen, αλλά επεκτείνοντας τη λογική ελαχιστοποίησης των εξαρτήσεων των unikernel στον υπερεπόπτη, θέλουμε να εκτελέσουμε το Rumprun ως microVM και για αυτό χρησιμοποιούμε το Firecracker. Με αυτόν τον τρόπο, επιθυμούμε να μειώσουμε περαιτέρω χαρακτηριστικά όπως τον χρόνο εκκίνησης, τη χρησιμοποίηση μνήμης και την επιφάνεια επίθεσης, τα οποία αποτελούν τα πλεονεκτήματα των unikernel.

Έτσι σε δεύτερη φάση, έγιναν μετρήσεις για την σύγκριση της επίδοσης του Rumprun όταν χρησιμοποιείται πάνω σε διαφορετικούς υπερεπόπτες. Επιπλέον, για αναφορά έγινε σύγκριση με άλλα unikernel.

Λέξεις Κλειδιά

Ενιαίος Χώρος Διευθύνσεων, Τεχνολογία Νέφους, Εικονικοποίηση, Εικονικές Μηχανές, Λειτουργικά Συστήματα, Διαχειριστής Εικονικής Μηχανής, Υπερεπόπτης.

Abstract

The purpose of this thesis was the modification of the Rumprun unikernel in order for it to boot and run in a virtualised environment using Firecracker as the virtual machine monitor.

Rumprun is implemeneted for hosting on QEMU or Xen, but extending the principle of dependency minimization of unikernels to the hypervisor, we wish to run Rumprun as a microVM and so we use Firecracker. In this way, we aim to further decrease attributes such as boot time, memory utilisation and attack surface, which are the advantages of unikernels.

Thus secondarily, measurements were carried out for the comparison of Rumprun's performance when it's used on top of different hypervisors. Furthermore, for reference the results were compared with those of different unikernels.

Keywords

Unikernels, Single Address Space, Cloud, Operating Systems, Virtualisation, Virtual Machines, Virtual Machine Monitor, Hypervisor, MicroVM, Rumprun, Firecracker.

Ευχαριστίες

Η παρούσα εργασία εκπονήθηκε στο διάστημα 2021-2023 στο Εργαστήριο Υπολογιστικών Συστημάτων του Εθνικού Μετσόβιου Πολυτεχνείου. Θα ήθελα να ευχαριστήσω τα μέλη του εργαστηρίου για την καθοδήγησή τους κατά τη διάρκεια της πορείας της εργασίας αυτής. Ιδιαίτερα, θα ήθελα να ευχαριστήσω τους Στράτο Ψωμαδάκη, Ορέστη Λάγκα Νικολό και Κωνσταντίνο Παπαζαφειρόπουλο για την πολύτιμη βοήθεια και τις συμβουλές τους που κατέστησαν την ολοκλήρωση της εργασίας αυτής δυνατή.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Αποσπασμάτων Κώδικα

Κεφάλαιο 1

Εισαγωγή

Τα τελευταία χρόνια έχει παρατηρηθεί η αύξηση της δημοφιλίας υπηρεσιών που προσφέρουν απομακρυσμένη πρόσβαση σε υπολογιστικούς πόρους. Σε συνδυασμό με την επικράτηση του Internet of Things (Διαδίκτυο των Πραγμάτων) και την απομακρυσμένη αποθήκευση δεδομένων, η υπολογιστική βιομηχανία παρατηρεί μια στροφή προς το Cloud Computing (Υπολογισμός στο Νέφος). Ως αποτέλεσμα, έρχεται στην επιφάνεια ένα νέο μοντέλο εκτέλεσης σε περιβάλλοντα συμβατά με τη φιλοξενία στο Cloud (Νέφος) και αντίστοιχα εξελίσσονται τεχνολογίες για την υποστήριξη αυτών των σεναρίων. Δεδομένης της εξάπλωσης αυτής, υπάρχει πίεση στο να βελτιστοποιηθεί η επίδοση των τεχνολογιών αυτών ως προς ποικίλους παράγοντες, όπως ο χρόνος εκτέλεσης και η χρήση μνήμης.

Σε αυτό το πλαίσιο, μια κατεύθυνση που εξερευνάται αφορά στη συνεισφορά του Λειτουργικού Συστήματος το οποίο φιλοξενεί την εφαρμογή στην κατανάλωση πόρων και πώς αυτή μπορεί να περιοριστεί. Θεμέλιο του Cloud Computing ωστόσο αποτελεί η εικονικοποίηση αφού η συντριπτική πλειονότητα υπολογισμού σε Cloud συμβαίνει σε εικονικό περιβάλλον. Έτσι, είναι λογικό οι μελέτες αυτές να δώσουν έμφαση στη συμπεριφορά του Λειτουργικού Συστήματος στο πλαίσιο εικονικοποίησης.

1.1 Αντικείμενο της διπλωματικής

Μία υποσχόμενη κατεύθυνση για τον περιορισμό της επιβάρυνσης που επιβάλλει το Λειτουργικό Σύστημα στην εφαρμογή αποτελεί η τεχνολογία των unikernel. Η υποκείμενη φιλοσοφία λέει ότι το περιβάλλον εκτέλεσης μιας εφαρμογής εισάγει πολλαπλές εξαρτήσεις ώστε να υποστηρίξει παλιά πρωτόκολλα, τις διαφορετικές πιθανές ανάγκες των εφαρμογών που μπορεί να τρέξει ένα σύστημα γενικού σκοπού (general purpose), τις τεχνολογίες σχεδιασμένες για να επιτύχουν συγχρονισμό όπως ο χρονοπρογραμματισμός διεργασιών ή τα νήματα. Παρατηρούμε όμως ότι πολύ συχνά η προσέγγιση μηχανών γενικού σκοπού δεν είναι σχετική. Συστήματα ειδικού σκοπού (single purpose) γίνονται όλο και πιο δημοφιλή με την εξάπλωση του Cloud Computing και των ενσωματωμένων συστημάτων. Στις περιπτώσεις αυτές, οι ποικίλες εξαρτήσεις που εισφέρουν οι βιβλιοθήκες και το Λειτουργικό Σύστημα είναι περιττές και επιβαρύνουν το σύστημα χωρίς λόγο.

Για την επίλυση του προβλήματος αυτού, επινοήθηκαν τα unikernel, τα οποία αποτελούν ενιαία εκτελέσιμα αρχεία τα οποία προκύπτουν δημιουργώντας ένα λεπτό στρώμα εξαρτήσεων (αντικαθιστώντας το ενδιάμεσο λογισμικό) και συνδυάζοντας το με την εφαρμογή. Ως αποτέλεσμα, το unikernel μπορεί να αυξήσει την επίδοση ως προς χαρακτηριστικά όπως ο χρόνος και η χρησιμοποίηση μνήμης· καθώς και την ασφάλεια του συστήματος, μειώνοντας την πιθανή επιφάνεια επίθεσης.

Μια ιδιαίτερα ενδιαφέρουσα περίπτωση unikernel αποτελεί το Rumprun το οποίο υποστηρίζει το πρότυπο POSIX και οπότε μπορεί να χρησιμοποιηθεί για ένα μεγάλο πλήθος εφαρμογών. Το Rumprun έχει υλοποιηθεί για εικονικοποίηση με χρήση του εξομοιωτή QEMU (με πιθανή επιτάχυνση από τη μονάδα πυρήνα KVM) ή του υπερεπόπτη του Xen Project, ωστόσο υπάρχει η επιθυμία για περαιτέρω βελτιστοποίηση στο επίπεδο του υπερεπόπτη, κάτι που οι τεχνολογίες αυτές δεν καταφέρνουν. Η φιλοσοφία των unikernel απαιτεί την ελαχιστοποίηση των εξαρτήσεων του εκτελέσιμου, αλλά στο πλαίσιο εικονικοποίησης είναι λογικό αυτή η αρχή να επεκταθεί και στον επόπτη της εικονικής μηχανής. Αυτή την ανάγκη καλείται να καλύψει το Firecracker το οποίο αποτελεί έναν επόπτη εικονικής μηχανής που δημιουργεί και εποπτεύει microVM, δηλαδή σμικροποιημένες εκδοχές των εικονικών μηχανών που τρέχει, περιορίζοντας το ίχνος που το ίδιο έχει ως υπερεπόπτης στην εφαρμογή.

Ωστόσο το Rumprun δεν προσφέρει υποστήριξη για το Firecracker, το οποίο χρησιμοποιεί διαφορετικά πρότυπα τα οποία δε μπορούν αυτόματα να επικοινωνήσουν με αυτά του Rumprun. Έτσι, σκοπός της διπλωματικής αυτής είναι να προσθέσουμε αυτή την υποστήριξη στο Rumprun ώστε να μπορεί να εκκινήσει και να εκτελεστεί υπό τη διαχείριση του Firecracker και ύστερα να αξιολογήσουμε κατά πόσο η μέθοδος αυτή μπορεί να βελτιώσει την επίδοση του Rumprun.

1.1.1 Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Μελετήθηκαν το unikernel Rumprun και ο υπερεπόπτης Firecracker.
2. Τροποποιήθηκε το Rumprun ώστε να εκκινείται στο εικονικό περιβάλλον του Firecracker.
3. Αξιολογήθηκε η επίδοση της εκτέλεσης του Rumprun επί του Firecracker.

1.2 Οργάνωση του τόμου

Στο Κεφάλαιο 2 παρουσιάζονται οι βασικές έννοιες που διαπραγματεύται η εργασία και οι τεχνολογίες πάνω στις οποίες χτίζει. Στο Κεφάλαιο 3 αναλύονται η διαδικασία τροποποίησης, τα κομβικά σημεία της υλοποίησης και τα τελικά αποτελέσματα αυτής. Η μεθοδολογία των μετρήσεων και η αξιολόγηση των αποτελεσμάτων δίνεται στο Κεφάλαιο 4. Τέλος, στο Κεφάλαιο 5 γίνεται ανασκόπηση της εργασίας και αναφέρονται πιθανές κατευθύνσεις για μελλοντικές επεκτάσεις.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

2.1 Cloud Computing

Το Cloud Computing αναφέρεται σε ένα σύνολο υπηρεσιών που σκοπό έχουν την παροχή υπολογιστικών δυνατοτήτων απομακρυσμένα μέσω του διαδικτύου διαμοιράζοντας ένα σύνολο πόρων (π.χ. δίκτυα, εξυπηρετητές, αποθηκευτικό χώρο, εφαρμογές και υπηρεσίες) στους καταναλωτές με αυτόματη και ταχεία πρόσβαση. Το NIST (National Institute of Standards and Technology) αναγνωρίζει τα 5 ακόλουθα κύρια χαρακτηριστικά [1]:

- *Κατ΄ απαίτηση αυτοεξυπηρέτηση*: Υπολογιστικοί πόροι είναι συνεχώς διαθέσιμοι για ζήτηση από τον χρήστη και ακολούθως παροχή τους, αυτοματοποιημένα και χωρίς την ανάγκη εμπλοκής ανθρώπινου παράγοντα.
- *Ευρεία δικτυακή πρόσβαση*: Οι πόροι διατίθονται διαδικτυακά, στοχεύοντας σε ευρεία κάλυψη των πελατειακών (client) συστημάτων.
- *Συγκέντρωση πόρων*: Οι παρεχόμενοι πόροι συναθροίζονται για διαμοιρασμό προς τους χρήστες με αφαιρετικό τρόπο ως προς λέπτομέρειες όπως η φυσική τοποθεσία των πόρων (καθόλου ή περιορισμένος έλεγχος και γνώση), η γνώση για πιθανούς χρήστες που μοιράζονται τους ίδιους πόρους και την ακριβή εξέλιξη της κατανομής αυτής που μπορεί να αλλάζει δυναμικά.
- *Ταχεία ελαστικότητα*: Ο χρήστης μπορεί δυναμικώς και με ελευθερία να επιλέξει ανά πάσα στιγμή να προσαρμόσει το μέγεθος και την ποσότητα των πόρων που καταλαμβάνει από ένα φαινομενικά απεριόριστο σύνολο.
- *Μετρούμενη Υπηρεσία*: Υπάρχει ικανότητα συλλογής δεδομένων για το βαθμό χρήσης των προσφερόμενων υπηρεσιών για πληροφόρηση του παρόχου καθώς και του χρήστη, αλλά πιθανώς και για κοστολόγηση των υπηρεσιών.

2.1.1 Μοντέλα Υπηρεσιών

Υπάρχουν διαφορετικά μοντέλα που περιγράφουν τις υπηρεσίες που προσφέρονται από το Cloud. Το NIST αναφέρει τις εξής [1]:

- *Infrastructure as a Service (IaaS — Υποδομή ως Υπηρεσία)*: Προσφέρονται στοιχειώδεις υπολογιστικοί πόροι ώστε να επιτρέψουν την υλοποίηση και έλεγχο λογισμικού σε χαμηλό επίπεδο. Ο χρήστης μπορεί να αναπτύξει είτε Λειτουργικά Συστήματα είτε εφαρμογές, τα οποία και μπορεί να διαχειρίζεται.

- *Platform as a Service (PaaS — Πλατφόρμα ως Υπηρεσία)*: Δίνεται η ικανότητα ανάπτυξης εφαρμογών προερχόμενες από τον χρήστη και υποστηριζόμενες από το περιβάλλον που προσφέρει ο πάροχος. Ο χρήστης δε διαχειρίζεται το περιβάλλον εκτέλεσης (π.χ. την υποδομή του νέφους και το Λειτουργικό Σύστημα), αλλά έχει έλεγχο της εφαρμογής και πιθανώς κάποιας παραμετροποίησης του συστήματος υποδοχής (host).

- *Software as a Service (SaaS — Λογισμικό ως Υπηρεσία)*: Προσφέρεται η δυνατότητα χρήσης εφαρμογών που δίνονται από τον πάροχο, εκτελώντας τες στο νέφος. Ο χρήστης δεν ελέγχει το περιβάλλον εκτέλεσης ή την εφαρμογή, με πιθανή εξαίρεση περιορισμένες παραμέτρους της εφαρμογής.

Πέραν από τα τρία μοντέλα που περιγράφει το NIST έχουν εξελιχτεί και μερικά παραπάνω μοντέλα τα οποία έχουν γίνει σχετικά δημοφιλή, ξεχωρίζοντας από τα προηγούμενα μέσω της εξειδίκευσης. Αυτά είναι:

- *Mobile "Backend" as a Service (MBaaS — Backend Κινητού ως Υπηρεσία)*: Προσφέρεται ένας τρόπος σύνδεσης των εφαρμογών για κινητά και τον ιστό με backend (σύστημα παρασκηνίου) αποθήκευση και επεξεργασία που συμβαίνει στο νέφος. Ταυτοχρόνως μπορεί να προσφέρει πολλά από τα κοινά χαρακτηριστικά που απαιτούν οι εφαρμογές κινητών (π.χ. διαχείριση χρηστών, push notifications, ενσωμάτωση των κοινωνικών δικτύων) [2]. Το MBaaS ξεχωρίζει από τα άλλα μοντέλα διότι στοχεύει συγκεκριμένα στην ανάπτυξη backend και τη χρήση τους για εφαρμογές κινητών (ή ιστού) και εφαρμόζει βελτιστοποιήσεις και διευκολύνσεις για αυτές τις περιπτώσεις [2, 3].

- *Serverless Computing ή Function-as-a-Service (FaaS — Συνάρτηση ως Υπηρεσία)*: Το serverless computing (υπολογισμός χωρίς εξυπηρετητή) προσφέρει στον χρήστη την ικανότητα να εκτελεί εφαρμογές χωρίς ο ίδιος να χρειάζεται να διαχειριστεί τις πιο περίπλοκες τεχνικές λεπτομέρειες όπως την ανάπτυξη (deployment) της εφαρμογής, την κλιμάκωση και την παρακολούθηση του συστήματος. Το μοντέλο αυτό δεν είναι πραγματικά χωρίς εξυπηρετητή (server), αλλά αναθέτοντας όλη τη διαχείριση σχετικά με αυτόν στον πάροχο, επιτρέπει στον καταναλωτή να αντιμετωπίσει την εφαρμογή αφαιρετικά σαν να μην υπήρχε server [4].

  Το FaaS αποτελεί ένα μοντέλο που αξιοποιεί serverless computing ώστε να επιτρέψει μικρές αυτόνομες συναρτήσεις να εκτελεστούν σε απάντηση γεγονότων όπως αιτήματα HTTPS. Οι πόροι που χρησιμοποιούνται κλιμακώνοται αυτόματα στο ρυθμό επικλήσεων των συναρτήσεων [5]. Σε κάποιες περιπτώσεις ο όρος FaaS ταυτίζεται με το serverless computing, αλλά από άλλους θεωρείται υποκατηγορία του δεύτερου.

2.1.2 Μοντέλα Ανάπτυξης

Το NIST περιγράφει τέσσερα μοντέλα deployment:

- *Private cloud (Ιδιωτικό Νέφος)*: Η υποδομή του νέφους παρέχεται σε ένα μόνο οργανισμό, αποτελούμενος από πολλαπλούς χρήστες. Δεν είναι απαραίτητο ότι ανήκει και διαχειρίζεται από τον ίδιο οργανισμό ενώ η τοποθεσία της μπορεί να βρίσκεται στις εγκαταστάσεις (on premises) ή όχι.

- *Community cloud (Κοινοτικό Νέφος)*: Η υποδομή του νέφους παρέχεται σε μία κοινότητα χρηστών από οργανισμούς με κοινούς προβληματισμούς (π.χ. αποστολή, απαιτήσεις ασφάλειας, πολιτική, ζητήματα συμμόρφωσης). Μπορεί να ανήκει και να διαχειρίζεται από κάποιον από τους οργανισμούς της κοινότητας, κάποιο τρίτο μέρος ή ενα συνδυασμό των προηγούμενων, ενώ η τοποθεσία της μπορεί να βρίσκεται στις εγκαταστάσεις των μελών ή όχι.

- *Public cloud (Δημόσιο Νέφος)*: Η υποδομή του νέφους παρέχεται στο γενικό κοινό. Μπορεί να ανήκει και να διαχειρίζεται από κάποιο επιχειρηματικό, ακαδημαϊκό ή κυβερνητικό οργανισμό, ή κάποιο συνδυασμό αυτών. Η τοποθεσία της βρίσκεται στις εγκταστάσεις του παρόχου του νέφους.

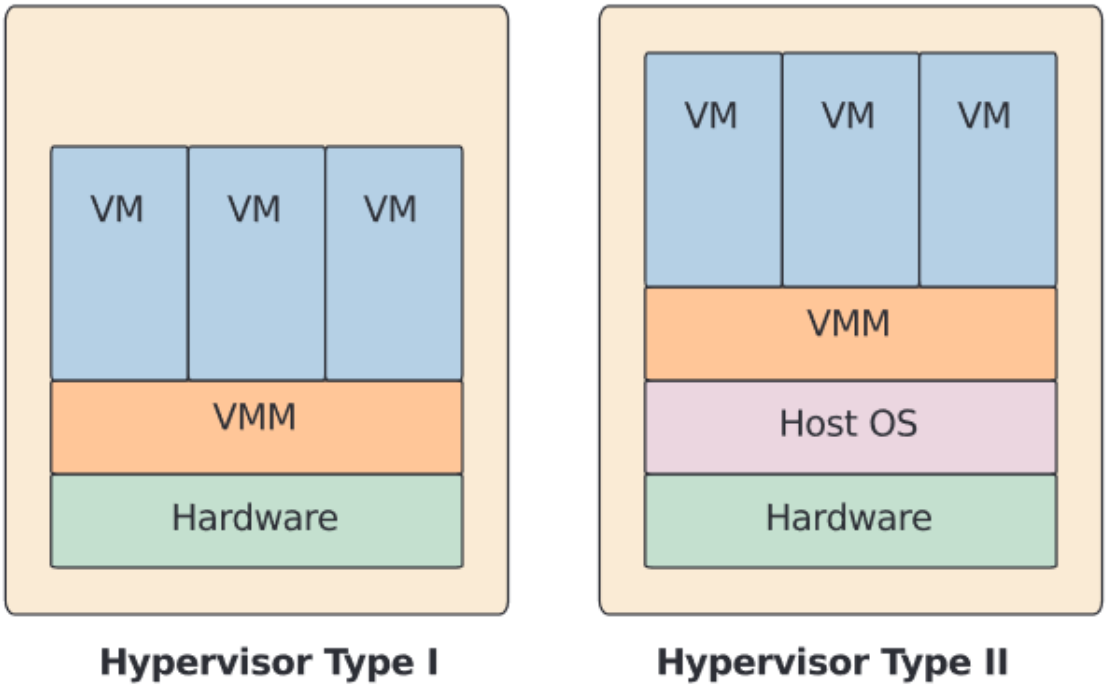
- *Hybrid cloud (Υβριδικό Νέφος)*: Η υποδομή του νέφους είναι η σύνθεση δύο η παραπάνω διακριτών υποδομών (private, community ή public) οι οποίες παραμένουν μοναδικές οντότητες, αλλά συνδυάζονται με τη χρήση τυποποιημένης ή ιδιόκτητης (proprietary) τεχνολογίας που επιτρέπει φορητότητα δεδομένων και εφαρμογών [1].

2.2 Εικονικοποίηση

Η εικονικοποίηση (virtualisation) μπορεί να οριστεί ως ένα σύστημα ή μέθοδος που συνδυάζει ή μοιράζει υπολογιστικούς πόρους σε ένα ή περισσότερα απομονωμένα περιβάλλοντα εκτέλεσης [17, 18]. Ένα εικονικό περιβάλλον θεωρείται από τον εαυτό του και εξωτερικά συστήματα το ίδιο ως ένα πραγματικό περιβάλλον, αλλά διαφέρουν οι υποκείμενοι μηχανισμοί που το απαρτίζουν [17]. Δεν υλοποιούνται, δηλαδή, γνήσια κατά τον ορισμό τους, αλλά εξομοιώνονται με διάφορους τρόπους από υποκατάστατα. Για αυτόν το σκοπό, χρησιμοποιούνται μέθοδοι όπως διαμέριση ή συσσωμάτωση υλικού και λογισμικού, μερική ή πλήρης προσομοίωση μηχανής, εξομοίωση και χρονομερισμός (time-sharing) [17].

Η ιδέα της εικονικοποίησης έχει ποικίλες εφαρμογές, όπως είναι η εικονική μνήμη, η εικονικοποίηση εφαρμογής (application virtualisation) και η εικονικοποίηση επιφάνειας εργασίας (desktop virtualisation) [17, 19]. Θα εστιάσουμε, όμως, στην εικονικοποίηση υλικού (hardware virtualisation) κατά την οποία δημιουργείται μια εικονική μηχανή (virtual machine — VM) η οποία φέρεται ως ένας πραγματικός υπολογιστής με Λειτουργικό Σύστημα. Γίνεται διαχωρισμός μεταξύ του λογισμικού που τρέχει σε μια τέτοια εικονική μηχανή από το πραγματικό υλικό (hardware) που την φιλοξενεί [20]. Το φιλοξενούν σύστημα ονομάζεται host (οικοδεσπότης), ενώ η εικονική πλατφόρμα ονομάζεται guest (επισκέπτης).

Hypervisor (υπερεπόπτης) ή virtual machine monitor (ελεγκτής εικονικών μηχανών) ή απλώς VMM αποκαλείται ένα στρώμα λογισμικού το οποίο εικονικοποιεί τους αναγκαίους πόρους χαμηλού επιπέδου για την υποστήριξη ενός εικονικού περιβάλλοντος. Ο hypervisor μπορεί να δημιουργεί και να εκτελεί πολλαπλές εικονικές μηχανές, οι οποίες είναι ανεξάρτητες και απομονωμένες μεταξύ τους [17, 21]. Υπάρχουν δύο τύποι VMM, οι λεγόμενοι Τύπος Ι και Τύπος ΙΙ. Ο τύπος Ι τρέχει κατευθείαν πάνω στο υλικό και προσφέρει τη διαχείριση πόρων στο χαμηλό επίπεδο, ενώ ο τύπος ΙΙ εκτελείται μέσα σε ένα λειτουργικό σύστημα, από του οποίου τον χρονοδρομολογητή διεργασιών εξαρτάται αφού κάθε εικονική μηχανή είναι μία διεργασία [21].



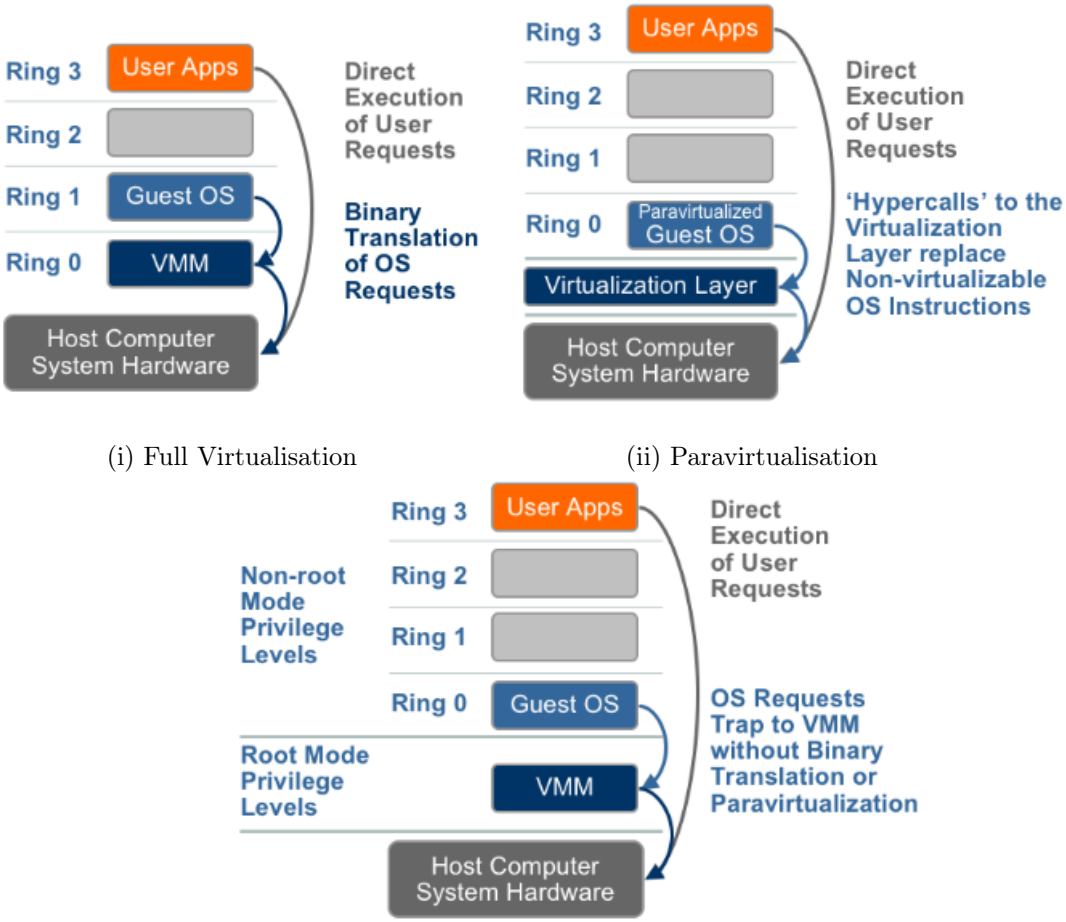
Σχήμα 2.1: Τύποι Hypervisor [21]

2.2.1 Είδη Hardware Virtualisation

- Το **Emulation** (εξομοίωση) μίας αρχιτεκτονικής επιτρέπει την εκτέλεση guest Λειτουργικών Συστημάτων χωρίς καθόλου τροποποιήσεις. Για αυτόν τον σκοπό, χρησιμοποιείται η τεχνική της δυαδικής μετάφρασης (binary translation), στην οποία εξομοιώνεται ένα σύνολο εντολών από ένα άλλο μέσω της μετάφρασης κώδικα. Η πλήρης εξομοίωση ενός επεξεργαστή προσφέρει το πλεονέκτημα της συμβατότητας κώδικα και λειτουργικών συστημάτων μεταξύ διαφορετικών πλατφορμών, αλλά έχει και το μειονέκτημα της επιβάρυνσης (overhead) που εισάγει η εξομοίωση ολόκληρου του συνόλου εντολών [21].
- Το **Full Virtualisation** (πλήρης εικονικοποίηση) επίσης χρησιμοποιεί binary translation, αλλά εξομοιώνει μόνο ένα μικρό μέρος του συνόλου εντολών. Συγκεκριμένα,

μεταφράζει τον κώδικα που απαιτεί προνομιακή εκτέλεση, όπως στην περίπτωση της λειτουργίας πυρήνα (kernel mode) και της λειτουργίας πραγματικών διευθύνσεων (real mode). Ο υπόλοιπος κώδικας εκτελείται άμεσα από τον επεξεργαστή του host. Αυτό έχει ως αποτέλεσμα σημαντική βελτίωση στην επίδοση σε σχέση με πλήρες emulation. Το full virtualisation επιτρέπει επίσης μη τροποποιημένους guest, αλλά μπορεί να υποστηρίξει μόνο την αρχιτεκτονική του host επεξεργαστή [21, 22].

- Η τεχνική **Paravirtualisation** (παραεικονικοποίηση) εισάγει ένα ειδικό σύνολο εντολών που ονομάζονται υπερκλήσεις (hypercall) και οι οποίες αντικαθιστούν κάποιες εντολές του πραγματικού επεξεργαστή και παραπέμπουν την εκτέλεση στον hypervisor. Αντίθετα με την πλήρη εικονικοποίηση, σε αυτήν την περίπτωση το σύστημα γνωρίζει ότι είναι εικονικό. Η τεχνική αυτή θυσιάζει την ικανότητα εκτέλεσης μη τροποποιημένων guest, διότι απαιτούνται αλλαγές στον πυρήνα (kernel) του Λειτουργικού Συστήματος για την αξιοποίηση των hypercall. Σε αντάλλαγμα, επιτύγχάνεται αυξημένη επίδοση λόγω του χαμηλού overhead εικονικοποίησης [21, 22].



(i) Full Virtualisation

(ii) Paravirtualisation

(iii) Hardware-assisted Virtualisation

Σχήμα 2.2: Σύγκριση Full, Para- και Hardware-assisted Virtualisation [22]

- Το **Hardware-assisted Virtualisation** (εικονικοποίηση υποβοηθούμενη από το υλικό) είναι μια παραλλαγή του full virtualisation που αξιοποιεί την υποστήριξη για εικονικοποίηση από το ίδιο το υλικό που έχουν προστεθεί σε κάποιους επεξεργαστές, όπως της AMD και της Intel. Οι επεξεργαστές αυτοί παρέχουν έναν νέο δακτύλιο χαμηλότερο του Δακτύλιου 0 (Ring 0), στον οποίο μπορεί να τρέχει ο VMM, επιτρέποντας στο guest Λειτουργικό Σύστημα να εκτελείται στο Ring 0 χωρίς να επηρεάζει τους άλλους guest ή τον host. Έτσι, υποστηρίζεται το virtualisation χωρίς να απαιτείται binary translation ή paravirtualisation. Αντί αυτού, οι προνομιούχες εντολές αυτόματα προκαλούν διακοπές (trap) που δίνουν τον έλεγχο στον hypervisor [21, 22].

- Το **Containerisation** (χρησιμοποίηση κοντέινερ) ή αλλιώς γνωστό και ως Operating System-level virtualisation (εικονικοποίηση στο επίπεδο Λειτουργικού Συστήματος), είναι μια τεχνική που χρησιμοποιεί έναν τροποποιημένο πυρήνα ώστε να εκτελέσει πολλαπλά απομονωμένα περιβάλλοντα που ονομάζονται container (επίσης virtual private server και jail). Κάθε container κάνει χρήση του ίδιου πυρήνα με το host Λειτουργικό Σύστημα. Η τεχνική αυτή έχει χαμηλό overhead και βρίσκεται σε ευρεία χρήση, ωστόσο δεν μπορεί να υποστηρίξει πολλαπλούς πυρήνες [21].

2.2.2 Τεχνολογίες Εικονικοποίησης

Με τα χρόνια, έχει αναπτυχθεί ένας μεγάλος αριθμός τεχνολογιών που υλοποιούν την ιδέα της εικονικοποίησης. Παρακάτω αναλύονται συντόμως μερικές εξ αυτών, στις οποίες θα γίνει αναφορά στα πλαίσια αυτής της διπλωματικής.

- Το *QEMU* είναι ένας ανοιχτού κώδικα γενικός εξομοιωτής μηχανής και εικονικοποιητής. Ως εξομοιωτής μπορεί να εκτελέσει λειτουργικά συστήματα και προγράμματα σχεδιασμένα για μία αρχιτεκτονική σε επεξεργαστή διαφορετικής αρχιτεκτονικής. Από την άλλη, χρησιμοποιώντας τον Xen hypervisor ή το KVM, το QEMU μπορεί να δράσει ως virtualiser και να επιτύχει επιδόσεις σχεδόν στο ίδιο επίπεδο πραγματικής μηχανής [26].

- Το *KVM*, ή Kernel-based Virtual Machine, είναι μία μονάδα πυρήνα (kernel module) για το λειτουργικό σύστημα Linux που προσφέρει hardware-assisted virtualisation σε x86 επεξεργαστές. Όταν φορτώνεται, μετατρέπει τον πυρήνα σε έναν hypervisor τύπου I και δίνει τη δυνατότητα στα Linux να τρέξουν πολλαπλές εικονικές μηχανές χωρίς να τροποποιείται ο κώδικας του kernel. Ο host βλέπει τις εικονικές μηχανές ως Linux διεργασίες, ενώ μια τροποποιημένη εκδοχή του QEMU μπορεί να χρησιμοποιηθεί για να παρέχει εξομοίωση συσκευών όπως το BIOS, η PCI δίαυλος (bus) και οι κάρτες δικτύου [19, 27].

- *Xen* είναι ένας τύπου I υπερεπόπτης ανοιχτού κώδικα ο οποίος στοχεύει στην ευελιξία και την εξατομίκευση, επιτρέποντας πολλά έργα να χτιστούν πάνω σε αυτόν. Είναι ιδιαίτερα διαδεδομένος στο Cloud [29, 30].

- Το *Solo5* αρχικά υλοποιήθηκε ώστε να μπορεί να εκτελεστεί το MirageOS unikernel στον Linux/KVM hypervisor. Εκ τότε όμως, έχει εξελιχθεί σε ένα πιο γενικό απομονωμένο (sandboxed) περιβάλλον εκτέλεσης, στοχευμένο προς τα unikernel. Αξιοποιεί διάφορες τεχνολογίες sandboxing σε μια πληθώρα host λειτουργικών συστημάτων και hypervisor [28].

2.2.2.1 Virtio

Ένα αξιοσημείωτο πρωτόκολλο στον χώρο της εικονικοποίησης, το οποίο χρησιμοποιείται και από το Firecracker είναι το virtio. Το virtio περιγράφει μια οικογένεια συσκευών οι οποίες εμφανίζονται σε εικονικά περιβάλλοντα, αλλά είναι σχεδιασμένες ώστε να φαίνονται ως φυσικές συσκευές στον guest της εικονικής μηχανής [25]. Με άλλα λόγια το virtio προσφέρει ένα μηχανισμό paravirtualisation, επιδιώκοντας την κανονικοποίηση του πώς αυτό επιτυγχάνεται.

Το virtio ορίζει προδιαγραφές και για συσκευές και για driver. Σε γενικά πλαίσια, συσκευές είναι φυσικά εξαρτήματα που συνδέονται με τον επεξεργαστή και επηρεάζουν τη λειτουργία του μέσω ανταλλαγής δεδομένων. Στα πλαίσια της εικονικοποίησης, ωστόσο, οι συσκευές μπορούν να αναφέρονται σε μονάδες κώδικα του hypervisor που εξοσομοιώνουν τη συμπεριφορά των φυσικών συσκευών για χρήση από μια εικονική μηχανή. Driver, από την άλλη, είναι προγράμματα του Λειτουργικού Συστήματος σχεδιασμένα για να ελέγχουν τη λειτουργία των συσκευών (είτε φυσικές, είτε εικονικές) από την πλευρά του επεξεργαστή. Στα πλαίσια της εικονικοποίησης, οι driver αναφέρονται συνήθως στα προγράμματα που υλοποιεί ο guest. Έτσι, για το virtio και γενικότερα το paravirtualisation, συσκευές και driver αποτελούν το λογισμικό που πρέπει να υλοποιηθεί από τις δύο πλευρές, τον εικονικοποιητή και την εικονική μηχανή, ώστε να επιτευχθεί η παραεικονικοποίηση.

Η λειτουργία του virtio βασίζεται κυρίως στη χρήση ενός τύπου ουρών που ονομάζονται virtqueues και χρησιμοποιούνται για μαζική μεταφορά δεδομένων. Ο driver μπορεί να κάνει αιτήματα προς τη συσκευή προσθέτοντας έναν διαθέσιμο buffer (ενδιάμεση μνήμη) στην ουρά, ενώ η συσκευή μπορεί να ενημερώσει τον driver ότι έχει πραγματοποιήσει κάποιο αίτημα επισημαίνοντας τον buffer ως χρησιμοποιημένο [25].

Το virtio επίσης χρησιμοποιεί ειδοποιήσεις (notifications) ώστε ο driver και η συσκευή να μπορούν να ενημερώνουν ο ένας τον άλλο για τυχούσες αλλαγές στην κατάστασή τους. Υπάρχουν τρεις τύποι ειδοποιήσεων: ειδοποιήσεις αλλαγής της διαμόρφωσης (configuration change), ειδοποιήσεις χρησιμοποιημένων buffer και ειδοποιήσεις διαθέσιμων buffer. Οι δύο πρώτες στέλνονται από τη συσκευή προς τον driver, ενώ η τελευταία στέλνεται από τον driver προς τη συσκευή. Οι ειδοποιήσεις των buffer χρησιμοποιούνται για να ενημερώσουν ότι κάποιος buffer είναι διαθέσιμος ή καταναλώθηκε. Η υλοποίηση των ειδοποιήσεων εξαρτάται από το στρώμα μεταφοράς, αλλά συνήθως οι ερχόμενες από τη συσκευή ειδοποιήσεις στέλνονται ως διακοπές [25].

Το virtio έχει υιοθετηθεί ευρέως και πολλά λειτουργικά συστήματα όπως το Linux και αυτά της οικογένειας BSD παρέχουν drivers για αυτό. Παρομοίως, παρατηρούμε χρήση του σε πολλές από τις πειραματικές τεχνολογίες εικονικών περιβάλλοντων, ιδίως σε αυτές που

προσπαθούν να μεγιστοποιήσουν την επίδοσή τους.

2.2.2.2 MicroVM

Μία εξέλιξη που έχει εμφανιστεί στον χώρο της εικονικοποίησης τα τελευταία χρόνια είναι τα microVM (micro virtual machines — μικρο-εικονικές μηχανές), ελαφριές εικονικές μηχανές που στόχο έχουν τη δημιουργία και εκτέλεση πολλών VM γρήγορα και ταυτόχρονα, χωρίς να θυσιάζεται η ασφάλεια αυτών ή του host. Για να το επιτύχουν αυτό διατηρούν και βελτιώνουν την απομόνωση των εικονικών μηχανών, ενώ δανείζονται τη λογική των container και ελαχιστοποιούν τους πόρους και διεπαφές που αυτές χρησιμοποιούν. Ως αποτέλεσμα, τα microVM είναι ιδιαίτερα κατάλληλα για εφαρμογές serverless computing [23, 24, 31].

Οι παραδοσιακές εικονικές μηχανές επιτρέπουν εξαιρετική απομόνωση, αφού δεν μοιράζονται πόρους και δεν γνωρίζουν για την ύπαρξη η μία της άλλης. Επιπλέον τα παραδοσιακά περιβάλλοντα εικονικών μηχανών όχι μόνο επιτρέπουν την εικονικοποίηση διαφορετικών guest λειτουργικών συστημάτων, αλλά επιχειρούν όσο μεγαλύτερη κάλυψη σε συμβατότητα των διαφορετικών υπάρχοντων πλατφορμών. Αυτές οι ιδιότητες ωστόσο έχουν το κόστος ότι οι εικονικές μηχανές είναι μεγάλες, ακριβές κατασκευές οι οποίες χρειάζονται αρκετό χρόνο να προετοιμαστούν, ενώ ένας server δύσκολα υποστηρίζει πολλά εκτελούμενα VM ταυτόχρονα [24].

Οι container, από την άλλη, είναι μικρά συστήματα τα οποία μοιράζονται τον ίδιο πυρήνα και απαιτούν λίγους υπολογιστικούς πόρους, επιτρέποντας δεκάδες ή εκατοντάδες container να εκτελούνται ταυτόχρονα, εκκινώντας σε λίγα δευτερόλεπτα. Η χρήση του ίδιου πυρήνα ωστόσο κάνει τους container λιγότερο ασφαλείς, αφού δεν είναι απομονωμένοι μεταξύ τους και ένα τρωτό σημείο στο Λειτουργικό Σύστημα μπορεί να επηρεάσει όλους τους container [23, 24].

Τα microVM συνδυάζουν τα πλεονεκτήματα και των δύο, ενώ ελαττώνουν τα μειονεκτήματά τους. Σε αντίθεση με τα παραδοσιακά VM, τα microVM υλοποιούν πολύ συγκεκριμένες και περιορισμένες διεπαφές και συσκευές για να εικονικοποιήσουν τον guest. Έτσι, απλοποιείται ο ενδιάμεσος hypervisor και τα δημιουργούμενα VM μπορούν να είναι μικρά και γρήγορα. Ταυτοχρόνως, κάθε microVM παραμένει απομονωμένο από τα υπόλοιπα και μπορεί να χρησιμοποιήσει το δικό του Λειτουργικό Σύστημα [24, 31].

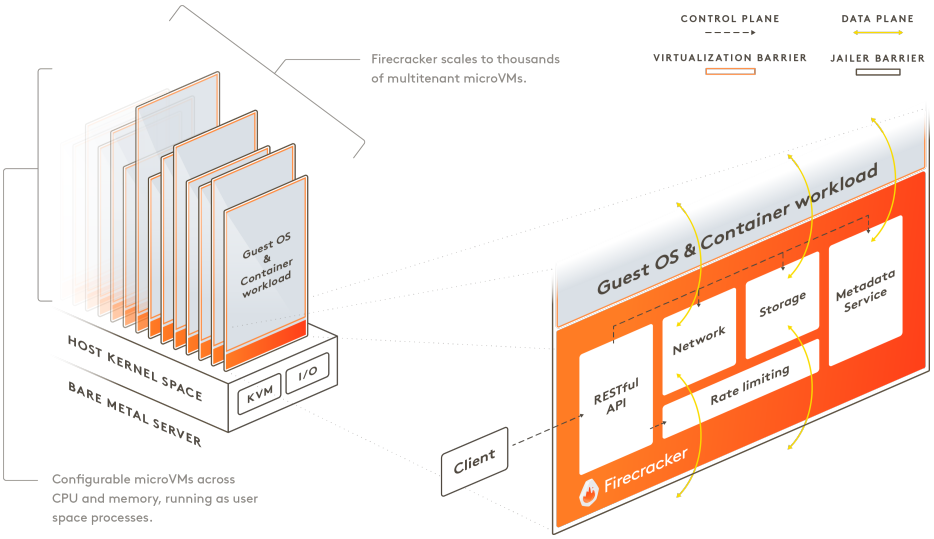
2.2.2.3 Firecracker

Ένα παράδειγμα τεχνολογίας microVM είναι ο Firecracker VMM, ένας τύπου II hypervisor, ο οποίος δημιουργεί και διαχειρίζεται microVM χρησιμοποιώντας το KVM. Το Firecrakcer είναι σχεδιασμένο μινιμαλιστικά, αφαιρώντας μη απαραίτητες συσκευές και λειτουργικότητα του guest. Με αυτό τον τρόπο περιορίζει το αποτύπωμα μνήμης και την επιφάνεια επίθεσης. Έτσι, πετυχαίνει το στόχο του να δημιουργήσει ασφαλείς και απομονωμένες εικονικές μηχανές, με την ταχύτητα και αποδοτικότητα πόρων των container. Το Firecracker είναι διαθέσιμο σε 64-bit Intel, AMD και ARM επεξεργαστές που υποστηρίζουν hardware virtualisation [31].

Το Firecracker σχεδιάστηκε από τις Amazon Web Services για αξιοποίηση σε υπηρεσίες serverless computing (π.χ. AWS Lambda, AWS Fargate), ωστόσο είναι δημοσιοποιημένο ως λογισμικό ανοιχτού κώδικα και έχει χρησιμοποιηθεί από και ενσωματωθεί με ένα μεγάλο πλήθος τεχνολογιών, συμπεραλαμβανομένων διαχειριστών container και πλατφορμών serverless computing. Μπορεί δε να εκτελέσει Linux και OSv guests [31].

Το Firecracker ξεκίνησε από το crosvm, έναν ανοιχτού κώδικα VMM γραμμένο σε Rust για το Chromium OS [31]. Το crosvm εκτελεί Linux guests (και άλλα λειτουργικά συστήματα) με σκοπό να τρέξει εγγενείς (native) εφαρμογές [36, 37]. Επίσης κοινή χρήση κώδικα με τα δύο παραπάνω έργα έχει και ο Cloud Hypervisor, ένας VMM σχεδιασμένος να υποστηρίζει Cloud Workloads (φόρτους εργασίας) γενικού σκοπού [38]. Αν και οι τρεις τεχνολογίες αυτές κατά την εξέλιξή τους απέκλιναν μεταξύ τους, υπηρετώντας αρκετά διαφορετικούς σκοπούς, συνεχίζουν να μοιράζονται ως κοινή σχεδιαστική αρχή την έμφαση στην απλότητα, ασφάλεια και ταχύτητα. Για αυτό το λόγο, υπήρξε μια συνεργατική προσπάθεια μεταξύ των δημιουργών τους καθώς και άλλων οργανισμών (συγκεκριμένα οι Alibaba, AWS, Cloud Base, Crowdstrike, Intel, Google, Linaro, Red Hat) και ατομικών συνεισφερόντων να αναπτυχθεί ένα ανοιχτού κώδικα Rust πακέτο (crate) για τη δημιουργία VMM. Αυτό ονομάζεται Rust-VMM και μπορεί να χρησιμοποιηθεί από οποιδήποτε έργο χωρίς να χρειάζεται επανα-υλοποίηση των ίδιων κοινών στοιχείων [31, 36, 38, 39].

Το Firecracker παρέχει μόνο πέντε εξομοιωμένες συσκευές: τη σειριακή κονσόλα, έναν περιορισμένο ελεγκτή πληκτρολογίου (keyboard controller) που χρησιμοποιείται μόνο για να τερματίσει το microVM καθώς και τις συσκευές virtio-net, virtio-block, virtio-vsock [31].



Σχήμα 2.3: Παράδειγμα εκτέλεσης Firecracker [31]

Όταν εκκινήσει η διαδικασία του Firecracker, αυτή μπορεί να ελεγχθεί από ένα RESTful API το οποίο επιτρέπει κοινές ενέργειες, όπως τη διαχείριση του αριθμού εικονικών επεξεργαστών (vCPU), τη ρύθμιση συσκευών και την εκκίνηση του microVM. Το API (application programming interface — διεπαφή προγραμματισμού εφαρμογών) επίσης παρέχει ευέλικτους rate limiter (περιοριστές ρυθμού), οι οποίοι μπορούν να θέσουν όρια στους πόρους (π.χ.

δίκτυο και αποθηκευτικός χώρος) που χρησιμοποιούν τα microVM. Επιπλέον, υπάρχει μια υπηρεσία μεταδεδομένων, η οποία επιτρέπει την ανταλλαγή πληροφοριών διαμόρφωσης μεταξύ του host και guest, τη λειτουργία της οποίας ελέγχει το API. Κάποιες από αυτές τις λειτουργίες μπορούν να διαμορφωθούν και από ένα φάκελο (config file) που ορίζεται κατά την εκτέλεση της εντολής firecracker [31].

Για να επιτευχθεί η ασφάλεια που επιδιώκει το Firecracker, λαμβάνονται πολλαπλά μέσα. Εξαρχής, η γλώσσα προγραμματισμού που επιλέχτηκε για αυτό είναι η Rust, η οποία δίνει μεγάλη σημασία στην ασφάλεια μνήμης και νημάτων. Κατά τη φιλοσοφία του microVM, υπάρχει απομόνωση των εικονικών μηχανών, ενώ το περιορισμένο μοντέλο συσκευών επιτρέπει τη μείωση της επιφάνειας επίθεσης. Ταυτόχρονα, το Firecracker προσφέρει περαιτέρω απομόνωση με τη χρήση του συνοδευτικού προγράμματος jailer, το οποίο χρησιμοποιεί φράγματα ασφαλείας του Linux user space (χώρου χρήστη) [31, 38].

Επιπλέον του μοντέλου συσκευών, το Firecracker επίσης ελαχιστοποιεί τη διαμόρφωση του guest kernel και επιταχύνει τη φόρτωσή του. Ως αποτέλεσμα, εξασφαλιζονται ιδιαίτερα γρήγοροι χρόνοι εκκίνησης. Το Firecracker είναι ικανό να προετοιμάσει και εισέλθει στην εφαρμογή σε 125 χιλιοστά του δευτερολέπτου (millisecond), ενώ μπορεί να δημιουργήσει μέχρι και 150 microVM το δευτερόλεπτο [31].

Μπορεί ακόμη να υποστηρίξει μεγάλο αριθμό microVM που συνυπάρχουν ταυτόχρονα. Αυτό γίνεται δυνατό λόγω του χαμηλού overhead μνήμης (λιγότερο από 5 MiB) και των rate limiter που προσφέρει για βελτιστοποίηση του καταμερισμού πόρων. Αυτό επιτρέπει την ταυτόχρονη λειτουργία πλήθους microVM στην τάξη μεγέθους των χιλιάδων [31].

2.3 Unikernel

2.3.1 Εισαγωγή

Τα unikernel είναι εξειδικευμένες εικόνες μηχανών (machine images) με ενιαίο χώρο διευθύνσεων κατασκευασμένες με τη χρήση λειτουργικών συστημάτων βιβλιοθηκών (library operating systems) [6]. Ο προγραμματιστής μπορεί να επιλέξει τις ελάχιστες μονάδες που η εφαρμογή απαιτεί από ένα σύνολο βιβλιοθηκών που υλοποιούν τη λειτουργικότητα που αντιστοιχεί στο Λειτουργικό Σύστημα (π.χ. η στοίβα δικτύου, το σύστημα αρχείων, τα προγράμματα οδήγησης συσκευών). Οι επιλεγμένες βιβλιοθήκες μεταγλωττίζονται μαζί με την εφαρμογή και τον κώδικα παραμετροποίησης, παράγοντας ένα εκτελέσιμο ειδικού σκοπού το οποίο μπορεί να τρέξει άμεσα είτε σε κάποιον υπερεπόπτη είτε κατευθείαν στο υλικό (bare metal).

Τα unikernel αποτελούν μια σύγχρονη τεχολογία, η ανάπτυξη της οποίας κατεστάθη δυνατή (και έγινε επιθυμητή) λόγω της εξάπλωσης του Cloud Computing και της εικονικοποίησης. Ωστόσο οι βασικές αρχές τους δεν είναι νέες και παρόμοιες τεχνολογίες έχουν εξελιχτει και παλαιότερα, με παραδείγματα το Exokernel και το Nemesis [7]. Η ιδέα του Library Operating System (ή libOS) δοκιμάστηκε τη δεκαετία του 1990, αλλά δεν εδραιώθηκε λόγω του μεγάλου εύρους φυσικών συσκευών που θα έπρεπε να μπορούν να υποστηρίζουν [7, 8].

Με τη σύγχρονη άνοδο της εικονικοποίησης ωστόσο, ενισχυμένη από την εξάπλωση του Cloud Computing, τα σενάρια στα οποία λογισμικό τρέχει πάνω σε έναν υπερεπόπτη, ο οποίος προσφέρει τους δικούς του driver για τις φυσικές συσκευές γίνονται όλο και πολυπληθέστερα [8]. Σε αυτή την περίπτωση το libOS που εκτελείται πάνω στον υπερεπόπτη αρκεί να υλοποιήσει driver για τον μικρότερο αριθμό εικονικών συσκευών που παρέχονται από αυτόν [7].

2.3.2 Η Θεωρία των Unikernel

Το Library Operating System που ιστορικά αποτελεί τον προκάτοχο των σημερινών unikernel περιγράφει μια αρχιτεκτονική στην οποία η εφαρμογή δε δέχεται περιορισμούς προνομίων στο επίπεδο λογισμικού, αλλά αυτά τα όρια προστασίας απαντούνται μόνο στο χαμηλότερο επίπεδο του υλικού. Ως αποτέλεσμα προκύπτουν:

- Ένα σύνολο βιβλιοθηκών που υλοποιούν μηχανισμούς που σε συμβατικά σενάρια αναλαμβάνει το Λειτουργικό Σύστημα (π.χ. οδήγηση του hardware, επικοινωνία με το δίκτυο).

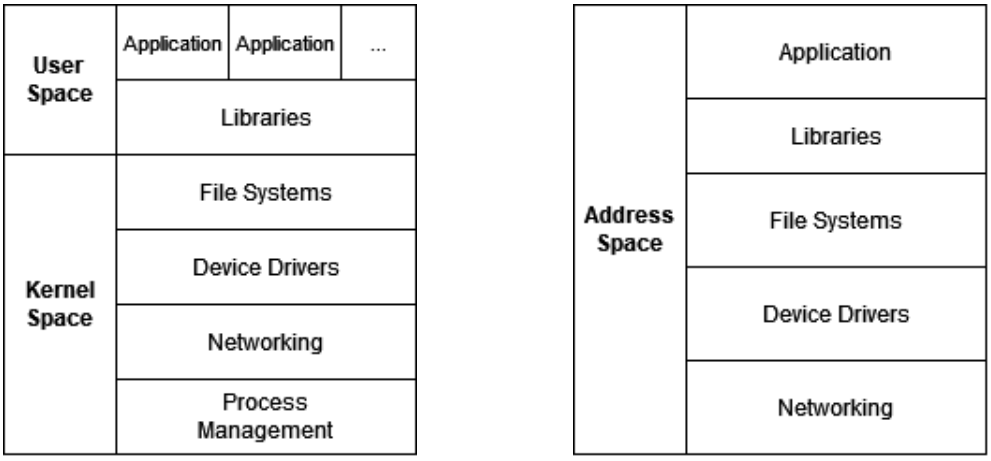
- Ένα σύνολο πολιτικών που επιβάλλουν τον έλεγχο πρόσβασης (access control) και την απομόνωση στο στρώμα εφαρμογής (application layer) [7].

Η προσέγγιση αυτή σημαίνει ότι πρόσβαση στο hardware και το δίκτυο δεν περιορίζεται στον χώρο πυρήνα (kernel space) αλλά οι εφαρμογές μπορούν να έχουν απευθείας πρόσβαση σε αυτά, αντί για ενδιαμέσως ως υπηρεσία που προσφέρει το kernel. Με αυτόν τον τρόπο, μπορούν να αποφευχθούν οι συνεχείς μεταβάσεις προνομίων (privilege transitions) για την ανταλλαγή δεδομένων μεταξύ user space και kernel space [7]. Αυτό δίνει ένα ιδιαίτερο πλεονέκτημα στο libOS, αφού επιτρέπει μια σημαντική βελτίωση στην επίδοση των εφαρμογών.

Τα unikernel χρησιμοποιούν τη λογική των libOS για να δημιουργήσουν μια εικόνα ειδικού σκοπού η οποία μπορεί να εξαλείψει όλες τις περιττές εξαρτήσεις. Έτσι, μπορούμε να κατανοήσουμε πώς σχηματίζεται ο ενιαίος χώρος διευθύνσεων συγκρίνοντας με τη στοίβα εφαρμογής (application stack) ένος παραδοσιακού Λειτουργικού Συστήματος γενικού σκοπού, όπως φαίνεται στο σχήμα 2.4.

Στην κλασική στοίβα, ο χώρος διευθύνσεων χωρίζεται σε δύο μέρη, το kernel space και το user space (Σχ. 2.4i). Το πρώτο περιέχει όλες τις ρουτίνες που διαχειρίζονται το σύστημα αρχείων, τις συσκευές, την επικοινωνία με το δίκτυο και τις διεργασίες. Από την άλλη, το user space περιέχει τις βιβλιοθήκες που αποτελούν διεπαφή με τον πυρήνα και τις ίδιες τις εφαρμογές.

Στο unikernel, ωστόσο, δεν υπάρχει ξεχωριστό kernel space και user space (Σχ. 2.4ii). Όλος ο κώδικας τρέχει σε προνομιακή λειτουργία και οι ρουτίνες του πυρήνα συνδυάζονται με την εφαρμογή και τις βιβλιοθήκες σε ένα μοναδικό εκτελέσιμο το οποίο διαθέτει ό,τι χρειάζεται για να εκκινήσει και να εκτελεστεί μόνο του. Είναι δηλαδή μια εικόνα στην οποία ο χώρος διευθύνσεων είναι ενιαίος.



(i) Στοίβα Λειτουργικού Συστήματος Γενικού Σκοπού (ii) Στοίβα Unikernel

Σχήμα 2.4: Στοίβα Λειτουργικού Συστήματος Γενικού Σκοπού και Unikernel

Σε αυτό το σημείο, ένα εύλογο ερώτημα που μπορεί να τεθεί είναι κατά πόσο ένα unikernel είναι ασφαλές εφόσον ολόκληρη η εφαρμογή εκτελείται στον Δακτύλιο 0 [10], ο οποίος επιτρέπει προνομιακές εντολές του επεξεργαστή που είναι επικίνδυνες αν καταχραστούν. Πρέπει να παρατηρηθεί, όμως, ότι ως επί το πλείστον, τα unikernel χρησιμοποιούνται σε κάποιο σενάριο εικονικοποίησης. Έτσι, σε αυτή την περίπτωση ακόμη και αν εκτεθεί το unikernel, υπάρχει σε ένα απομωνομένο περιβάλλον και δεν αποτελεί κίνδυνο προς το φυσικό μηχάνημα, ενώ οι ικανότητες του πρώτου που μπορεί να εκμεταλλευτεί μια πιθανή επίθεση είναι περιορισμένες από τη φύση των unikernel. Αν θέλουμε, βεβαίως, να ασφαλίσουμε και το ίδιο το unikernel έναντι αυτής της αδυναμίας, είναι και αυτό εφικτό. Εφόσον ο hypervisor μπορεί να προσφέρει διεπαφή προς συγκεκριμένες συσκευές σε ένα paravirtualized unikernel και να προετοιμάσει τους πίνακες σελιδών (page table) εκ των προτέρων, τότε δεν υπάρχει πλέον ανάγκη για πρόσβαση στο Ring 0 και μπορεί να τρέχει στο λιγότερο προνομιακό Ring 3 [10].

Μία επιπλέον διαφορά που παρατηρούμε μεταξύ των δύο σχημάτων είναι ότι στο παραδοσιακό Λειτουργικό Σύστημα, υπάρχουν πολλαπλές εφαρμογές που εκτελούνται και κάνουν χρήση της στοίβας παράλληλα. Αντιθέτως, στο unikernel υπάρχει μόνο μια εφαρμογή και αντιστοίχως μόνο μια διεργασία. Για την εκτέλεση πολλαπλών εφαρμογών, απαιτούνται πολλαπλά unikernel, το καθένα με τη δικιά του στοίβα. Έτσι, δεν υπάρχει ανάγκη διαχείρισης διεργασιών, οπότε αυτό το κομμάτι της στοίβας εξαλείφεται πλήρως, επιτυχάνοντας μια ακόμη βελτιστοποίηση ως προς την επίδοση.

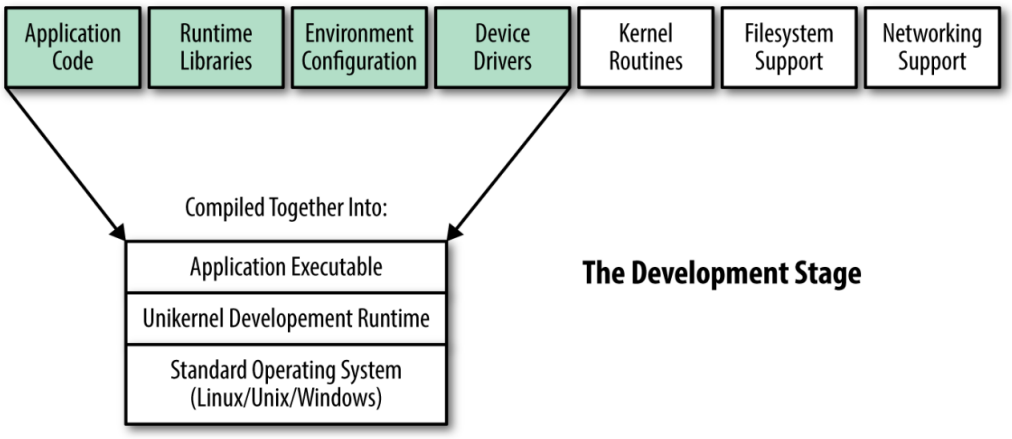
2.3.3 Η διαδικασία ανάπτυξης

Ένα πρόβλημα που μπορεί να προκύψει στο σχήμα που περιγράφεται στην προηγούμενη ενότητα είναι η δυσκολία ανάπτυξης (development) κώδικα για μια ενιαία στοίβα. Ένα από τα βασικά προτερήματα του Λειτουργικού Συστήματος γενικού σκοπού είναι ότι προσφέρει σταθερό δοκιμασμένο κώδικα για πολλές από τις λειτουργίες που δεν είναι χαρακτηριστικές του προβλήματος που τίθεται να λύσει η εφαρμογή και ένας προγραμματιστής δε θέλει να υλο-

ποιήσει ο ίδιος. Ως εναλλακτική του παραδοσιακού Λειτουργικού Συστήματος, το unikernel δε μπορεί να χάσει αυτή την ιδιότητα, αλλιώς η ανάπτυξη για την πλατφόρμα αυτή θα γινόταν πρακτικά αδύνατη [11].

Για αυτόν ακριβώς το λόγο, τα unikernel χρησιμοποιούν ένα εξειδικευμένο σύστημα μεταγλώττισης, ώστε να κάνουν κατά τη μεταγλώττιση (compile time) αυτό που τα καθιερωμένα προγράμματα κάνουν κατά την εκτέλεση (runtime) [11]. Δηλαδή, αντί να επιλέγεται δυναμικά ποιες ρουτίνες του Λειτουργικού Συστήματος θα κληθούν κατά τη διάρκεια εκτέλεσης, προαποφασίζεται κατά τη μεταγλώττιση ποιες ρουτίνες χρειάζεται η εφαρμογή και αυτές συμπεριλαμβάνονται στατικά στην προκύπτουσα εικόνα. Οι απαραίτητες συναρτήσεις χαμηλού επιπέδου παρέχονται από ένα libOS, οπότε δε χρειάζεται η υλοποίησή τους ακριβώς όπως με ένα συμβατικό Λειτουργικό Σύστημα.

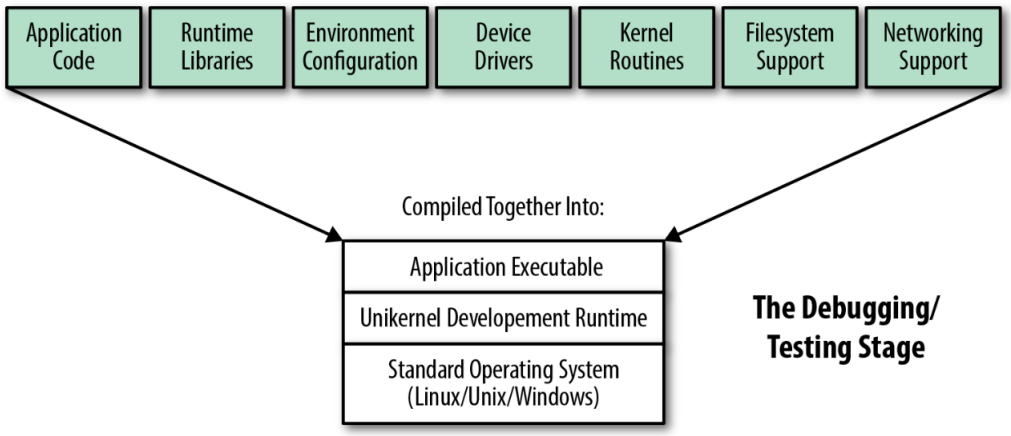
Ωστόσο, η διαδικασία αυτή θέτει ένα νέο ερώτημα: πόσο εύκολο είναι κανείς να αναπτύξει και αποσφαλματώσει λογισμικό υπό αυτές τις συνθήκες· Η ανησυχία αυτή είναι κατανοητή, αλλά τα unikernel έχουν σχεδιαστεί έτσι ώστε η διαδικασία του development να είναι εφικτή και όσο παρόμοια στη συμβατική περίπτωση όσο γίνεται [11]. Γενικώς μπορούμε να διαχωρίσουμε τρεις φάσεις στη διαδικασία ανάπτυξης λογισμικού, οι οποίες μπορούν να εφαρμοστούν και στην ανάλυση των unikernel: η φάση development, η φάση ελέγχου (testing) και η φάση παραγωγής (production).



Σχήμα 2.5: Φάση Unikernel Development [11]

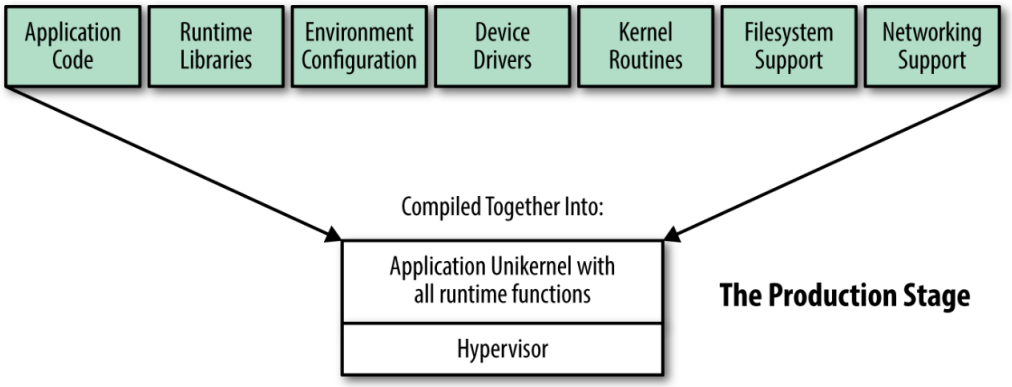
Στη φάση development (Σχ. 2.5), η εφαρμογή χτίζεται ως λογισμικό στην παραδοσιακή στοίβα. Όλες οι συναρτήσεις χαμηλού επιπέδου χειρίζονται από τον πυρήνα της μηχανής στην οποία γίνεται development, όπως και στη συμβατική περίπτωση. Επιτρέπεται έτσι η ελεύθερη ανάπτυξη λογισμικού και χρήση όλων των εργαλείων και μεθόδων που χρησιμοποιεί κανονικά ένας προγραμματιστής. Δεν υπάρχει δηλαδή κάποια παραπάνω δυσκολία σε αυτή τη φάση από ό,τι σε άλλες περιπτώσεις.

Στη φάση ελέγχου (Σχ. 2.6) από την άλλη, ο μεταγλωττιστής προσθέτει τις συναρτήσεις που σχετίζονται με τη δραστηριότητα του Λειτουργικού Συστήματος στο εκτελέσιμο από το χρησιμοποιούμενο libOS. Είναι πάλι διαθέσιμα όλα τα συνηθισμένα εργαλεία. Η διαφορά



Σχήμα 2.6: Φάση Unikernel Testing [11]

έγκειται στο ότι ο μεταγλωττιστής συμπεριλαμβάνει τις user space συναρτήσεις του libOS στην παραγόμενη εικόνα ώστε οι δοκιμές να γίνουν χωρίς εξάρτηση από τις βιβλιοθήκες του Λειτουργικού Συστήματος δοκιμής.



Σχήμα 2.7: Φάση Unikernel Production [11]

Τέλος, στη φάση production (Σχ. 2.7), η εικόνα είναι έτοιμη για χρήση στο περιβάλλον παραγωγής, με πλήρη λειτουργικότητα και την ικανότητα να εκκινήσει μόνο του.

2.3.4 Πλεονεκτήματα

Τα τέσσερα βασικά πλεονεκτήματα των unikernel συνοψίζονται ως εξής [6]:

- **Μικρό μέγεθος:** Οι εικόνες unikernel είναι συχνά τάξεις μεγέθους μικρότερες από εγκαταστάσεις παραδοσιακών Λειτουργικών Συστημάτων. Η αφαίρεση όλων των περιττών εξαρτήσεων και η διατήρηση μόνο των απαραίτητων στοιχείων για την αυτόνομη εκτέλεση της εφαρμογής έχει ως αποτέλεσμα τη μείωση του συνολικού μεγέθους μεταξύ μερικών εκατοντάδων KB και λίγων MB

- **Υψηλή επίδοση:** Το μοντέλο μεταγλώττισης των unikernel επιτρέπει βελτιστοποίηση ολόκληρου του συστήματος, συμπεριλαμβάνομένων τόσο στοιχείων χαμηλού επιπέδου όπως οι driver συσκευών όσο και τον κώδικα της εφαρμογής. Επιπλέον, περίσσιες διαδικασίες αποφεύγονται εντελώς, όπως για παράδειγμα περιγράψαμε για τα privilege transition στην ενότητα 2.3.2. Ως αποτέλεσμα, τα unikernel έχουν μια μοναδική ευκαιρία για αύξηση της επίδοσης.

- **Γρήγορη εκκίνηση:** Η αφαίρεση περιττών πρωτοκόλλων και βιβλιοθηκών εξαλείφει επίσης την ανάγκη αρχικοποίσης πολλών δομών κατά την έναρξη του unikernel. Έτσι, επιτυγχάνεται πολύ μικρός χρόνο εκκίνησης (boot), μετρούμενος σε millisecond. Αυτό δίνει τη δυνατότητα να εκκινούνται unikernel στο Cloud αρκετά γρήγορα ώστε να απαντήσουν σε εισερχόμενα αιτήματα (και αντιστοίχως γρήγορα να τερματιστούν όταν εξυπηρετήσουν το αίτημα) [9].

- **Αυξημένη ασφάλεια:** Με όλες τις ελαχιστοποιήσεις των unikernel, το μέγεθος του συνολικού κώδικα που χρησιμοποιείται περιορίζεται σημαντικά, οπότε μειώνεται αντίστοιχα και η πιθανή επιφάνεια επίθεσης (attack surface). Επιπλέον, στη σύνηθη περίπτωση τα unikernel εκτελούνται πάνω από έναν hypervisor, οπότε συνδυάζουν και την ιδιότητα της απομόνωσης, μην επιτρέποντας επικοινωνία με το φιλοξενούν ή άλλα φιλοξενούμενα σύστηματα.

2.3.5 Rumprun

Μία ενδιαφέρουσα υλοποίηση της ιδέας των unikernel είναι το Rumprun, το οποίο είναι ένα πλαίσιο λογισμικού (framework) που χτίζεται με τη χρήση των rump kernel. Ιδιαίτερο στοιχείο του Rumprun αποτελεί ότι υποστηρίζει POSIX εφαρμογές, επιτρέποντας την προσαρμογή σε unikernel υπάρχοντων εφαρμογών με καθόλου ή ελάχιστες αλλαγές [12].

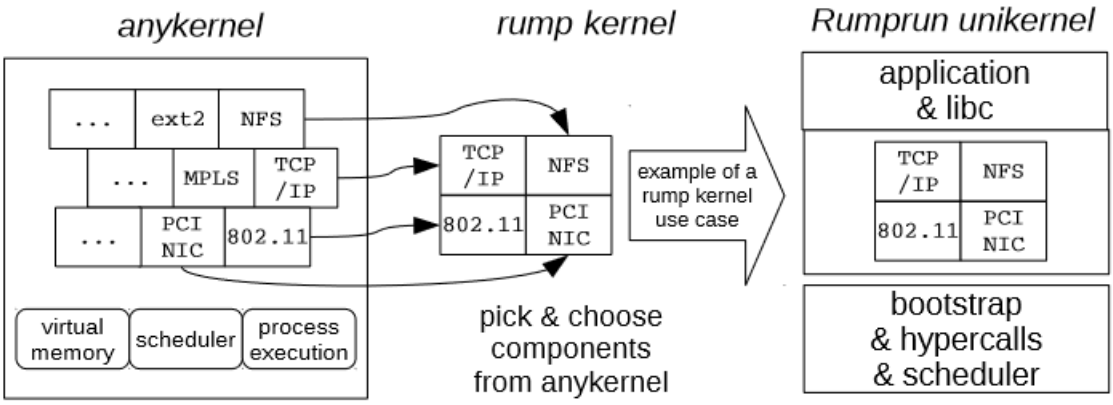
2.3.5.1 Rump kernels

Για να κατανοήσουμε τη δομή του Rumprun, θα πρέπει προηγουμένως να εξηγήσουμε τι είναι τα Rump kernel πάνω στα οποία χτίστηκε το πρώτο, καθώς και άλλες σχετικές έννοιες.

Πολλές διαφορετικές αρχιτεκτονικές έχουν σχεδιαστεί για τους πυρήνες των Λειτουργικών Συστημάτων. Ο μονολιθικός πυρήνας (monolithic kernel) αποτελεί την παραδοσιακή προσέγγιση, κατά την οποία ολόκληρο το Λειτουργικό Σύστημα λειτουργεί στο kernel space. Αντίθετα το microkernel ελαχιστοποιεί όσο το δυνατό το λογισμικό που βρίσκεται στον χώρο πυρήνα και προωθεί τις υπόλοιπες υπηρεσίες στον χώρο χρήστη. Πέραν των δύο αυτών βασικών ιδεών υπάρχουν και άλλες παραλλαγές ή συνδυασμοί αυτών όπως τα exokernel και hybrid kernel. Για το αντικείμενό μας, μας ενδιαφέρει ιδιαίτερα η αρχιτεκτονική anykernel.

Το anykernel περιγράφει μία βάση κώδικα τύπου πυρήνα από την οποία μπορούν να εξαχθούν οι driver (ανάμεσα στους οποίους, πέραν από τους driver συσκευών, καταχρηστικά θεωρούνται και το σύστημα αρχείων και η στοίβα δικτύου) για ενσωμάτωση σε οποιοδήποτε — ή σχεδόν οποιοδήποτε — από τα πολλά μοντέλα Λειτουργικών Συστημάτων [12], χωρίς την

ανάγκη τροποποιήσεων. Έτσι, οι driver μπορούν ανάμεσα σε άλλα να συντελέσουν σε έναν πλήρες μονολιθικό πυρήνα ή να αξιοποιηθούν για να χτιστεί ένα πιο επιλεκτικό microkernel.



Σχήμα 2.8: Σχέση Anykernel, Rump Kernel και Rumprun Unikernel [12]

Τα Rump kernel είναι ακριβώς μια υλοποίηση του μοντέλου των anykernel. Όπως το όνομα υπονοεί (rump: υπόλειμμα), ένα Rump kernel είναι ο πυρήνας ενός παραδοσιακού Λειτουργικού Συστήματος από το οποίο έχουν αφαιρεθεί κομμάτια. Παραμένουν οι driver και όποιες βασικές ρουτίνες χρειάζονται για να λειτουργήσουν αυτοί (π.χ. συγχρονισμός, κατανομή μνήμης). Από την άλλη, αφαιρούνται πολιτικές όπως η χρονοδρομολόγηση νημάτων, η εικονική μνήμη και οι διεργασίες [12].

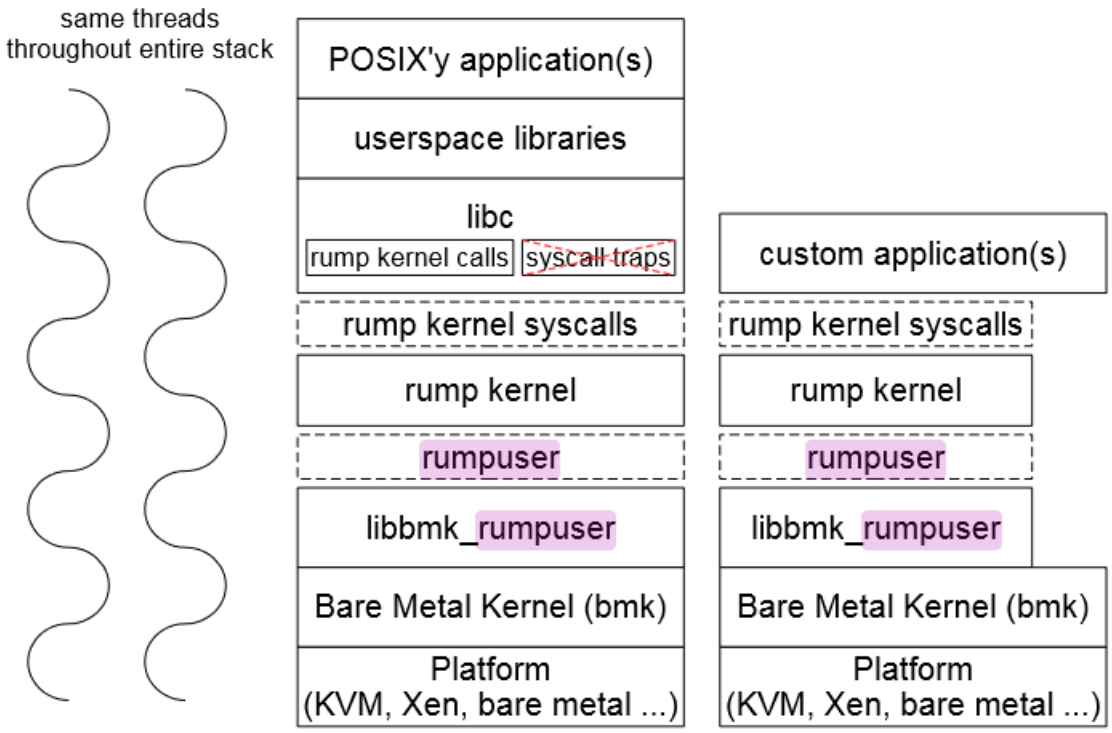
Τα Rump kernel προήλθαν από το χώρο του NetBSD, έχοντας αρχικό σκοπό να διευκολύνουν τον έλεγχο driver κατά την ανάπτυξή τους για το NetBSD. Το πετυχαίνουν αυτό επιτρέποντας την εκτέλεση κώδικα των driver στο user space χωρίς να χρειάζεται να εκτελεστεί ολόκληρο το Λειτουργικό Σύστημα [12]. Ωστόσο, με την πάροδο του χρόνου μια νέα περίπτωση χρήσης των Rump kernel έγινε εμφανής. Όπως φαίνεται και στο σχήμα 2.8, ένα Rump kernel μπορεί να ενσωματωθεί μαζί με τις κατάλληλες ρουτίνες και μια εφαρμογή για τη δημιουργία ενός unikernel. Η πλατφόρμα που χτίζεται πάνω σε αυτή την ιδέα ονομάζεται Rumprun.

2.3.5.2 Δομή του Rumprun

Σκοπός του Rumprun είναι να παρέχει τον απλότερο δυνατό ενδιάμεσο κώδικα στα rump kernel ώστε να υλοποιηθεί η unikernel πλατφόρμα. Οι driver αποτελούν τη συντριπτική πλειονότητα του τμήματος του unikernel που υποκαθιστά το Λειτουργικό Σύστημα. Έτσι, πολύ λίγος κώδικας χρειάζεται να προστεθεί, ενώ το υπόλοιπο έργο αναλαμβάνεται από τα rump kernel [12].

Εφόσον τα rump kernel έχουν αφαιρέσει τις λειτουργίες ενός κανονικού πυρήνα πέρα από τους driver, χρειάζεται να προσθέσουμε έναν πραγματικό πυρήνα στη στοίβα λογισμικού του Rumprun ώστε να επαναπροσθέσει αυτή τη λειτουργικότα. Το ρόλο αυτό καλύπτει το Bare Metal Kernel, ή συντόμως bmk, το οποίο αναλαμβάνει τη διαδικασία εκκίνησης (bootstrap),

τη δημιουργία νημάτων, τη χρονοδρομολόγηση, τις διακοπές και τη διαχείριση μνήμης στο επίπεδο σελιδών [12].



Σχήμα 2.9: Στοίβα Λογισμικού Rumprun [12]

Επιπλέον, τα rump kernel ορίζουν μια διεπαφή υπερκλήσεων (hypercall) η οποία χρησιμοποιείται για να έχουν πρόσβαση στους πόρους της πλατφόρμας που τα φιλοξενεί, στην περίπτωσή μας το bmk. Η διεπαφή αυτή ονομάζεται rumpuser και απαιτείται υλοποίηση αυτής που να συνδέει τον host με το rump kernel. Χτίζεται, λοιπόν, αυτή η διεπαφή πάνω από το bmk και παρέχει τις απαραίτητες υπερκλήσεις για να λειτουργήσει το Rumprun [12].

Όσον αφορά στο τμήμα της εφαρμογής, το Rumprun σχεδιάστηκε με ρητό σκοπό την υποστήριξη του POSIX, ένα σύνολο προτύπων σχεδιασμένο για τη διατήρηση συμβατότητας μεταξύ διαφορετικών λειτουργικών συστημάτων [51]. Πολλές υπάρχουσες POSIX εφαρμογές μπορούν να χτιστούν ως unikernel και να εκτελεστούν όπως είναι χωρίς αλλαγές. Σε άλλες περιπτώσεις, περιορισμοί του Rumprun μπορεί να επιβάλουν τροποποιήσεις, με παρόμοιο τρόπο που μπορεί να απαιτείται ενίοτε τροποποίηση του κώδικα για να εκτελείται ανάμεσα σε διαφορετικές συμβατικές UNIX πλατφόρμες. Στην ιδανική περίπτωση, όμως, αυτό αποφεύγεται [12].

Για να επιτευχθεί αυτό, το Rumprun παρέχει μία "user space" στοίβα πάνω από το rump kernel, η οποία είναι ουσιαστικά το ίδιο user space περιβάλλον το οποίο διατίθεται σε ένα κανονικό POSIX λειτουργικό σύστημα. Στο παρασκήνιο, το Rumprun φροντίζει οι κλήσεις συστήματος να καλούν τις αντίστοιχες συναρτήσεις του rump kernel [12].

Η επιβολή μίας πλήρους user space στοίβας, βεβαίως, έχει το μειονέκτημα ότι αυξάνει

το αποτύπωμα μνήμης (memory footprint) του εκτελέσιμου. Αυτό είναι ιδιάιτερα σημαντικό στον τομέα των unikernel όπου επιζητείται η ελαχιστοποίηση αυτού του αποτυπώματος και η αφαίρεση μη απαραίτητων βιβλιοθηκών. Για αυτόν ακριβώς το λόγο, το Rumprun προσφέρει έναν διαφορετικό τρόπο λειτουργίας κατά τον οποίο δεν παρέχεται το POSIX περιβάλλον. Όπως φαίνεται στο σχήμα 2.9, με αυτόν τον τρόπο προσαρμοσμένες εφαρμογές μπορούν να εκτελεστούν. Αυτές επικοινωνούν με το στρώμα του Λειτουργικού Συστήματος κατευθείαν και μπορούν να ελαχιστοποιήσουν το αποτύπωμά τους [12].

Υποστηρίζονται εφαρμογές γραμμένες σε ένα εύρος γλωσσών προγραμματισμού, συμπεριλαμβανομένων των εξής: C, C++, Erlang, Go, Java, Javascript (node.js), Python, Ruby και Rust [13].

2.3.5.3 Πλατφόρμες του Rumprun

Το Rumprun διακρίνει μεταξύ δύο τύπους πλατφορμών στις οποίες μπορεί να εκτελεστεί: την hw πλατφόρμα και τη Xen πλατφόρμα [13].

- **hw:** Η πλατφόρμα hardware προορίζεται για ενσωματωμένα συστήματα και το Cloud. Μπορεί να λειτουργήσει σε bare metal περιβάλλον, αλλά διατίθεται και για εικονικοποίηση, υποστηρίζοντας virtio driver και το KVM. Υποστηρίζονται οι αρχιτεκτονικές επεξεργαστή x86_32 και x86_64. Θεωρητικά υπάρχει μια πρώιμη υποστήριξη για την αρχιτεκτονική ARM, αλλά προηγούμενες μελέτες έχουν δείξει ότι δεν προκύπτει κάτι τέτοιο από τα πρακτικά δεδομένα. [12, 16].

- **Xen:** Επιτρέπει την εκτέλεση επί τον Xen hypervisor ως ένας paravirtualised guest. Παρέχει βελτιστοποιήσεις για το Xen και υποστήριξη για λειτουργίες εικονικοποίησης που δεν είναι διαθέσιμες στην hw πλατφόρμα. Η Xen πλατφόρμα είναι συμβατή με τα εργαλεία xl καθώς και το Amazon EC2 cloud. Υποστηριζόμενες αρχιτεκτονικές είναι οι x86_32 και x86_64.

Επιπλέον των παραπάνω, έχει γίνει ανεξάρτητα μια προσπάθεια στο πλαίσιο των Nabla Container να επιτρέψουν στο Rumprun να εκτελεστεί στο περιβάλλον του Solo5 [15].

2.3.6 OSv

Ένα ακόμη παράδειγμα unikernel έιναι το OSv, ένα λειτουργικό σύστημα γραμμένο σε C++11 που σχεδιάστηκε για χρήση σε εικονικές μηχανές στο Cloud. Στόχος του OSv είναι να εκτελεί υπάρχουσες εφαρμογές του Cloud (Linux εκτελέσιμα) και να το κάνει πιο αποτελεσματικά από ό,τι το ίδιο το Linux. Ταυτοχρόνως εξερευνεί νέα API για βελτιστοποιημένες εφαρμογές με ακόμη καλύτερη απόδοση. Πιο συγκεκριμένα το OSv ακολουθεί τη λογική των microVM, επιχειρώντας να μειώσει τις επιβαρύνσεις μνήμης και χρόνου εκκίνησης του Λειτουργικού Συστήματος τόσο ώστε η εκκίνηση νέου VM να είναι βιώσιμη εναλλακτική στην επαναδιαμόρφωση ενός υπάρχοντος VM [34].

Ο σχεδιασμός του OSv σε όλα τα σημεία επιδιώκει τους στόχους της βελτιστοποίησης και συμβατότητας. Σαν unikernel, εκτελεί μόνο μία εφαρμογή ανά VM και χρησιμοποιεί ενιαίο

χώρο διευθύνσεων. Σε αντίθεση με το Rumprun, όμως, χρησιμοποιεί εικονική μνήμη για αυτό τον χώρο διευθύσεων, επιτρέποντάς του να υποστηρίξει εφαρμογές που χρειάζονται paging και memory mapping [34].

Για να τρέχει την εκάστοτε εφαρμογή, ο πυρήνας του OSv περιέχει έναν δυναμικό συνδέτη αρχείων ELF (ELF dynamic linker). Όταν η εφαρμογή καλεί συναρτήσεις από το Linux ABI (application binary interface — διεπαφή δυαδικού εφαρμογής), ο linker παραπέμπει σε συναρτήσεις του OSv kernel. Ακόμη και οι κλήσεις συστήματος του Linux αντιμετωπίζονται σαν απλές συναρτήσεις, αποφεύγοντας τα κόστη ειδικών κλήσεων. Έτσι, ο linker μπορεί να δέχεται κώδικα για Linux αρκεί να είναι δυναμικά συνδεδεμένος. Άρα, το OSv παρέχει υποστήριξη του POSIX, αφού τα Linux API που χρησιμοποιούν αυτές οι εφαρμογές είναι υπερσύνολο των POSIX API [34].

Ωστόσο, τα Linux API όντας σχεδιασμένα για λειτουργικά συστήματα γενικού σκοπού, τα οποία είναι πολύ διαφορετικά από ένα unikernel, έχουν επιβαρύνσεις οι οποίες δεν είναι απαραίτητες στο OSv. Για αυτό το λόγο, επιπλέον το OSv επιχειρεί να εξερευνήσει και νέα API τα οποία να βελτιώνουν την επίδοση των εφαρμογών εκμεταλλευόμενα τη δομή του OSv. Για να μην απαιτείται, όμως, η τροποποίηση υπάρχοντων εφαρμογών ή η δημιουργία καινούργιων, το OSv εστιάζει στην επίδοση των περιβαλλόντων χρόνου εκτέλεσης (runtime environments), όπως το Java Virtual Machine (JVM). Βελτιστοποιόντας αυτό, όλες οι εφαρμογές που εκτελεί οφελούνται επίσης [34].

Όσον αφορά στους driver, επειδή το OSv προορίζεται για εικονικές μηχανές δεν είναι αναγκασμένο να υλοποιήσει driver για όλο το εύρος φυσικών συσκευών που χρησιμοποιούνται στους υπολογιστές. Περιορίζεται στην υλοποίηση driver για ένα μικρό σύνολο συσκευών που συνήθως εξομοιώνονται από τους hypervisor (π.χ. πληκτρολόγιο, VGA, σειριακή θύρα), καθώς και μερικών paravirtual driver για αυξημένη απόδοση (π.χ. virtio) [34].

Το OSv υποστηρίζει τις αρχιτεκτονικές x86_64 και aarch64 και πολλές γλώσσες προγραμματισμού, όπως Java, Python, Javascript (node.js), Ruby, Erlang, C, C++, Golang και Rust. Είναι επίσης συμβατό με πολλούς hypervisor, συμπεριλαμβανομένων των QEMU-/KVM, Firecracker, Cloud Hypervisor, Xen, VMWare, VirtualBox και Hyperkit [33].

Αρχικά το OSv δεν υποστήριζε το Firecracker και χρειάστηκε να προσαρμοστεί ώστε να γίνουν συμβατά. Η υλοποίηση του έργου αυτού συμπεριέλαβε τρία κύρια μέρη: τροποποίηση της διαδικασίας εκκίνησης του OSv ώστε να μπορέσουν να επικοινωνήσουν τα διαφορετικά πρωτόκολλα που χρησιμοποιούν το Firecracker και το OSv (Linux/x86 και Multiboot) και γενικως να γεφυρώσει τη διαφορετική λογική που ακολουθούν τα δύο προγράμματα· εκσυγχρονισμός και εμπλουτισμός του virtio ώστε να είναι συμβατό με την έκδοση που χρησιμοποιεί το Firecracker· τροποποίηση ορισμένων σημείων του OSv ώστε να μπορεί να λειτουργήσει χωρίς το ACPI αν αυτό δεν είναι διαθέσιμο. Ως αποτέλεσμα, η χρήση του OSv σε συνδυασμό με το Firecracker κατάφερε ακόμη καλύτερες επιδόσεις, έχοντας χρόνο εκκίνησης στα 5 ms και χρήση μνήμης στα 11 MB [33, 35].

Κεφάλαιο 3

Υλοποίηση

3.1 Εισαγωγή

Όπως είδαμε στα προηγούμενα κεφάλαια, τα microVM και τα unikernel είναι και τα δύο τεχνολογίες που έχουν αναπτυχθεί με κύριο σκοπό τη χρήση στο Cloud. Ως εκ τούτου, έχουν κοινές αρχές και στόχους που καθιστά τη συνεργατική λειτουργία τους θεμιτή. Η αρχική υπόθεση είναι ότι τα πλεονεκτήματα που και τα δύο προσφέρουν (μικρός χρόνος εκκίνησης, μικρό μέγεθος, ασφάλεια) μπορούν να αυξηθούν ακόμη πιο πολύ εάν τα χρησιμοποιήσουμε μαζί. Τα unikernel άλλωστε συνήθως αποσκοπούν σε εκτέλεση μέσα σε εικονικό περιβάλλον, όπως τα microVM.

Η ιδέα αυτή έχει ήδη δοκιμαστεί στο OSv, το οποίο τροποποιήθηκε για να γίνει συμβατό με το Firecracker, με θετικά αποτελέσματα [33]. Ακολουθώντας την ίδια λογική, στα πλαίσια αυτή της εργασίας, θα προσθέσουμε υποστήριξη για το Firecracker στο Rumprun unikernel. Στο κεφάλαιο αυτό, θα περιγράψουμε τα βήματα που χρειάστηκαν για να γίνει αυτό δυνατό.

Σημειώνεται εδώ ότι οι τροποποιήσεις του Rumprun γίναν στην πλατφόρμα hw, αφού υποστηρίζει το KVM που χρησιμοποιείται και από το Firecracker. Συγκεκριμένα, η εργασία δούλεψε πάνω στην αρχιτεκτονική x86_64, αλλά παρόμοια διαδικασία μπορεί να εφαρμοστεί και για την αρχιτεκτονική x86_32. Στη συνέχεια, όταν αναφερόμαστε στο Rumprun, θα εννοούμε την πλατφόρμα hw και αρχιτεκτονική x86_64.

Ο πλήρης κώδικας που παρήχθησε κατά τη διάρκεια αυτής της εργασίας έχει αναρτηθεί στο αποθετήριο `https://github.com/ggkanas/rumprun`.

3.2 Περιβάλλον ανάπτυξης

Η ανάπτυξη και δοκιμή της μεταφοράς (port) που επιχειρήσαμε έγινε σε λειτουργικό σύστημα Linux Mint 20.2 (Ubuntu 20.04, kernel 5.4.0-91-generic) και αρχιτεκτονική x86_64. Χρησιμοποιήθηκε ο μεταγλωττιστής gcc έκδοσης 7.5.0 (παρατηρήθηκε ότι με υψηλότερες εκδόσεις το κτίσιμο του Rumprun δεν ήταν επιτυχές). Χρησιμοποιήθηκε η έκδοση 0.24.2 του Firecracker και το πιο πρόσφατο commit του Rumprun που παρέχεται στο αποθετήριό του (repository).

3.3 Boot

Για να λειτουργήσει το Rumprun επί το Fircracker πρώτα έπρεπε να φροντίσουμε ότι εκκινεί σωστά και η εικονική μηχανή εισέρχεται στον κώδικα του Rumprun. Ωστόσο, το Rumprun υποστηρίζει μόνο το πρότυπο Multiboot, ενώ το Firecracker χρησιμοποιεί το Linux/x86 πρωτόκολλο [12, 35].

Τα δύο αυτά πρωτόκολλα περιγράφουν τρόπους εκκίνησης του πυρήνα και την επικοινωνία του με τον φορτωτή εκκίνησης (bootloader). Το Multiboot σχεδιάστηκε με σκοπό να το υιοθετήσουν πολλαπλά Λειτουργικά Συστήματα και πολλαπλοί bootloader, ώστε να είναι συμβατά μεταξύ τους. Έτσι, οποιοσδήποτε bootloader υποστηρίζει το Multiboot μπορεί να φορτώσει οποιοδήποτε Λειτουργικό Σύστημα συμμορφώνεται στο Multiboot. Το Multiboot έχει δύο έκδοσεις, την Multiboot1 και την Multiboot2. Το Rumprun υλοποιεί την πρώτη [44, 49].

Αντίθετα, το Linux/x86 πρωτόκολλο εκκίνησης, όπως καταλαβαίνει κανείς και από το όνομά του, είναι το πρωτόκολλο που χρησιμοποιεί συγκεκριμένα το Linux στην x86 αρχιτεκτονική για να φορτωθεί και να εκκινήσει [50].

Το Multiboot1 έχει τρεις βασικές απαιτήσεις για την υλοποίηση της διεπαφής του μεταξύ Λειτουργικού Συστήματος και bootloader: τη μορφή της εικόνας του Λειτουργικού Συστήματος, όπως τη βλέπει ο bootloader για να μπορέσει να τη φορτώσει σωστά· την κατάσταση της μηχανής (machine state) τη στιγμή που δίνεται ο έλεγχος στο Λειτουργικό Σύστημα· και τη μορφή των πληροφοριών που περνάει ο bootloader στο Λειτουργικό Σύστημα. Για την υλοποίησή μας σημαντικά είναι τα δύο τελευταία [49].

Πριν αρχίσει να τρέχει ο κώδικας του πυρήνα, ένας Multiboot bootloader φροντίζει να εισάγει το πρόγραμμα σε 32-bit λειτουργία και να θέσει τις τιμές μερικών καταχωρητών του επεξεργαστή. Από αυτούς μας ενδιαφέρουν οι eax, ο οποίος περιέχει μία ειδική τιμή (magic value) που επιβεβαιώνει ότι η φόρτωση έγινε από Multiboot1 bootloader, και ebx, ο οποίος περιέχει τον δείκτη προς τη δομή πληροφοριών Multiboot (multiboot_info). Αυτή η δομή έχει συγκεκριμένη μορφή και περιέχει ό,τι πληροφορίες θέλει να μεταφέρει ο bootloader στο Λειτουργικό Σύστημα [35, 49].

Στο Linux/x86 πρωτόκολλο, από την άλλη, χρησιμοποιείται μια δομή που ονομάζεται boot_params για τη μεταφορά δεδομένων, δείκτης στην οποία γράφεται στον καταχωρητή rsi. Επίσης, το Firecracker λειτουργεί μόνο με 64-bit εικόνες, οπότε σύμφωνα με τις απαιτήσεις του Linux/x86 φροντίζει να εισέλθει σε 64-bit λειτουργία πριν παραδώσει τον έλεγχο στον guest [35, 50].

Για να μπορέσουμε να χρησιμοποιήσουμε το Firecracker και το Rumprun μαζί, έπρεπε να τροποποιήσουμε τη διαδικασία boot του Rumprun ώστε να εισέρχεται κατευθείαν σε 64-bit λειτουργία και ύστερα να επεξεργάζεται τα δεδομένα της δομής boot_params ώστε να τα εμφανίζουμε στο Rumprun στη μορφή της δομής multiboot_info.

Για να το επιτύχουμε αυτό, πρέπει να τροποποιήσουμε τον κώδικα του Rumprun στο σημείο εισαγωγής (entry point) του. Αυτό βρίσκεται στο αρχείο platform/hw/arch/amd64/locore.S και είναι η ρουτίνα _start. Η ρουτίνα αυτή, μεταξύ άλλων, αποθηκεύει τον δείκτη του multi-

boot_info στη στοίβα, ελέγχει ότι η φόρτωση έγινε κατά το Multiboot1 (ελέγχει τον καταχωρητή eax), αρχικοποιεί τιμές σχετικές με την κονσόλα, εγκαθιστά τους πίνακες σελίδων, αλλάζει σε 64-bit λειτουργία και τέλος καλεί τη ρουτίνα _start64. Αυτή καλεί τη συνάρτηση x86_boot με παράμετρο τον δείκτη του multiboot_info και συνεχίζεται το boot.

Έτσι, προσθέσαμε στο locore.S μία νέα ρουτίνα, ονόματι _start64_from_linux. Αυτή θα αντικαταστήσει την _start ως το νέο entry point, αλλά λειτουργώντας κατευθείαν σε 64-bit. Σε αυτό το σημείο, επίσης, θα εισάγουμε κλήσεις που χρειάζεται να γίνουν μόνο όταν το Rumprun τρέχει σε Firecracker. Για να αλλάξουμε το entry point, τροποποιούμε τα αρχεία platform/hw/arch/amd64/kern.ldscript και το platform/hw/Makefile ώστε να κάνουν αναφορά στο _start64_from_linux αντί του _start.

Κώδικας 3.1: platform/hw/Makefile, γραμμές 64-69

```
${MAINOBJ}: ${OBJS} platformlibs
  ${CC} -nostdlib ${CFLAGS} ${LDFLAGS} -Wl,-r ${OBJS} -o $@ \
     -L${RROBJLIB}/libbmk_core -L${RROBJLIB}/libbmk_rumpuser \
     -Wl,--whole-archive -lbmk_rumpuser -lbmk_core -Wl,--no-whole-archive
  ${OBJCOPY} -w -G bmk_* -G rumpuser_* -G jsmn_* \
     -G rumprun_platform_rumpuser_init -G _start64_from_linux $@
```

Κώδικας 3.2: platform/hw/arch/amd64/kern.ldscript, γραμμές 1-3

```
OUTPUT_FORMAT("elf64-x86-64", "elf64-x86-64", "elf64-x86-64")
OUTPUT_ARCH(i386:x86-64)
ENTRY(_start64_from_linux)
```

Στη νέα ρουτίνα, δεν κάνουμε έλεγχο του Multiboot1 αφού δεν μας ενδιαφέρει αν και ξέρουμε ότι δε χρησιμοποιείται κατά τη φόρτωση. Επίσης αφαιρούμε την αλλαγή λειτουργίας σε 64-bit αφού βρισκόμαστε ήδη σε αυτή. Άλλαζουμε επίσης κάποιους από τους καταχωρητές και τις εντολές που τους χρησιμοποιούν στις 64-bit εκδοχές τους, π.χ. από `movl %cr4, %eax` σε `movq %cr4, %rax`.

Καλείται, ακόμη, η συνάρτηση extract_linux_boot_params (platform/hw/arch/amd64/vmlinux.c). Αυτή έχει σκοπό να εξάγει τα χρήσιμα δεδομένα της δομής boot_params και να τα ενσωματώσει στη δομή multiboot_info. Αποτελεί παραλλαγή της συνάρτησης που χρησιμοποιεί το OSv για τον ίδιο λόγο. Μεταφέρεται η γραμμή εντολών εκκίνησης καθώς και ο πίνακας των εγγραφών e820, οι οποίες περιγράφουν περιοχές μνήμης οι οποίες είναι ελεύθερες ή είναι κατοχυρωμένες για το BIOS. Επίσης εγγράφεται το πεδίο flags, το κάθε bit του οποίου είναι μια σημαία που ενδεικνύει αν κάποια δυνατότητα (feature) του Multiboot είναι διαθέσιμη. Θέτουμε σε 1 τα bit των πεδίων που γράψαμε, καθώς και το bit 0, διότι το απαιτεί το Rumprun (αν και δεν χρησιμοποιεί την αντίστοιχη δυνατότητα — πιθανώς εσφαλμένα να ελέγχει το bit 0 αντί για το bit 6, το οποίο αναφέρεται στον χάρτη μνήμης e820).

Επειδή η συνάρτηση extract_linux_boot_params θεωρεί ότι ο δείκτης του multibo-

ot_info έχει την τιμή 0x1000, αποθηκεύουμε αυτήν αντί για την τιμή του καταχωρητή ebx στη στοίβα (όπου η _start64 περιμένει να βρει την τιμή του δείκτη).

Κατά τα άλλα δεν αλλάζουμε τα σημεία της ρουτίνας που δεν απασχολούν την υλοποίησή μας. Μετά από αυτά τα βήματα, καθέστη δυνατό να εκκινήσει σωστά το Rumprun.

Κώδικας 3.3: Ρουτίνα _start64_from_linux

```
ENTRY(_start64_from_linux)
    cld
    mov $bootstack, %rsp

    /* _start64 looks for the multiboot info pointer in the boot stack
       extract_linux_boot_params assumes the pointer is at 0x1000 */
    push $0x1000

    /* save BIOS data area values */
    movw $0x3f8, %bx
    movw %bx, bios_com1_base
    movw $0x0, %bx
    movw %bx, bios_crtc_base
    movw bios_com1_base, %bx

   lgdt (gdt64_ptr)

1: movl $0x18, %eax
   movl %eax, %ds
   movl %eax, %es
   movl %eax, %ss

   xorl %eax, %eax
   movl %eax, %fs
   movl %eax, %gs

    /* 1: enable pae and sse */
    movq %cr4, %rax
    orl $(CR4_OSXMMEXCPT|CR4_OSFXSR|CR4_PAE), %eax
    movq %rax, %cr4

    /* 3: load pml4 pointer */
    movq $cpu_pml4, %rax
    movq %rax, %cr3

    /* 4: enable paging */
    movq %cr0, %rax
    orl $(CR0_PG|CR0_WP|CR0_PE), %eax
    movq %rax, %cr0
```

```
    mov %rsi, %rdi
    call extract_linux_boot_params

    jmp _start64
    hlt
END(_start64_from_linux)
```

3.4 Σειριακή Έξοδος

Σε αυτό το σημείο παρατηρήθηκε ένα άλλο πρόβλημα. Όταν τρέχαμε το Rumprun πάνω από Firecracker, δεν εμφανίζονταν καθόλου αποτελέσματα στην οθόνη. Ύστερα από μελέτη, ανακαλύψαμε ότι κατά τη διάρκεια του boot το οποίο εκτελεί το Rumprun, για έξοδο στην οθόνη υποστηρίζονται η κονσόλα VGA και η σειριακή κονσόλα, με προεπιλογή στην πρώτη. Από την άλλη, το Firecracker με τον περιορισμένο αριθμό εξομοιομένων συσκευών, υποστηρίζει μόνο τη σειριακή κονσόλα.

Έτσι, σε πρώτο βήμα χρειάστηκε να αλλάξουμε την κονσόλα που χρησιμοποιείται στη σειριακή. Η επιλογή αυτή γίνεται στο αρχείο platform/hw/arch/x86/cons.c, όπου εάν η μεταβλητή bios_crtc_base είναι 0 και η σειριακή κονσόλα είναι διαθέσιμη, προτιμάται αυτή. Η bios_crtc_base παίρνει την τιμή της στο _start, οπότε μπορούμε εμείς αντί αυτού στο _start64_from_linux να φροντίσουμε να παίρνει τιμή 0, ώστε να προτιμάται πάντα η σειριακή κονσόλα.

Σε δεύτερο βήμα, παρατηρήσαμε ότι η διεύθυνση της σειριακής θύρας που χρησιμοποιούταν δεν ήταν η κατάλληλη. Το Firecracker, κατά τη σύμβαση, ορίζει σειριακές θύρες στις I/O (Input/Output — Είσοδος/Έξοδος) διευθύνσεις 0x3f8, 0x2f8, 0x3e8 και 0x2e8 (src/vmm/src/device_manager/legacy.rs). Το Rumprun ορίζει τη σειριακή θύρα που χρησιμοποιεί μέσω της μεταβλητής bios_com1_base, που επίσης παίρνει την τιμή της στο locore.S. Φορτώνει την τιμή της από τη διεύθυνση 0x400, αναμένοντας ότι το BIOS θα έχει φροντίσει αυτή να είναι ορθή. Ωστόσο, το Firecracker δεν κάνει κάτι τέτοιο. Γράφοντας την τιμή 0x3f8 κατευθείαν στη μεταβλητή bios_com1_base μπορούμε να το διορθώσουμε αυτό και επιτυγχάνεται η σωστή λειτουργία της σειριακής κονσόλας. Ως αποτέλεσμα, έχουμε και έξοδο στην οθόνη την οποία μπορούμε να χρησιμοποιήσουμε για να παρακολουθήσουμε τη διαδικασία εκκίνησης του Rumprun.

3.5 Προνομιακές Εντολές

Ένα ενδιαφέρον σημείο είναι η αρχικοποίηση μερικών προστατευμένων καταχωρητών η οποία πραγματοποιείται και από το Firecracker και από το Rumprun. Η αρχικοποίηση αυτή αφορά κυρίως τρία στοιχεία: τη σελιδοποίηση (page tables), τον πίνακα περιγραφέων διακοπών (interrupt descriptor table) και τον πίνακα καθολικών περιγραφέων (global descriptor

table). Αυτά είναι ευαίσθητα χαρακτηριστικά, τα οποία επιτρέπεται να διαμορφωθούν μόνο σε προνομιακή λειτουργία του επεξεργαστή (Ring 0).

Σελιδοποίηση είναι ένα σύστημα το οποίο επιτρέπει σε μια διεργασία να έχει πρόσβαση σε έναν πλήρη εικονικό χώρο διευθύνσεων, χωρίς να χρειάζεται αντίστοιχο μέγεθος φυσικής μνήμης να υπάρχει πραγματικά ή να είναι διαθέσιμο [45]. Όταν μία διεργασία χρησιμοποιεί μια εικονική διεύθυνση, δεν είναι απαραίτητο ότι η αντίστοιχη φυσική διέυθυνση θα έχει την ίδια τιμή. Η αντιστοιχία μεταξύ εικονικών και φυσικών διευθύνσεων και ποιες περιοχές διευθύνσεων είναι διαθέσιμες καθορίζεται από τους πίνακες σελίδων. Η σελιδοποίηση ελέγχεται από τους προστατευμένους καταχωρητές ελέγχου (control register) cr0, cr3 και cr4. Το Rumprun και το Firecracker το καθένα ξεχωριστά κατασκευάζουν τους δικούς τους πίνακες σελίδων και πρέπει να αποφασίσουμε ποιον από τους δύο κατάλογους σελίδων θα χρησιμοποιήσουμε.

Ο interrupt descriptor table ή idt είναι ένας πίνακας στον οποίο αποθηκεύονται οι διαφορετικές διακοπές που μπορούν να γίνουν. Περιέχει διάφορα δεδομένα, αλλά το πιο βασικό είναι η διεύθυνση της ρουτίνας που καλείται για να χειριστεί τη διακοπή όταν αυτή προκύψει (interrupt service routine). Ο επεξεργαστή ενημερώνεται για τη θέση του idt από την προνομιακή εντολή lidt [46]. Το Firecracker τοποθετεί τον πίνακα αυτόν στη διεύθυνση 0x520, αλλά το Rumprun δηλώνει μεταβλητή για τον ίδιο σκοπό, της οποίας η θέση δεν είναι αναγκαστικά η ίδια.

Ο global descriptor table ή gdt είναι ένας πίνακας στην x86 αρχιτεκτονική ο οποίος περιγράφει τα τμήματα μνήμης memory segment ενός προγράμματος. Μερικά παραδείγματα τμημάτων είναι το τμήμα κώδικα code segment, το τμήμα δεδομένων data segment και το τμήμα της στοίβας stack segment. Μεταξύ άλλων, οι εγγραφές του gdt περιέχουν τη διεύθυνση και το μέγεθος του τμημάτος που περιγράφουν. Ο επεξεργαστής ενημερώνεται για τη θέση του πίνακα με την προνομιακή εντολή lgdt [47, 48]. Το Firecracker τοποθετεί τον gdt στη διεύθυνση 0x500, ενώ το Rumprun δεν ορίζει συγκεκριμένη διεύθυνση για αυτόν.

Θα ήταν θεμιτό αυτή η αρχικοποίηση να γίνεται μόνο από το Firecracker, ώστε να γίνει εφικτό να εκτελούνται Rumprun guest σε μη προνομιακή λειτουργία κάνοντάς τους έτσι ακόμη πιο ασφαλείς. Για αυτό το λόγο, κάναμε δοκιμές να αφαιρέσουμε τη χρήση των προνομιακών αυτών εντολών από το Rumprun και να δούμε κατά πόσο μπορεί να λειτουργήσει σωστά το Rumprun εξαρτώμενο από τη διαμόρφωση που κάνει το Firecracker.

Η χρήση της εντολής lgdt και ο χειρισμός των control register γίνεται στη ρουτίνα _start, οπότε για τον πειραματισμό μας τα αφαιρούμε από τη ρουτίνα _start64_from_linux. Οι εντολές lidt και ltr, η οποία επίσης σχετίζεται με τον gdt, χρησιμοποιούνται στη συνάρτηση cpu_init που καλείται αργότερα κατά τη διάρκεια του boot (`platform/hw/arch/amd64/machdep.c`). Αφαιρούμε και την κλήση αυτών.

Το αποτέλεσμα αυτής της παραλλαγής όταν εκτελέστηκε αρχικά φάνηκε φυσιολογικό. Το unikernel εκκινούσε, έφτανε στην εκτέλεση της εφαρμογής και τελείωνε φυσιολογικά. Όταν δοκιμάσαμε, όμως, πιο περίπλοκες περιπτώσεις χρήσης εντοπίστηκαν προβλήματα. Δοκιμάσαμε να χρησιμοποιήσουμε MMIO επικοινωνία, η οποίο κάνει χρήση υψηλών διευθύνσεων (συνήθως αρχίζει από τη διεύθυνση 0xd0000000) και η εκτέλεση διακόπηκε μόλις το πρόγραμμα προσπάθησε να τις προσπελάσει.

Διαπιστώσαμε ότι οι πίνακες σελίδων που εγκαθιστά το Firecracker ορίζουν μόνο το πρώτο 1 GiB ως διαθέσιμη φυσική μνήμη, ενώ το Rumprun ορίζει 4 GiB, τα οποία και απαιτούνται για την προσπέλαση των παραπάνω διευθύνσεων. Εφόσον το Rumprun περιμένει 4 GiB φυσικής μνήμης ενδεχομένως να προκύπτουν και άλλα προβλήματα στη γενική περίπτωση.

Δοκιμαστικά, τροποποιήσαμε τον κώδικα του Firecracker ώστε να ορίζει τις απαιτούμενες σελίδες. Χρειάστηκε επίσης να ορίσουμε στο config file του, το σωστό ποσό μνήμης (4096 MiB). Αυτό επιβεβαίωσε την υπόθεσή μας ότι οι ελλιπείς πίνακες σελίδων ήταν η πηγή του προβλήματος, αφού το Rumprun εκτελέστηκε φυσιολογικά μετά από αυτές τις αλλαγές.

Επιχειρήσαμε ως συμβιβασμό να αφήσουμε την αρχικοποίηση της σελιδοποίησης στο Rumprun και τα υπόλοιπα να τα εκτελέσουμε στο Firecracker. Ωστόσο, αυτό μας δημιουργεί ένα άλλο πρόβλημα διότι στον κατάλογο σελίδων του Rumprun τα πρώτα 4 KiB (έως τη διεύθυνση 0xFFF) είναι μόνο για ανάγνωση, ενώ ο πίνακες idt, στον οποίο θέλουμε ύστερα να γράψουμε δεδομένα για την εγκατάσταση των διακοπών, βρίσκεται στη θέση μνήμης 0x520.

Ένα άλλο θέμα που παρατηρήσαμε είναι ότι το Rumprun θέτει μερικές σημαίες των control register παραπάνω από ό,τι το Firecracker. Χωρίς να τροποποιήσουμε το δεύτερο, αναγκαζόμαστε να γράψουμε τους καταχωρητές αυτούς και στο Rumprun ώστε να έχουν τις τιμές που αυτό περιμένει.

Σκοπός της διπλωματικής αυτής ήταν να τροποποιήσουμε το Rumprun χωρίς να κάνουμε αλλαγές στο Firecracker. Ωστόσο καταλήγουμε στο συμπέρασμα πως αν θέλουμε να περιορίσουμε τις προνομιακές εντολές μόνο στο Firecracker, θα πρέπει να τροποποιηθεί και αυτό. Στην έκδοση του Rumprun που παρουσιάζεται στο αποθετήριό μας, οι προνομιακές εντολές αφήνονται ως έχουν.

3.6 Ολοκλήρωση

Για την ολοκλήρωση του έργου έγιναν επίσης μερικές ακόμη μικρές τροποποιήσεις ώστε να μην διακόπτεται η ροή του προγράμματος. Για παράδειγμα σε συγκεκριμένα σημεία, παρατηρήθηκε ότι η αρχικοποίηση των δομών (struct) της C έπρεπε να αλλάξει ώστε τα μέλη της δομής να παίρνουν τις τιμές τους ένα-ένα.

Χρειάστηκε επιπλέον να προσθέσουμε στο σενάριο build-rr.sh, το οποίο χτίζει την εργαλειοθήκη (toolchain) που χρειάζεται για την κατασκευή του Rumprun unikernel, την επιλογή να στοχεύσει για την πλατφόρμα του Firecracker ή για την πλατφόρμα του QEMU. Για αυτόν τον λόγο, προσθέσαμε την παράμετρο -f, η οποία μας επιτρέπει να διαλέξουμε την Firecracker εκδοχή, στην οποία αλλάζει το σημείο εισαγωγής.

Κώδικας 3.4: Παράδειγμα διαμόρφωσης του firecracker

```
{
  "boot-source": {
    "kernel_image_path": "/file/path/to/hello.bin",
    "boot_args": "console=ttyS0 reboot=k panic=1"
  },
```

```
  "drives": [],
  "machine-config": {
    "vcpu_count": 1,
    "mem_size_mib": 512,
    "ht_enabled": false
  }
}
```

Για να γίνει η εκτέλεση στην πράξη δημιουργούμε ένα config file του Firecracker στο οποίο δίνουμε ως παράμετρο το εκτελέσιμο αρχείο που παράγει το Rumprun. Το Firecracker περιμένει μία εικόνα ενός ασυμπίεστου 64-bit ELF (Executable and Linking Format, ένας τύπος εκτελέσιμου) πυρήνα [32] και όπως βλέπουμε στην εικόνα 3.1 το εκτελέσιμο που παράγει το Rupmrun έχει ακριβώς αυτή τη μορφή.

```
~/code/diplo/rumprun2/rumprun $ file demo/hello
demo/hello: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), with debug info, not stripped
```

Σχήμα 3.1: Τύπος αρχείου ενός Rumprun unikernel

Ένα παράδειγμα εκτέλεσης φαίνεται παρακάτω:

```
~/code/diplo/firecracker $ ./firecracker --api-sock /tmp/firecracker.socket --config-file config.json --level Info  --log-path log
rump kernel bare metal bootstrap

x86_initclocks(): Using PV clock for timekeeping
console=ttyS0 reboot=k panic=1
Copyright (c) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005,
    2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016
    The NetBSD Foundation, Inc.  All rights reserved.
Copyright (c) 1982, 1986, 1989, 1991, 1993
    The Regents of the University of California.  All rights reserved.

NetBSD 7.99.34 (RUMP-ROAST)
total memory = 253 MB
timecounter: Timecounters tick every 10.000 msec
timecounter: Timecounter "clockinterrupt" frequency 100 Hz quality 0
cpu0 at thinair0: rump virtual cpu
root file system type: rumpfs
kern.module.path=/stand/amd64/7.99.34/modules
mainbus0 (root)
pci0 at mainbus0 bus 0
pci0: i/o space, memory space enabled, rd/line, rd/mult, wr/inv ok
timecounter: Timecounter "bmktc" frequency 1000000000 Hz quality 100
mounted tmpfs on /tmp
rumprun: could not find start of json.  no config?

=== calling "rumprun" main() ===

Hello world from Rumprun through Firecracker!

=== main() of "rumprun" returned 0 ===

=== _exit(0) called ===
rump kernel halting...
syncing disks... done
unmounting file systems...
unmounted tmpfs on /tmp type tmpfs
unmounted rumpfs on / type rumpfs
unmounting done
halted
```

Σχήμα 3.2: Παράδειγμα Εκτέλεσης Rumprun με το Firecracker

3.7 Περιορισμοί

Παρατηρήθηκαν οι ακόλουθοι περιορισμοί στις ικανότητες του Rumprun όταν εκτελείται μέσω του Firecracker:

- Δεν υποστηρίζεται το virtio, επειδή το Rumprun και το Firecracker υλοποιούν διαφορετικές εκδοχές. Για αυτό το θέμα θα μιλήσουμε παραπάνω στην επόμενη ενότητα.

- Παρατηρήθηκε ότι η είσοδος από το πληκτρολόγιο δεν δουλεύει σωστά. Δοκιμάστηκαν οι συναρτήσεις scanf και getchar, καθώς και η κλήση συστήματος read. Αντί να μπλοκάρει το πρόγραμμα μέχρι να δοθεί είσοδος από το πληκτρολόγιο, αναγνώστηκε από μόνη της κάποια άσχετη τιμή.

- Όταν τελειώνει η εκτέλεση της εφαρμογής και το Rumprun τερματίζει τη λειτουργία του, η διεργασία του firecracker δεν τερματίζει, αλλά παραμένει στο σύστημα μέχρι να σκοτωθεί από εξωτερικό σήμα. Αυτό το πρόβλημα δεν υπάρχει στο QEMU, άρα κάπως προκύπτει από τον συνδυασμο Rumprun και Firecracker.

3.8 Υποστήριξη Virtio

Η πρώτη επίσημη έκδοση του virtio είναι η 1.0 και η τελευταία η 1.2, ωστόσο μέχρι την οριστικοποίηση των προδιαγραφών της έκδοσης 1.0, υπήρξαν νωρίτερα προσχέδια και τα οποία έχουν ευρέως υλοποιηθεί. Οι προδιαγραφές του virtio ονομάζουν τέτοιες πρώιμες συσκευές, προγράμματα οδήγησης (driver) και διεπαφές ως παλαιού τύπου (legacy) [25].

Το virtio απαιτεί ένα στρώμα μεταφοράς τον ρόλο του οποίου μπορεί να καλύψει ένα εκ των τριών bus: PCI, MMIO, Channel I/O [25]. Το στρώμα μεταφοράς φροντίζει να εντοπιστεί η virtio συσκευή όταν είναι παρούσα καθώς και καθιστά δυνατή τη μεταφορά πληροφορίας μεταξύ guest και host. Για παράδειγμα, στην περίπτωση Virtio over MMIO (Virtio πάνω από MMIO), χρησιμοποιείται ένα εύρος διευθύνσεων μνήμης στο οποίο μπορούν να γράψουν και να διαβάσουν και ο guest και ο host επιτρέποντάς τους έτσι να επικοινωνήσουν.

Το Firecracker υλοποιεί την έκδοση 1.2 του virtio και χρησιμοποιεί το στρώμα μεταφοράς Virtio over MMIO. Αντίθετα το Rumprun υλοποιεί legacy έκδοση του virtio και χρησιμοποιεί Virtio over PCI. Το Firecracker χρησιμοποιεί το virtio για να παρέχει συσκευές δίσκου (block) και δικτύου στα microVM του, το οποίο σημαίνει ότι αυτή η ασυμβατότητα στο virtio εμποδίζει τη χρήση τέτοιων συσκευών και άρα περιορίζει σημαντικά τις εφαρμογές που μπορούν να εκτελεστούν με το συνδυασμό Rumprun-Firecracker.

Κατά τη διάρκεια της διπλωματικής, έγινε προσπάθεια να εκσυγχρονίστει η υλοποίηση του Rumprun και να εμπλουτιστεί με υποστήριξη για το MMIO. Ωστόσο, το έργο αυτό αποδείχθηκε να είναι πιο μεγάλο από ό,τι επέτρεπαν τα πλαίσια της εργασίας αυτής και δεν ολοκληρώθηκε. Παρακάτω θα αναλυθούν οι προσπάθειες που έγιναν για τη λύση αυτού του προβλήματος.

3.8.1 Εκσυγχρονισμός του Virtio

Σε πρώτη φάση, υπήρξε η ιδέα να μιμηθούμε τη διαδικασία που ακολούθησε το OSv ώστε να υλοποιήσει το σύγχρονο πρωτόκολλο virtio και το στρώμα μεταφοράς MMIO [35]. Το OSv είναι γραμμένο σε C++, οπότε είναι σχετικά απλό να μεταφερθεί κώδικας από αυτό στο Rumprun, όσον αφορά στην προσθήκη MMIO. Ωστόσο, η τροποποίηση του ήδη υπάρχοντος κώδικα των συσκευών virtio και PCI ώστε τα νέα πρωτόκολλα να ενσωματωθούν στο σύστημα των Rump kernel θα αποτελούσε πιο περίπλοκο έργο. Για αυτό το λόγο, αναζητήσαμε διαφορετικές λύσεις.

Αυτό που ανακαλύφθηκε στη συνέχεια είναι ότι το NetBSD και άρα τα Rump kernel έχουν ήδη υλοποιήσει τα παραπάνω πρωτόκολλα, αλλά σε νεότερες εκδόσεις από ό,τι αυτή την οποία χρησιμοποιεί το Rumprun (7.99.34). Για αυτό το λόγο, δοκιμάστηκε να αναβαθμιστεί η έκδοση του NetBSD (Rump kernel) που χρησιμοποιείται ώστε να αξιοποιηθεί ο νέος κώδικας του virtio. Η προσπάθεια αυτή είχε μερική επιτυχία, αλλά τελικά εγκαταλείφθηκε λόγω της περιπλοκότητας ελέγχου της σωστής λειτουργίας.

Αντί αυτού επιλέξαμε να εισάγουμε μόνο τον κώδικα που χρειαζόμασταν. Η έκδοση 10 του NetBSD περιέχει υλοποιήσεις του Virtio over PCI στη σύγχρονη και την παλαιά έκδοση και του Virtio over MMIO μόνο στην παλαιά έκδοση. Για αυτό το λόγο, χρειάζεται επιπλέον να εμπλουτίσουμε τον κώδικα του MMIO ώστε να ακολουθεί τη διαδικασία της έκδοσης 1 (δεν είναι απολύτως απαραίτητο να υλοποιήσουμε όλες τις λειτουργίες της έκδοσης 1.2, αφού οι προσθήκες της σε σχέση με την 1.0 και 1.1 στο κομμάτι που μας αφορά είναι κυρίως προαιρετικές). Για αυτό τον σκοπό, προσθέσαμε το φάκελο `sys/dev/virtio` που υλοποιεί τη δίαυλο MMIO, τροποποιώντας το κατάλληλα, αντικαταστήσαμε τα αρχεία του virtio driver και για λόγους συμβατότητας πραγματοποιήσαμε μικρές αλλαγές στους driver των Virtio συσκευών (π.χ. if_vioif, ld_virtio).

Με τη συγκεκριμένη διαρρύθμιση, το Rumprun μπόρεσε να μεταγλωττιστεί και να εκτελεστεί. Ωστόσο για να αξιοποιήσουμε τον νέο κώδικα κατάλληλα, χρειαζόταν παραπάνω έργο. Αυτή τη στιγμή στο NetBSD το Virtio over MMIO χρησιμοποιείται μόνο στην ARM αρχιτεκτονική μέσω είτε του fdt είτε του acpi. Για την αρχιτεκτονική amd64 στην οποία εργαζόμαστε, θα πρέπει να γράψουμε τον δικό μας ενδιάμεσο κώδικα και να φροντίσουμε ώστε οι κατάλληλες συσκευές να αρχικοποιηθούν. Το θέμα αυτό θα συζητηθεί στα επόμενα τμήματα.

3.8.2 Επικοινωνία με Firecracker

Το Firecracker ως VMM αναλαμβάνει το ρόλο να παρέχει τις εξομοιώμενες συσκευές Net, Block και Vsock μέσω του virtio [31]. Για να είναι δυνατή η επικοινωνία με τις συσκευές virtio, πρέπει το Firecracker να γνωστοποιήσει τις λεπτομέρειές τους στον guest (Rumprun). Εφόσον το πρωτόκολλο μεταφοράς είναι το MMIO τα δεδομένα αυτά είναι η φυσική διεύθυνση μνήμης της αρχής και το μέγεθος του εύρους μνήμης στο οποίο έχουν απεικονιστεί η είσοδος και η έξοδός της (I/O). Επίσης δίνεται ο αριθμός της διακοπής (interrupt) που χρησιμοποιείται για τις virtio ειδοποιήσεις από τη συσκευή.

Ο τρόπος με τον οποίο περνάει τα δεδομένα αυτά το Firecracker στον guest είναι προ-

σαρτίζοντάς τα στη γραμμή εντολών του boot. Έτσι, από την πλευρά του Rumprun μπορούμε να ανακτήσουμε τα δεδομένα αυτά ελέγχοντας το τέλος της γραμμής εντολών. Θέλουμε να γίνει αυτό πριν το rump kernel ξεκινήσει να αρχικοποιεί τις συσκευές του ώστε να γνωρίζουμε τότε ποιες συσκευές virtio να αρχικοποιηθούν. Για αυτό το λόγο κάλούμε την rump_parse_mmio_device_configuration στην αρχή της συνάρτησης rumprun_boot() του αρχείου lib/librumprun_base/rumprun.c, όπου είναι διαθέσιμη η γραμμή εντολών ως παράμετρος. Η rump_parse_mmio_device_configuration έχει το ρόλο να ανακτήσει τα δεδομένα και ύστερα να τα αφαιρέσει από τη γραμμή εντολών ώστε να μην παρεμποδίσουν την υπόλοιπη διαδικασία. Οι πληροφορίες σώζονται σε έναν πίνακα που μπορεί ύστερα να προσπελαστεί για να γίνει γνωστό ποιες συσκευές virtio είναι διαθέσιμες.

3.8.3 Δημιουργία driver

Στην κλασική υλοποίηση του virtio μέσω PCI, φροντίζει ο driver αυτού ώστε να αρχικοποιήσει τον virtio driver, και στη συνέχεια αυτός φροντίζει να εντοπίσει και να αρχικοποιήσει τις υπάρχουσες virtio συσκευές. Στην υλοποίηση του virtio μέσω MMIO για την αρχιτεκτονική evbarm αναλαμβάνει τον ενδιάμεσο ρόλο αυτό να αρχικοποιήσει τις συσκευές virtio το fdt ή το acpi. Για να επιτύχουμε το ίδιο στην περίπτωσή μας, πρέπει να κατασκευάσουμε έναν αντίστοιχο ενδιάμεσο driver ο οποίος να εντοπίζει τις virtio συσκευές μέσω της boot γραμμής εντολών και ύστερα να τις συνδέει στην ιεραρχία συσκευών. Ονομάσαμε αυτό τον πειραματικό driver mmiocmdl.

3.8.3.1 Θεωρία

Το Rumprun κληρονομεί από το NetBSD μία πληθώρα driver, αλλά μαζί με αυτούς υιοθετεί και το configuration που απαιτούν. Κατά το χτίσιμο του unikernel πρέπει να γίνει γνωστό ποιοι driver να συμπεριληφθούν στο εκτελέσιμο, ενώ κατά την εκτέλεση πρέπει να αναγνωριστούν οι παρούσες συσκευές και να αρχικοποιηθούν οι αντίστοιχοι driver. Για το πρώτο, το Rumprun παρέχει το εργαλείο rumprun-bake με το οποίο μπορούν να επιλεχθούν οι ομάδες driver που επιθυμείς στην τελική εικόνα [14]. Ωστόσο για να είναι διαθέσιμος ένας driver ως επιλογή ύστερα από το κτίσιμο του Rumprun καθώς και για τη διαχείριση των συσκευών κατά την εκτέλεση, χρησιμοποιείται το σύστημα του NetBSD. Εφόσον θέλουμε να τροποποιήσουμε το Rumprun και να προσθέσουμε τους δικούς μας driver είναι θεμιτό να εξηγήσουμε πώς λειτουργεί αυτό το σύστημα.

Στο NetBSD χρησιμοποιείται ένα αρχείο που ονομάζεται kernel configuration file (αρχείο διαμόρφωσης πυρήνα) για να περιγράψει, μεταξύ άλλων, τις συσκευές που θα συμπεριληφθούν στον πυρήνα κατά τη μεταγλώττιση αυτού [40, 41]. Στα rump kernel αυτό αντικαθίσταται από μια σειρά αρχείων με κατάληξη .ioconf το καθένα εκ των οποίων περιγράφει μία συσκευή. Και στις δύο περιπτώσεις, στην περιγραφή αυτή όλες οι συσκευές με μοναδική εξαίρεση τον ριζικό κόμβο έχουν έναν γονέα, σχηματίζοντας έτσι έναν κατευθυνόμενο γράφο [41]. Στην περιγραφή αυτή συσχετίζουμε τη συσκευή και με συγκεκριμένα αρχεία κώδικα ώστε αυτά να συμπεριληφθούν για μεταγλώττιση. Στην περίπτωση του Rumprun αυτό έχει ως αποτέλεσμα

μια στατική βιβλιοθήκη που μπορεί να προστεθεί στο εκτελέσιμο με τη χρήση του rumprun-bake.

Κατά τη διάρκεια της εκκίνησης, το NetBSD (και άρα το Rumprun), ακολουθεί μια διαδικασία που ονομάζεται autoconfiguration (αυτοδιαμόρφωση) κατά την οποία ταιριάζει όλες τις παρούσες συσκευές στους αντίστοιχους driver και φροντίζει για την κατάλληλη αρχικοποίησή τους [42]. Κατά τη διάρκεια αυτής της διαδικασίας γίνεται μία κατά βάθος διάβαση (depth-first traversal) του κατευθυνόμενου γράφου και αυτός μετατρέπεται σε δέντρο [41].

Για να γίνει δυνατή αυτή η διαδικασία, ένας driver πρέπει να περιέχει μία διεπαφή μέσω της cfattach δομής. Αξιοσημείωτες από αυτή τη διεπαφή είναι οι συναρτήσεις match και attach. Η συνάρτηση match καλείται κατά το autoconfiguration για να προσδιοριστεί αν μία δοθείσα συσκευή ταιριάζει στον driver. Για θετική απάντηση επιστρέφει έναν μη μηδενικό ακέραιο. Η συνάρτηση attach καλείται αν η match έδωσε θετική απάντηση και έχει ρόλο να 'συνδέσει' τον driver στο γονέα του (το bus στο οποίο εντοπίστηκε η συσκευή), δηλαδή να εκτελέσει όποια αναγκαία αρχικοποίηση ώστε να λειτουργήσει βάσει του συγκεκριμένου γονέα [43].

3.8.3.2 Υλοποίηση

Ο mmiocmdl εισήχθη στο rump kernel ως μία συσκευή που συνδέεται στη ρίζα (root) του δέντρου συσκευών. Συμπεριλαμβάνοντας το αρχείο `sys/rump/lib/libmmiocmdl/mmiocmdl_cfdata.c` στον πηγαίο κώδικα φροντίζουμε να γίνει η σύνδεση της συσκευής mmiocmdl στη ρίζα. Στην attach συνάρτηση που καλείται τότε φροντίζουμε να διαβάσουμε τον πίνακα των συσκευών MMIO και να καλέσουμε τη συναρτήση virtio_mmio_common_attach() που εκτελεί την αρχικοποίηση του MMIO πρωτοκόλλου και προετοιμάζει βάσει αυτής τις δομές δεδομένων ώστε να ακολουθήσει η κλήση της attach συνάρτησης της συσκευής που βρέθηκε.

```
~/code/diplo/firecracker $ ./firecracker --api-sock /tmp/firecracker.socket --config-file config5.json --level Info  --log-path
g
rump kernel bare metal bootstrap

x86 initclocks(): Using PV clock for timekeeping
console=ttyS0 reboot=k panic=1 virtio mmio.device=4K@0xd0000000:5
Copyright (c) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005,
    2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016
    The NetBSD Foundation, Inc.  All rights reserved.
Copyright (c) 1982, 1986, 1989, 1991, 1993
    The Regents of the University of California.  All rights reserved.

NetBSD 7.99.34 (RUMP-ROAST)
total memory = 253 MB
timecounter: Timecounters tick every 10.000 msec
timecounter: Timecounter "clockinterrupt" frequency 100 Hz quality 0
cpu0 at thinair0: rump virtual cpu
root file system type: rumpfs
kern.module.path=/stand/amd64/7.99.34/modules
mainbus0 (root)
mainbus0: WARNING: power management not supported
pci0 at mainbus0 bus 0
pci0: i/o space, memory space enabled, rd/line, rd/mult, wr/inv ok
mmiocmdl0: network device (rev. 0x02)
mmiocmdl0: interrupting on irq 5
vioif0 at mmiocmdl0: Ethernet address aa:fc:00:00:00:01
vioif0: Features: 0x100000020<MAC>
mmiocmdl0: allocated 12288 byte for virtqueue 0 for rx, size 256
mmiocmdl0: allocated 12288 byte for virtqueue 1 for tx, size 256
vioif0: WARNING: power management not supported
timecounter: Timecounter "bmktc" frequency 1000000000 Hz quality 100
mounted tmpfs on /tmp
rumprun: could not find start of json.  no config?

=== calling "rumprun" main() ===
```

Σχήμα 3.3: Παράδειγμα Virtio Over MMIO στο Rumprun-Firecracker

Αυτό μας επέτρεψε να διεξαχθεί η απαραίτητη επικοινωνία μεταξύ guest driver και host συ-

σκευής, με αποτέλεσμα την αρχικοποίηση της συσκευής, συμπεριλαμβανομένης της κατανομής virtqueues. Ένα παράδειγμα των αποτελεσμάτων φαίνεται στην εικόνα 3.3.

Παρακάτω φαίνεται αρχείο διαμόρφωσης του firecracker που προσθέτει συσκευές δίσκου και δικτύου.

Κώδικας 3.5: Παράδειγμα διαμόρφωσης του firecracker μ έ

```
{
  "boot-source": {
    "kernel_image_path": "/file/path/to/netOrBlock.bin",
    "boot_args": "console=ttyS0 reboot=k panic=1"
  },
  "drives": [
       {
              "drive_id": "rootfs",
              "path_on_host": "/file/path/to/hello-rootfs.ext4",
              "is_root_device": true,
              "is_read_only": false
        }
  ],
  "machine-config": {
    "vcpu_count": 1,
    "mem_size_mib": 512,
    "ht_enabled": false
  },
  "network-interfaces": [
     {
       "iface_id": "eth0",
       "guest_mac": "AA:FC:00:00:00:01",
       "host_dev_name": "tap0"
     }
   ]
}
```

Για να μπορέσει να λειτουργήσει το virtio, βεβαίως χρειάζονται και οι ειδοποιήσεις από τη συσκευή προς τον driver, οι οποίες υλοποιούνται με διακοπές. Πρέπει, λοιπόν, ο driver να φροντίσει να εγκαταστήσει χειριστές διακοπών ώστε όταν προκύπτει μία σχετική διακοπή, να ειδοποιείται η αντίστοιχη συσκευή virtio.

Οι διακοπές διακρίνονται μεταξύ τους από τον αριθμό IRQ, ο οποίος σε φυσικά συστήματα προκύπτει από τις εισόδους ολοκληρωμένων κυκλωμάτων που λαμβάνουν σήμα αίτησης διακοπής. Το Firecracker θεωρεί ότι έχει τους αριθμούς 5 έως 24 διαθέσιμους για το virtio και τους διανείμει στις συσκευές που παρέχει με τη σειρά (η πρώτη συσκευή χρησιμοποιεί τη διακοπή με αριθμό 5, η δεύτερη χρησιμοποιεί τη διακοπή 6, και ούτω καθεξής).

Στο Rumprun, όμως, δεν είναι όλες οι διακοπές διαθέσιμες για να εγκαταστήσουμε δι-

κούς μας χειριστές. Η δυνατότητα αυτή προστίθεται για κάθε διακοπή ξεχωριστά στα αρχεία `platform/hw/arch/amd64/intr.S` και `platform/hw/arch/x86/cpu_subr.c`). Για αρχή δοκιμάσαμε να ενεργοποιήσουμε τις διακοπές 5 και 6, επιτρέποντας έτσι δύο συσκευές virtio ταυτόχρονα.

Το Rumprun προσφέρει ήδη τη συνάρτηση rumpcomp_pci_irq_establish για να επιτρέπει στο PCI να εγκαθιστά χειριστές διακοπών. Εμείς χρησιμοποιήσαμε μια δικιά μας παραλλαγή αυτής που επιτρέπει να ορίσουμε συγκεκριμένα σε ποιον αριθμό διακοπής αντιστοιχεί ο χειριστής. Αυτή καλείται όταν ο virtio driver προσπαθεί να αρχικοποιήσει τις διακοπές ώστε να θέσει τη συνάρτηση virtio_mmio_intr ως το χειριστή της διακοπής.

Εν τέλει λόγω των περιορισμών χρόνου στα πλαίσια της διπλωματικής εργασιάς, δεν μπορέσαμε να ελέγξουμε τα αποτελέσματα του έργου αυτού πλήρως, σε πρακτικό επίπεδο. Δεν είχαμε τη δυνατότητα να επιβεβαιώσουμε αν λειτουργούν σωστά οι διακοπές. Ταυτόχρονα, μερικές πρώτες προσπάθειες να τρέξουμε εφαρμογές που θα χρησιμοποιούσαν τις virtio συσκευές δεν ήταν επιτυχείς, δείχνοντας ότι το Rumprun σε Firecracker δεν είναι ακόμη έτοιμο να τρέξει εφαρμογές που απαιτούν Virtio. Ωστόσο, μπορέσαμε να προχωρήσουμε αρκετά προς αυτή τη δυνατότητα.

Κεφάλαιο 4

Αξιολόγηση

Για να μπορέσουμε να αξιολογήσουμε το port που παράξαμε, πραγματοποιήθηκε μια σειρά μετρήσεων ώστε να ελέγξουμε την επίδοση του Rumprun σε συνδυασμό με το Firecracker. Ως μέτρο σύγκρισης, κάναμε αντίστοιχες μετρήσεις στους συνδυασμούς unikernel-VMM: Rumprun-QEMU/KVM, Rumprun-Solo5, OSv-QEMU/KVM, OSv-Firecracker. Δοκιμάζουμε επίσης το Rumprun-Firecracker χρησιμοποιώντας τον Jailer του Firecracker.

4.1 Παράμετροι αξιολόγησης

Οι παράμετροι που χρησιμοποιήσαμε ήταν ο χρόνος εκκίνησης και το αποτύπωμα μνήμης. Η θεωρία προβλέπει ότι η εκτέλεση του Rumprun ως microVM θα επιτρέψει τη γρηγορότερη εκκίνησή του καθώς και τον περιορισμό χρησιμοποίησης μνήμης. Κληθήκαμε λοιπόν να ελέγξουμε την πραγματικότητα αυτής της υπόθεσης.

4.2 Σύστημα αξιολόγησης

Για τον χρόνο εκκίνησης, μετρήθηκε ο χρόνος πριν την εκτέλεση του Rumprun και ο χρόνος αμέσως αφού εισήλθε στο main πρόγραμμα, σε ακρίβεια μικροδευτερόλεπτου. Η διαφορά των δύο αυτών τιμών μας δίνει τον χρόνο εκκίνησης. Στην περίπτωση του OSv, χρησιμοποιήθηκε η τιμή που υπολογίζει το ίδιο το OSv. Λάβαμε 20 μετρήσεις για κάθε ζεύγος, από τις οποίες υπολογίσαμε το ελάχιστο, το μέγιστο και τη μέση τιμή.

Κατά τον πειραματισμό παρατηρήσαμε ότι μετά την πρώτη εκτέλεση ενός guest με το Firecracker (και το Solo5), επακόλουθες εκτελέσεις είχαν σημαντικά μειωμένο χρόνο εκκίνησης. Υποθέσαμε ότι η συμπεριφορά αυτή προκύπτει διότι η μνήμη και ο δίσκος χρησιμοποιούν κρυφές μνήμες (cache), οι οποίες επιταχύνουν τη φόρτωση δεδομένων από τον δίσκο στη μνήμη. Για να επιβεβαιώσουμε αυτή την υπόθεση, αλλά και για επιπλέον ανάλυση λάβαμε μετρήσεις για το Firecracker με τρεις τρόπους: αδειάζουμε τις cache πριν την πρώτη εκτέλεση (προσομοιώνουμε το σενάριο της αρχικής εκκίνησης του unikernel χωρίς προηγούμενη προετοιμασία), αδειάζουμε τις cache πριν κάθε εκτέλεση (ελέγχουμε τη συμπεριφορά ανεξάρτητα από τις επιταχύνσεις των cache), δεν αδειάζουμε τις cache καθόλου (προσομοίωση της περίπτωσης ενός

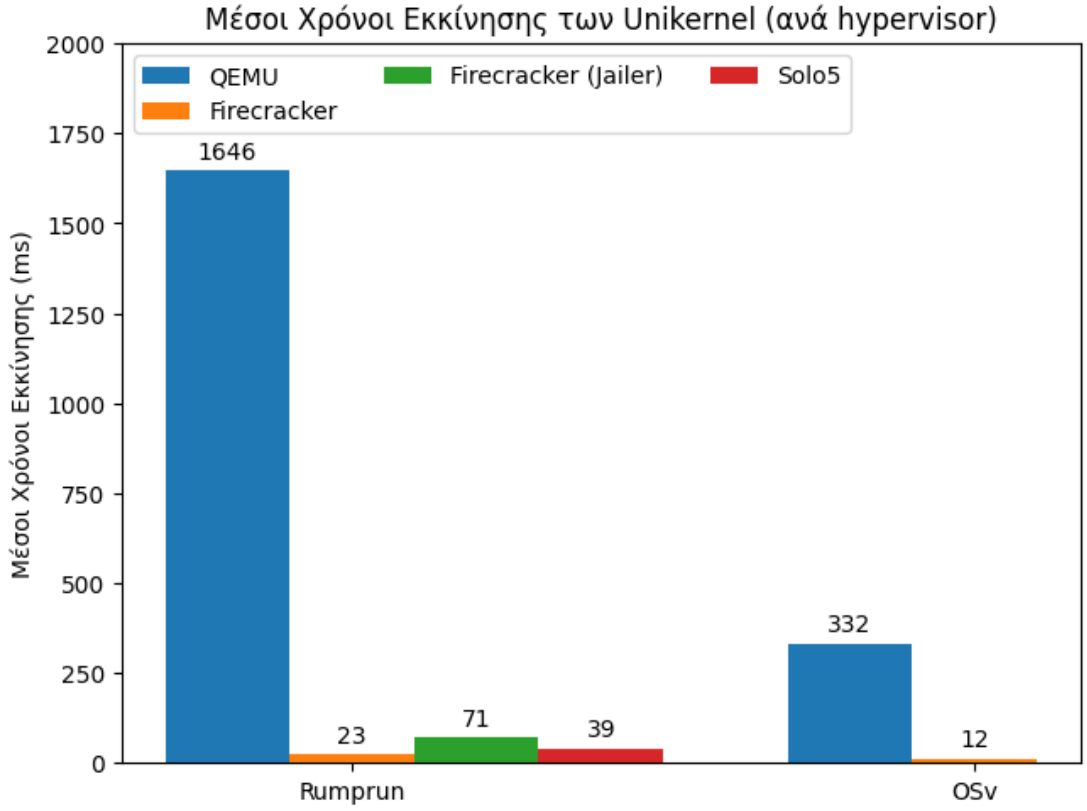
unikernel το οποίο εκτελείται επανειλημμένα και έχει ήδη επιταχυνθεί και θεωρείται το τυπικό επιθυμητό σενάριο).

Για την αξιοποίηση μνήμης, χρησιμοποιήσαμε εργαλεία που παρέχει το Λειτουργικό Σύστημα, όπως το htop και το System Monitor (Linux Mint). Με αυτά μετρήσαμε τη μνήμη που χρησιμοποιούν οι σχετικές διαδικασίες του κάθε hypervisor για μια εφαρμογή που εκτελεί μόνο έναν άδειο ατέρμονο βρόχο. Συγκεκριμένα, μετρήσαμε τις τιμές Resident Memory (φυσική μνήμη σε χρήση από τη διεργασία) και Shared Memory (κοινόχρηστη μνήμη) και σημειώσαμε τη διαφορά της δεύτερης από την πρώτη (RES - SHR). Για το QEMU, προσθέσαμε τη μνήμη της διαδικασίας του ίδιου του QEMU και της διαδικασίας που δημιουργεί για τον guest. Για το Firecracker, υπάρχει μόνο μία διαδικασία. Οι τιμές της μνήμης δεν έχουν μεγάλη διακύμανση, οπότε ασχοληθήκαμε κυρίως με τις μέσες τιμές. Όταν χρησιμοποιούμε τον Jailer, προσθέτουμε τη μνήμη που χρησιμοποιεί και αυτός.

4.3 Αποτελέσματα

4.3.1 Χρόνος εκκίνησης

Από τις 20 μετρήσεις χρόνου που συλλέξαμε για κάθε ζεύγος, εξάγαμε τον ελάχιστο, μέγιστο και μέσο χρόνο εκκίνησης. Οι μέσοι χρόνοι, ομαδοποιημένοι κατά unikernel φαίνονται γραφικά παρακάτω:

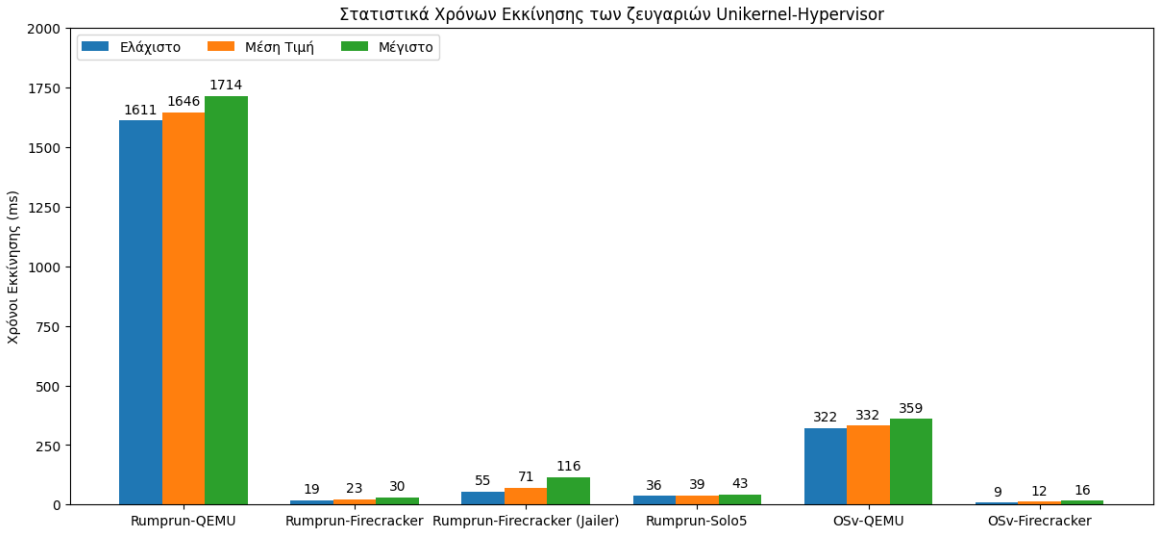


Σχήμα 4.1: Μέσοι Χρόνοι Εκκίνησης

Πιο αναλυτικά παρουσιάζονται οι τιμές αυτές στο σχήμα 4.2.

Μπορούμε να συμπεράνουμε, λοιπόν, ότι όπως υποθέσαμε στην αρχή της διπλωματικής, η χρήση του Firecracker δίνει σημαντική βελτίωση στον χρόνο εκκίνησης (δύο τάξεις μεγέθους μικρότερη), και μάλιστα ακόμη περισσότερο από ό,τι στο OSv. Το Firecracker αποτελεί την πιο γρήγορη επιλογή μεταξύ των τριών VMM που δοκιμάσαμε για το Rumprun, αν και ο συνδυασμός OSv-Firecracker παραμένει γρηγορότερος.

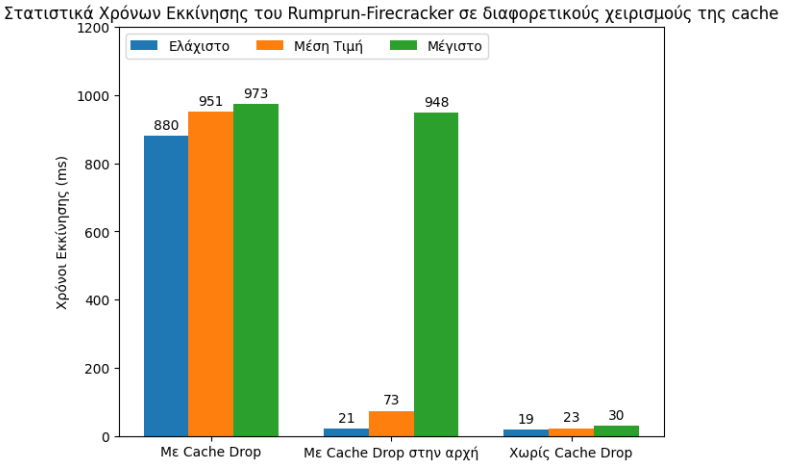
Παρατηρούμε ότι η χρήση του Jailer αυξάνει τον χρόνο εκκίνησης, όχι όμως υπερβολικά και παραμένει στην ίδια τάξη μεγέθους. Προσφέρεται έτσι ένα αντάλλαγμα μειωμένης ταχύτητας για αυξημένη ασφάλεια.



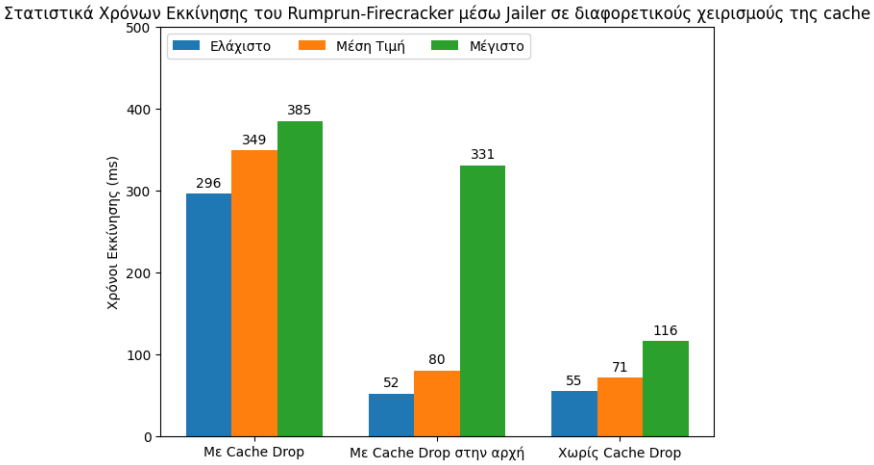
Σχήμα 4.2: Χρόνοι Εκκίνησης για ζεύγη Unikernel - Hypervisor

Σημειώνουμε, επίσης, ότι όταν μετρήθηκε ο χρόνος εκκίνησης για το Rumprun-Firecracker που έκανε χρήση των ημιτελών driver του Virtio over MMIO, ήταν πάλι πιο υψηλός, με μέσο όρο 41 ms. Επιβαρύνεται λίγο, δηλαδή από το Virtio over MMIO (παραμένει σημαντικά βελτιωμένος σε σχέση με το QEMU), αλλά στην υλοποίηση αυτή φορτώνεται ακόμη το PCI. Μία πρακτική εκδοχή του Rumprun που κάνει χρήση του MMIO λογικό είναι να αφαιρέσει και το PCI, οπότε να εξισορροπήσει πιθανώς αυτή την επιβάρυνση. Ενδιαφέρον είναι επίσης, ότι ο μέσος χρόνος εκκίνησης ήταν ο ίδιος είτε προσθέταμε μόνο συσκευή δικτύου είτε προσθέταμε και συσκευή δικτύου και δίσκου. Αυτό υπονοεί ότι η καθυστέρηση δεν αυξάνεται με τον αριθμό των virtio συσκευών που πρέπει να συνδεθούν.

Τα αποτελέσματα των μετρήσεων με διαφορετικές διαχειρίσεις των cache παρατίθονται στα σχήματα 4.3, 4.4 και 4.5.

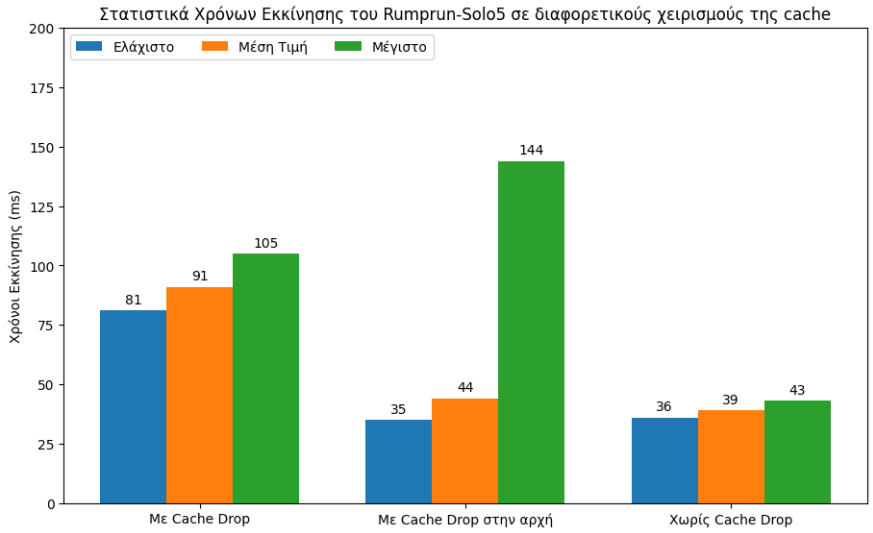


Σχήμα 4.3: Χρόνοι Εκκίνησης για Firecracker σε διαφορετικούς χειρισμούς των cache



Σχήμα 4.4: Χρόνοι Εκκίνησης για Firecracker (Jailer) σε διαφορετικούς χειρισμούς των cache

Παρατηρούμε ό,τι αν αφαιρέσουμε τις επιταχύνσεις των cache πλήρως, τα αποτελέσματα διατηρούν μια διακύμανση, η οποία μπορεί να αποδοθεί στην στοχαστικότητα της χρονοδρομολόγησης διαδικασιών σε έναν υπολογιστή, αλλά δεν υπάρχει πτωτική τάση. Αντιθέτως, όταν αδειάζουμε τις cache μόνο στην αρχή, παρατηρούμε υψηλό αρχικό χρόνο εκκίνησης, ο οποίος όμως γρήγορα πέφτει ώστε ο μέσος όρος παραμένει αρκετά χαμηλός. Όταν δεν αδειάζουμε τις cache καθόλου, οι χρόνοι εκκίνησης είναι εξαρχής μικροί. Από αυτά τα αποτελέσματα μπορούμε να συμπεράνουμε ότι όντως οι ασύμφωνες μετρήσεις που παρατηρήσαμε αρχικώς οφείλονται στη συμπεριφορά των cache και η παραπάνω ανάλυσή μας είναι χρήσιμη αν υποθέσουμε συνθήκες χρήσης κρυφών μνημών, το οποίο είναι αποδεκτό λόγω της διαδεδομένης χρήσης τους στα σύγχρονα υπολογιστικά συστήματα. Επιπλεόν, ακόμη και στην περίπτωση 'ψυχρής εκκίνησης', η αρχική καθυστέρηση της πρώτης εκτέλεσης δεν επηρεάζει

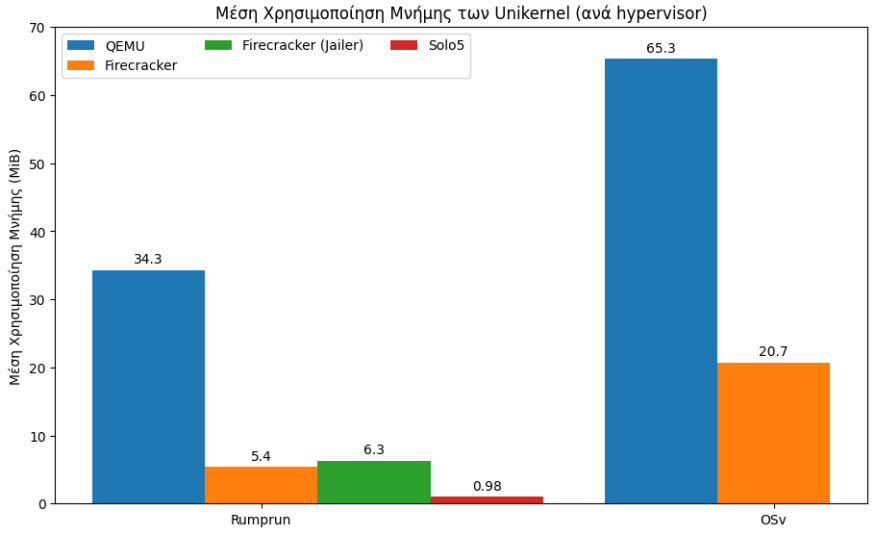


Σχήμα 4.5: Χρόνοι Εκκίνησης για Solo5 σε διαφορετικούς χειρισμούς των cache

το μέσο χρόνο υπερβολικά, αρκεί να υπάρχουν αρκετές επανεκτελέσεις, κάτι που προβλέπεται στις βασικές περιπτώσεις χρήσης των microVM.

4.3.2 Αξιοποίηση Μνήμης

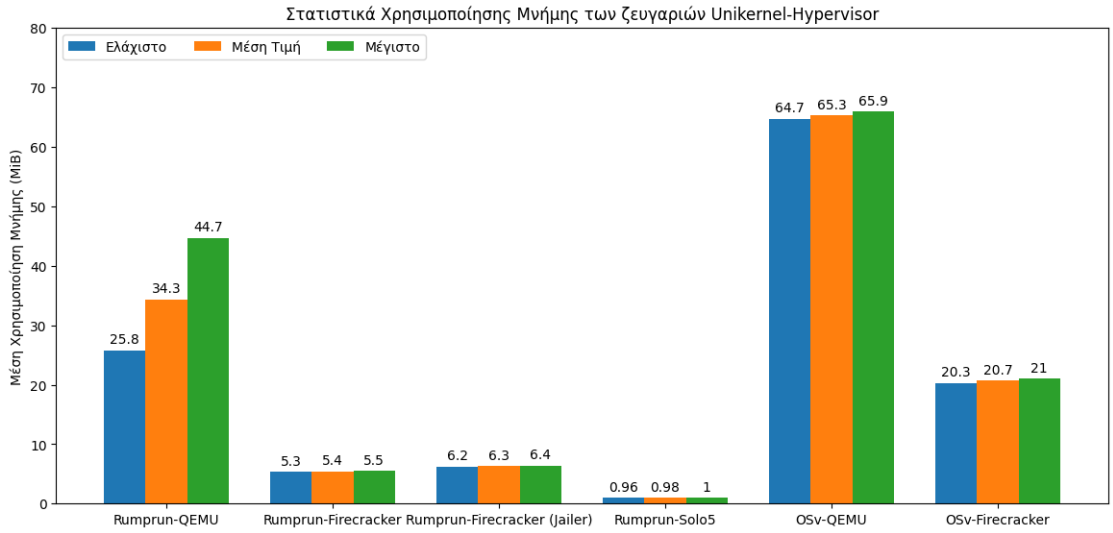
Συγκρίνουμε τη χρησιμοποίηση μνήμης (μέσος όρος) σε Rumprun και OSv για διαφορετικούς hypervisor στο γράφημα 4.6.



Σχήμα 4.6: Μέση χρησιμοποίηση μνήμης

Πιο αναλυτικά παρουσιάζουμε τα αποτελέσματά μας στη γραφική παράσταση 4.7.

Τα αποτελέσματα αυτά μας δείχνουν ότι υπάρχει όντως βελτίωση στο μέγεθος της μνήμης που χρησιμοποιείται, σε σύγκριση με το QEMU, κατά μία τάξη μεγέθους. Παρατηρούμε επίσης ότι το Rumprun έχει καλύτερες επιδόσεις και μεγαλύτερη βελτίωση ως προς τη μνήμη



Σχήμα 4.7: Χρησιμοποίηση μνήμης για ζεύγη Unikernel - Hypervisor

από ό,τι το OSv, ενώ το Rumprun-Solo5 κάνει τη μικρότερη χρήση μνήμης από όλους τους συνδυασμούς.

Παρατηρούμε εδώ ότι η χρήση μνήμης του Firecracker με και χωρίς τον Jailer ήταν η ίδια (5.4 MiB), με μόνη διαφορά την ύπαρξη της διαδικασίας του Jailer η οποία είχε κατά μέσο όρο μέγεθος 892 KiB, που είναι σχετικά μικρή αύξηση.

Επίσης πρέπει να σχολιαστεί ότι οι μετρήσεις για το Rumprun-Firecracker ήταν πιο ψηλές (6.5 MiB) όταν προσθέσαμε συσκευή δικτύου, με αποτέλεσμα να φορτωθούν οι (ημιτελείς) driver MMIO και οι driver δικτύου. Περαιτέρω, όταν προσθέσαμε συσκευές δικτύου και δίσκου μαζί, η χρήση μνήμης ανέβηκε στα 7,1 MiB. Αυτό ενδεικνύει ότι μια πιο πρακτική εκδοχή του Rumprun που μπορεί να τρέξει εφαρμογές virtio θα χρειαστεί παραπάνω μνήμη από ό,τι δείχνουν τα παραπάνω σχήματα, άλλα όχι σημαντικά περισσότερη ώστε να αλλάξει τα συμπεράσματά μας. Υπονοεί, βεβαίως, επίσης ότι με μια βελτιστοποίηση στην επιλογή των driver (αφαίρεση όλων των αχρείαστων που ακόμη φορτώνονται) θα μπορούσαμε να επιτύχουμε ακόμη μικρότερη αξιοποίηση μνήμης.

4.4 Σύνοψη συμπερασμάτων αξιολόγησης

Ανακεφαλαιώνοντας, η μελέτη μας υπέδειξε τα εξής συμπεράσματα:

1. Ο συνδυασμός Rumprun-Firecracker βελτιώνει τον χρόνο εκκίνησης και την χρησιμοποίηση μνήμης σε σχέση με τον συνδυασμό Rumprun-QEMU.

2. Το OSv-Firecracker έχει τον μικρότερο χρόνο εκκίνησης, ενώ το Rumprun-Solo5 έχει τη μικρότερη χρησιμοποίηση μνήμης, αλλά το Rumprun-Firecracker έχει συγκρίσιμες ενδιάμεσες τιμές και στα δύο.

3. Η χρήση του Jailer του Firecracker κάνει τα microVM πιο ακριβά, ιδίως ως προς τον

χρόνο εκκίνησης, αλλά όχι σε τέτοιο βαθμό ώστε να μην είναι βιώσιμα.

4. Η φόρτωση του mmiocmdl driver έχει επιβάρυνση στις μετρικές μας, άλλα όχι πολύ μεγάλη.

5. Η χρήση κρυφών μνημών έχει αισθητή διαφορά στον χρόνο εκκίνησης του Rumprun-Firecracker, το οποίο όμως δεν επηρεάζει τη συνολική του επίδοση υπό φυσιολογικές συνθήκες.

Κεφάλαιο 5

Επίλογος

Στο κεφάλαιο αυτό συνοψίζουμε το έργο που εκπονήθηκε στα πλαίσια της διπλωματικής αυτής και τα συμπεράσματα που προέκυψαν από αυτό. Τέλος, παραθέτουμε πιθανά σημεία ανάπτυξης της εργασίας, τα οποία θα μπορούσαν να αποτελέσουν θέμα μελλοντικής μελέτης.

5.1 Σύνοψη και συμπεράσματα

Ζούμε μια εποχή άνθησης και ευρείας διάδοσης του Cloud Computing, όπου αυξανόμενος αριθμός εφαρμογών που ανήκουν σε αυξανόμενο αριθμό χρηστών συμβιώνουν στα ίδια φυσικά συστήματα. Είναι επακόλουθο, λοιπόν, να αναζητούνται τρόποι ώστε να ελαττωθεί το αποτύπωμα αυτών των εφαρμογών, καθώς και να αυξηθεί η ασφάλεια της μίας από την άλλη και του συνολικού συστήματος από αυτές. Δημιουργείται η απαίτηση να μπορούν να απομονωθούν οι εφαρμογές των διαφορετικών χρηστών, διατηρώντας όμως τους πόρους που αποδίδονται στην καθεμία στους απολύτως απαραίτητους.

Στην προσπάθεια να απομακρυνθούμε από τα δαπανηρά πλήρη εικονικοποιημένα λειτουργικά συστήματα, τα microVM και τα unikernel είναι δύο τεχνολογίες που έχουν αναπτυχθεί με σκοπό να διατηρήσουν — ή και να αυξήσουν — την ασφάλεια των εικονικών μηχανών, ταυτοχρόνως ελαχιστοποιώντας τους χρησιμοποιούμενους πόρους. Τα microVM ασχολούνται με τη βελτιστοποίηση του κομματιού της εικονικοποίησης, ενώ τα unikernel στοχεύουν στην ελάττωση της εφαρμογής και του Λειτουργικού Συστήματος σε ένα ενιαίο εκτελέσιμο. Οι δύο τεχνολογίες αναπτύχθηκαν ξεχωριστά, αλλά μας δίνουν την ευκαιρία να συνεργαστούν για ακόμη καλύτερα αποτελέσματα.

Για αυτό το λόγο, στην διπλωματική εργασία αυτή κληθήκαμε να δοκιμάσουμε τη συνεργασία συγκεκριμένων παραδειγμάτων αυτών των τεχνολογιών, τον Firecracker VMM και το Rumprun unikernel. Το Rumprun δεν ήταν σχεδιασμένο να μπορεί να εκτελεστεί μέσω του Firecracker, οπότε το κύριο αντικείμενο της εργασίας ήταν η τροποποίησή του ώστε να μπορέσει να λειτουργήσει ως microVM δημιουργημένο από το Firecracker.

Σε δεύτερη φάση, αφού επιτεύχθηκε ο πρώτος στόχος, εξετάσαμε τα αποτελέσματα του συνδυασμού αυτού και τα συγκρίναμε με προηγούμενα παραδείγματα σύμπραξης microVM και unikernel. Τα συμπεράσματα που προέκυψαν ήταν ότι η χρήση του Firecracker ως hypervisor

για το Rumprun βελτιώνει σημαντικά μετρικές όπως ο χρόνος εκκίνησης και το αποτύπωμα μνήμης, ενώ αποτελεί ανταγωνιστικό σύστημα σε σχέση με άλλους δημοφιλείς συνδυασμούς όπως το OSv πάνω σε Firecracker.

Αν και δεν είναι ο βέλτιστος συνδυασμός ως προς τις μετρικές που αξιολογήθηκαν, είναι αρκετά κοντά ώστε η συμβατότητα με το POSIX να τον κάνει μια ελκυστική επιλογή. Ωστόσο, η ισχύουσα έλλειψη συμβατότητας με εφαρμογές virtio μειώνει αρκετά τη χρησιμότητα που μπορεί να έχει το Rumprun-Firecracker σε πρακτικές εφαρμογές.

5.2 Μελλοντικές επεκτάσεις

Τα αποτελέσματα της μελέτης δείχνουν πως το Rumprun για Firecracker δεν είναι ακόμη έτοιμο για χρήση από ευρύ κοινό. Ταυτοχρόνως, όμως, δείχνουν ότι είναι πολλά υποσχόμενο, εάν συγκεκριμένα προβλήματα ξεπεραστούν. Για αυτό το λόγο, είναι ένα αντικείμενο που αξίζει να αναπτυχθεί παραπάνω σε μελλοντικές μελέτες ώστε η ευρής χρήση του να γίνει βιώσιμη.

Η πιο σημαντική πιθανή επέκταση είναι η προσθήκη της υποστήριξης του Virtio over MMIO. Όπως διαπιστώθηκε κατά την έρευνά μας, η έλλειψη MMIO υποστήριξης στο Rumprun εμποδίζει την virtio επικοινωνία με το Firecracker και άρα τη χρήση εφαρμογών που αξιοποιούν το virtio. Αυτές συμπεριλαμβάνουν το σημαντικό σύνολο εφαρμογών που κάνουν χρήση του δίσκου ή του δικτύου. Προσπάθεια να διορθωθεί αυτή η έλλειψη έγινε και ως μέρος αυτής της εργασίας και προχώρησε το έργο, αλλά δεν μπόρεσε να το ολοκληρώσει. Έτσι, μία πιθανή μελέτη η οποία θα προσπαθούσε να συμπληρώσει το έργο αυτό θα ήταν πολύ χρήσιμη και το αναμενόμενο επόμενο βήμα ανάπτυξης του Rumprun για Firecracker.

Μια άλλη κατεύθυνση μελλοντικής επέκτασης θα ήταν να αφαιρεθούν πρωτόκολλα όπως το PCI τα οποία φορτώνονται αυτόματα στο Rumprun, αλλά δεν χρησιμοποιούνται από το Firecracker ώστε να βελτιστοποιηθεί περαιτέρω το Rumprun όταν στοχεύεται συγκεκριμένα στο Firecracker. Επιπλέον μπορεί να μελετηθεί πόσο επηρεάζονται ο χρόνος εκκίνησης και η χρησιμοποίηση μνήμης όταν αλλάζουν οι driver που επιλέγεται να συμπεριληφθούν στο εκτελέσιμο, καθώς και όταν δεν χρησιμοποιείται το στρώμα υποστήριξης POSIX.

Θα μπορούσε επίσης να διερευνηθεί παραπάνω η διαχείριση της σελιδοποίησης και γενικώς του προνομιακού κώδικα από το Rumprun και το Firecracker. Εάν περιορίσουμε τις προνομιακές εντολές στο Firecracker, στη συνέχεια μπορούμε να ερευνήσουμε πώς μπορεί το Firecracker να εκτελέσει το Rumprun στο Ring 3.

Βιβλιογραφία

[1] Mell, Peter, and Tim Grance, The NIST definition of cloud computing, *Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg,* 2011.

[2] COMPUTING, MOBILE. Mobile Computing. *Context Awareness, and Energy Awareness in Soft*, 2012.

[3] Wikipedia - Mobile backend as a service, `https://en.wikipedia.org/wiki/Mobile_backend_as_a_service`, Last accessed on 27/07/2022.

[4] JONAS, Eric, et al. Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*, 2019.

[5] SHAHRAD, Mohammad; BALKIND, Jonathan; WENTZLAFF, David. Architectural implications of function-as-a-service computing. In: *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture.* 2019. p. 1063-1075.

[6] Unikernels - Rethinking Cloud Infrastructure, `www.unikernel.org`, Last accessed on 29/07/2022.

[7] MADHAVAPEDDY, Anil; SCOTT, David J. Unikernels: Rise of the Virtual Library Operating System: What if all the software layers in a virtual appliance were compiled within the same safe, high-level language framework?. *Queue*, 2013, 11.11: 30-44.

[8] MADHAVAPEDDY, Anil, et al. Unikernels: Library operating systems for the cloud. *ACM SIGARCH Computer Architecture News*, 2013, 41.1: 461-472.

[9] MADHAVAPEDDY, Anil, et al. Jitsu:Just-In-Time Summoning of Unikernels. In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 2015. p. 559-573.

[10] BUER, Per. Unikernels are secure. Here is why., `http://unikernel.org/blog/2017/unikernels-are-secure`, Last accessed on 17/08/2022.

[11] PAVLICEK, Russell. Unikernels: Beyond Containers to the Next Generation of Cloud. *O'Reilly Media*, 2017.

[12] KANTEE, Antti. The Design and Implementation of the Anykernel and Rump Kernels, 2nd edition. 2016.

[13] Rumprun repository, `https://github.com/rumpkernel/rumprun`, Last accessed on 1/09/2022.

[14] Rump Kernel Wiki - Repo: rumprun, `https://github.com/rumpkernel/wiki/wiki/Repo:-rumprun`, Last accessed on 4/03/2023.

[15] Nabla containers: a new approach to container isolation, `https://nabla-containers.github.io/`, Last accessed on 1/09/2022.

[16] ΛΑΓΚΑΣ ΝΙΚΟΛΟΣ, Ορέστης. Μελέτη και αποτίμηση μεθόδων εκτέλεσης εφαρμογών ως Unikernels σε αρχιτεκτονικές ARM. 2018.

[17] CHIUEH, Susanta Nanda Tzi-cker; BROOK, Stony. A survey on virtualization technologies. *Rpe Report*, 2005, 142.

[18] KOLYSHKIN, Kirill. Virtualization in linux. *White paper, OpenVZ*, 2006, 3.39: 8.

[19] GRAZIANO, Charles David. A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project. 2011.

[20] Wikipedia - Virtualization, `https://en.wikipedia.org/wiki/Virtualization`, Last accessed on 14/09/2022.

[21] RODRÍGUEZ-HARO, Fernando, et al. A summary of virtualization techniques. *Procedia Technology*, 2012, 3: 267-272.

[22] MARSHALL, David. Understanding full virtualization, paravirtualization, and hardware assist. *VMWare White Paper*, 2007, 17: 725.

[23] What is a micro VM?, `https://www.techtarget.com/searchsecurity/definition/micro-VM-micro-virtual-machine`, Last accessed on 05/04/2023.

[24] MIEDEN, Philipp; PARTARRIEU, Philippe. Performance analysis of KVM-based microVMs orchestrated by Firecracker and QEMU. Technical Report. *University of Amsterdam*, 2019.

[25] *Virtual I/O Device (VIRTIO) Version 1.1.* Edited by Michael S. Tsirkin and Cornelia Huck. 20 December 2018. OASIS Committee Specification Draft 01 / Public Review Draft 01. `https://docs.oasis-open.org/virtio/virtio/v1.1/csprd01/virtio-v1.1-csprd01.html`. Latest version: `https://docs.oasis-open.org/virtio/virtio/v1.1/virtio-v1.1.html`.

[26] QEMU Wiki - Main Page, `https://wiki.qemu.org/Main_Page`, Last accessed on 14/09/2022.

[27] Kernel Virtual Machine, `https://www.linux-kvm.org/page/Main_Page`, Last accessed on 14/09/2022.

[28] Solo5 repository, `https://github.com/Solo5/solo5`, Last accessed on 14/09/2022.

[29] Xen Project, `https://xenproject.org`, Last accessed on 15/10/2023.

[30] Virtualization - Xen Project, `https://xenproject.org/users/virtualization/`, Last accessed on 15/10/2023.

[31] Secure and fast microVMs for serverless computing, `https://firecracker-microvm.github.io/`, Last accessed on 05/04/2023.

[32] Firecracker Repository - Creating Custom rootfs and kernel Images, `https://github.com/firecracker-microvm/firecracker/blob/main/docs/rootfs-and-kernel-setup.md`, Last accessed on 07/10/2023.

[33] OSv repository, `https://github.com/cloudius-systems/osv`, Last accessed on 4/03/2023.

[34] KIVITY, Avi, et al. OSv—optimizing the operating system for virtual machines. In: *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*. 2014. p. 61-72..

[35] Making OSv Run on Firecracker, `http://blog.osv.io/blog/2019/04/19/making-OSv-run-on-firecraker/`, Last accessed on 17/09/2022.

[36] ChromiumOS Virtual Machine Monitor, `https://chromium.googlesource.com/chromiumos/platform/crosvm/`, Last accessed on 21/04/2023.

[37] Book of crosvm, `https://crosvm.dev/book/introduction.html`, Last accessed on 21/04/2023.

[38] Cloud Hypervisor Repository, `https://github.com/cloud-hypervisor/cloud-hypervisor`, Last accessed on 21/04/2023.

[39] The rust-vmm Community, `https://github.com/rust-vmm/community`, Last accessed on 21/04/2023.

[40] config(5) - NetBSD Manual Pages, `https://man.netbsd.org/config.5`, Last accessed on 03/10/2023.

[41] config(9) - NetBSD Manual Pages, `https://man.netbsd.org/config.9`, Last accessed on 03/10/2023.

[42] autoconf(9) - NetBSD Manual Pages, `https://man.netbsd.org/autoconf.9`, Last accessed on 03/10/2023.

[43] driver(9) - NetBSD Manual Pages, `https://man.netbsd.org/driver.9`, Last accessed on 03/10/2023.

[44] Multiboot - OSDev Wiki, `https://wiki.osdev.org/Multiboot`, Last accessed on 06/10/2023.

[45] OSDev Wiki - Paging, `https://wiki.osdev.org/Paging`, Last accessed on 07/10/2023.

[46] OSDev Wiki - Interrupt Descriptor Table, `https://wiki.osdev.org/Interrupt_Descriptor_Table`, Last accessed on 07/10/2023.

[47] OSDev Wiki - Global Descriptor Table, `https://wiki.osdev.org/Global_Descriptor_Table`, Last accessed on 07/10/2023.

[48] OSDev Wiki - Segmentation, `https://wiki.osdev.org/Segmentation`, Last accessed on 07/10/2023.

[49] Multiboot Specification version 0.6.96, `https://www.gnu.org/software/grub/manual/multiboot/multiboot.html`, Last accessed on 06/10/2023.

[50] 1. The Linux/x86 Boot Protocol — The Linux Kernel documentation, `https://www.kernel.org/doc/html/v5.6/x86/boot.html`, Last accessed on 06/10/2023.

[51] Wikipedia - POSIX, `https://en.wikipedia.org/wiki/POSIX`, Last accessed on 03/10/2023.