ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΉΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

-ΕΙΣΑΓΕΤΕ ΤΟΝ ΤΟΜΕΑ-
-ΕΙΣΑΓΕΤΕ ΤΟ ΕΡΓΑΣΤΗΡΙΟ-

-Εισάγετε τον τίτλο της διπλωματικής εργασίας-

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

-Εισάγετε το σωστό άρθρο (της/του/των)-

-Εισάγετε το όνομα, αρχικό πατρώνυμου και επώνυμο των συγγραφέων-

-another author, whose name is here-

Επιβλέπων: -Εισάγετε το όνομα, αρχικό πατρώνυμου και επίθετο-
-Εισάγετε τον τίτλο του επιβλέποντα-

Αθήνα, -Εισάγετε το μήνα και το έτος κατάθεσης της εργασίας-

ΕΘΝΙΚΌ ΜΕΤΣΌΒΙΟ ΠΟΛΥΤΕΧΝΕΊΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
-ΕΙΣΑΓΕΤΕ ΤΟΝ ΤΟΜΕΑ-
-ΕΙΣΑΓΕΤΕ ΤΟ ΕΡΓΑΣΤΗΡΙΟ-

-Εισάγετε τον τίτλο της διπλωματικής εργασίας-

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

-Εισάγετε το σωστό άρθρο (της/του/των)-

-Εισάγετε το όνομα, αρχικό πατρώνυμου και επώνυμο των συγγραφέων-

Επιβλέπων: -Εισάγετε το όνομα, αρχικό πατρώνυμου και επίθετο-
-Εισάγετε τον τίτλο του επιβλέποντα-

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την -εισάγετε ημερομηνία-.

.....	.....	.....
-Εισάγετε ονοματεπώνυμο-	-Εισάγετε ονοματεπώνυμο-	-Εισάγετε ονοματεπώνυμο-
-Εισάγετε τίτλο-	-Εισάγετε τίτλο-	-Εισάγετε τίτλο-

Αθήνα, (Εισάγετε το μήνα και το έτος κατάθεσης της εργασίας).

.....
(Εισάγετε όνομα, αρχικό πατρώνυμου και επίθετο συγγραφέα)
(Εισάγετε τον απονεμηθέντα τίτλο του συγγραφέα)

Περίληψη

Τα τελευταία χρόνια το cloud computing αποτελεί ένα σημαντικό κεφάλαιο στη σύγχρονη επιστήμη υπολογιστών. Η κύρια τεχνολογία που χρησιμοποιείται, προκειμένου να μπορεί να υποστηριχθεί το cloud computing είναι αυτή της εικονικοποίησης. Με αυτό τον τρόπο ένα φυσικό μηχάνημα, μπορεί να φιλοξενήσει πολλά εικονικά μηχανήματα, κάθε ένα από τα οποία αποτελεί έναν αυτοδύναμο υπολογιστή. Ωστόσο, αποτελεί συχνό φαινόμενο οι εικονικές αυτές μηχανές να χρησιμοποιούνται για την εκτέλεση μίας και μόνο εφαρμογής. Αυτό έχει ως αποτέλεσμα, να χαραμείζονται πόροι σε ενέργειες που δε χρειάζονται από την εφαρμογή, αλλά είναι απαραίτητες για το λειτουργικό σύστημα στο οποίο τρέχουν αυτές οι εφαρμογές.

Μία νεότερη τάση για την υποστήριξη του cloud computing είναι τα containers, που προσφέρουν ελαφρύτερη εικονικοποίηση με γρηγορους χρόνους εκτέλεσης, μικρή κατανάλωση μνήμης και άλλα πλεονεκτήματα. Από την άλλη, παρουσιάζουν αρκετά σημαντικά ζητήματα που αφορούν την ασφάλεια. Ένα από τα ζητήματα αυτά είναι, εκείνο της απομόνωσης το οποίο αναγκάζει σε αρκετές περιπτώσεις να οδηγεί στη χρήση εικονικών μηχανών για τη φιλοξενία των containers, χάνοντας αρκετά από τα πλεονεκτήματα τους.

Μία ακόμη προσέγγιση στο θέμα είναι οι unikernels. Πρόκειται για μία εικόνα εικονικής μηχανής, με ένα μόνο address space το οποίο κατασκευάζεται από library operating systems και είναι ειδικευμένο για μία συγκεκριμένη εφαρμογή. Πιο απλά, περιέχει τον κώδικα της εφαρμογής και ακριβώς ό,τι κομμάτι του λειτουργικού συστήματος χρειάζεται η εφαρμογή για να λειτουργήσει η διεργασία (drivers, βιβλιοθήκες, κ.λ.π.), ενοποιημένα σαν ένα αυτόνομο πρόγραμμα που μπορεί να τρέξει ως εικονική μηχανή. Οι unikernels καταφέρνουν να έχουν γρήγορους χρόνους εκκίνησης και μικρή κατανάλωση μνήμης, χωρίς να θυσιάζεται η ασφάλεια Εν τούτοις, ένα πρόβλημα είναι ότι οι unikernels υποστηρίζουν μία και μόνο διεργασία, με αποτέλεσμα να μην μπορούν εφαρμογές με παραπάνω από μία διεργασίες να εκτελεστούν σε unikernels.

Σκοπός, λοιπόν, αυτής της εργασίας είναι η υλοποίηση ενός μηχανισμού που θα επιτρέπει σε εφαρμογές με περισσότερες από μία διεργασίες να μπορούν να εκτελεστούν και σε unikernels. Επιπλέον, υλοποιείται και ένας απλός μηχανισμός για επικοινωνία μεταξύ των εικονικών μηχανών, στα πρότυπα του pipe.

Λέξεις-Κλειδιά: εικονικοποίηση, εικονικές μηχανές, ενδοεπικοινωνία εικονικών μηχανών, unikernel, kvm, QEMU

Abstract

In recent years cloud computing is one important chapter of modern commputer science. The main technology used in order to support cloud computing is virtualization. Virtualization makes possible for a physical machine to host many virtual machines, each one of which is a self-sufficient computer. However, virtual machines are often used to execute a single application. As a result, resources are devoted to actions that are not needed by the application, but they are necessary for the operating system in which the applications run.

An another technology to support cloud computing is containers which offer lightweight virtualization, fast instantiation times and small per-instance memory footprints among other features. On the other hand, containers have several important security issues. Isolation is one of these issues , which in several cases leads to the use of virtual machines to host the containers, losing several of their advantages.

A further approach in cloud computing is unikernerls. Unikernels are specialised, single-address-space machine images constructed by using library operating systems and are specialised for one application. In somewhat simplified terms, unikernels consist of the application's source code and the parts of an opperating system that are necessary for the proccess to run (drivers, libraries, etc.) consolidated as a stand-alone virtual machine. The Unikernels manage to have fast instantiation times, small memory footprints, without sacrificing security. However, one of the problems of unikernels is that they are single-proccess and as a result multi-proccess applcations are not able to run on unikernels.

The purpose of this thesis is to implement a mechanism that will enable the execution of multi-proccess applications on unikernels. Furthermore, a pipe-like mechanism for inter-vm communication is impemented.

Keywords: virtualization, virtual machines, inter-vm communication, unikernel, kvm, QEMU

Περιεχόμενα

Κατάλογος σχημάτων

Κεφάλαιο 1

Εισαγωγή

Στις μέρες μας το cloud computing έχει γίνει ένα από τα ταχύτερα αναπτυσσόμενα και ενδιαφέροντα θέματα στην επιστήμη των υπολογιστών. Το cloud computing δίνει τη δυνατότητα σε απομακρυσμένους χρήστες να αποκτάνε πρόσβαση σε υπολογιστικούς πόρους (αποθηκευτικός χώρος, εφαρμογές, υπηρεσίες κ.λ.π.) όταν χρειαστούν από τους χρήστες. Ιδιαίτερα, τα τελευταία χρόνια το cloud έχει μπει και στις ζωές των απλών χρηστών. Η χρήση των προσωπικών υπολογιστών αρχίζει να αλλάζει σημαντικά, καθώς πλέον προγράμματα και δεδομένα απομακρύνονται από τους προσωπικούς υπολογιστές για να εκτελεστούν και να αποθηκευτούν στο λεγόμενο cloud.

Μία από τις βασικές τεχνολογίες που κρύβονται πίσω από το cloud είναι αυτή της εικονικοποίησης. Χάρις την εικονικοποίηση, μπορούμε σε ένα φυσικό μηχάνημα να φιλοξενήσουμε πολλά εικονικά μηχανήματα κάθε ένα από τα οποία είναι ανεξάρτητο. Με αυτό τον τρόπο, μπορούμε να αξιοποιήσουμε καλύτερα τους φυσικούς πόρους του μηχανήματος και να τους διαμοιράσουμε όπως επιθυμούμε μεταξύ των εικονικών μηχανών. Ακόμα, η εικονικοποίηση δημιούργησε και κάποιες νέες δυνατότητες, όπως αυτή της μεταφοράς εικονικών μηχανών σε διαφορετικό φυσικό μηχάνημα, η δημιουργία αντιγράφων εικονικών μηχανών, αλλά και περισσότερη ασφάλεια, αφού ένα πρόβλημα σε ένα εικονικό μηχάνημα δε θα επηρεάσει ούτε το φυσικό ούτε τα υπόλοιπα εικονικά μηχανήματα.

Ένα συχνό φαινόμενο κατά τη χρήση της εικονικοποίησης, είναι η χρήση μία εικονικής μηχανής με συμβατικά λειτουργικά συστήματα για την υποστήριξη μίας και μόνο υπηρεσίας. Όπως είναι φυσικό το λειτουργικό σύσστημα μέσα στην εικονική μηχανή χρειάζεται κάποιους πόρους για να λειτουργήσει, ενώ συχνά μπορεί να εκτελεί λειτουργίες οι οποίες δε χρειάζονται από την εφαρμογή. Ακόμα, ο κώδικας όλου του λειτουργικού συστήματος αυξάνει αρκετά και το μέγεθος σε μνήμη της εικονικής μηχανής. Γίνεται λοιπόν, κατανοητό ότι σπαταλούνται πόροι, οι οποίοι θα μπορούσα να διατεθούν είτε στην ίδια την υπηρεσία είτε σε κάποια άλλη.

Για την επίλυση του παραπάνω προβλήμαατος αναζητήθηκαν λύσεις για να γίνει η εικονικοποίηση πιο ελαφρυά, αλλά και να μη σπαταλούνται πόροι σε αχρείαστες λειτουργίες. Μία από αυτές τις λύσεις, ήταν η εικονικοποίηση σε επίπεδο λειτουργικού συστήματος ή διαφορετικά containerization. Με τη συγκεκριμένη μέθοδο ο πυρήνας επιτρέπει την ύπαρξη πολλαπλών απομονωμένων user-space instances, που ονομάζονται containers. Τα containers μοιράζονται μεταξύ τους το λειτουργικό σύστημα στα οποία εκτελούνται, ενώ ταυτόχρονα παρέχουν ένα απομονωμένο περιβάλλον για τις διεργασίες μέσα σε αυτό.

Η τεχνολογιά των containers, όπως κάθε άλλη τεχνολογιά δε θα μπορούσε να μην έχει και κάποια μειονεκτήματα. Ένα από τα κύρια μειονεκτήματα, είναι αυτό της ασφάλειας. Τα containers μοιράζονταai τον ίδιο πυρήνα του host, ενώ οι μηχανισμοί απομόνωσης δεν είναι το ίδιο ισχυροί με αυτούς στα εικονικά μηχανήματα. Μάλιστα, έχουν αναφερθεί περιπτώσεις όπου από ένα container μπορούσαν να παρθούν πληροφορίες και δεδομένα τόσο για το host, όσο και για άλλα containers [5]. Όλα αυτά έχουν οδηγήσει σε περιπτώσεις όπου τα containers χρησιμοποιούνται πάνω από ένα πλήρη λειτουργικό σύστημα το οποίο τρέχει μέσα σε μία εικονική μηχανή.

Μία ιδέα που αρχίζει να αποκτά όλο και περισσότερο ενδιαφέρον και προσοχή είναι αυτή των unikernels. Η βάση της ιδέας, είναι η κατασκευή ειδικευμένω εικονικών μηχανών για κάθε ξεχωριστή υπηρεσία. Στην εικονική αυτή μηχανή δε χρειάζεται να υπάρχει ένα πλήρη λειτουργικό σύστημα, αλλά μόνο τα κομμάτια αυτού τα οποία είναι απαραίτητα για την εκτέλεση της υπηρεσίας. Με αυτό τον τρόπο, μειώνεται σημαντικά το μέγεθος των εικονικών μηχανών, ενώ πλέον οι πόροι χρησιμοποιούνται ακριβώς για τις λειτουργίες που χρειάζεται η υπηρεσία. Τέλος, εφόσον αναφερόμαστε σε εικονικές μηχανές, η απομόνωση τους είναι κάτι το οποίο έχουν φροντίσει ήδη οι ελεγκτές.

1.1 Σκοπός

Η φιλοσοφία των unikernels είναι ότι κάθε εικονική μηχανή θα έχει ένα συγκεκριμένο σκοπό. Για το λόγο αυτό οι unikernels δεν υποστηρίζουν παραπάνω από μία διεργασία σε κάθε εικονική μηχανή. Από την άλλη, υπάρχουν unikernel frameworks τα οποία επιτρέπουν την υποστήριξη περισσότερων από ένα νήμα. Ωστόσο οι περισσότερες εφαρμογές και υπηρεσίες στο cloud είναι φτιαγμένες για ένα πλήρη λειτουργικό σύστημα. Επομένως, προκειμένου να μπορούν να τρέξουν σε ένα unikernel θα πρέπει να αλλάξει άλλοτε περισσότερο και άλλοτε λιγότερο ο σχεδιασμός τους. Ιδιαίτερα, οι εφαρμογές που χρησιμοποιούν παραπάνω από μία διεργασίες θα πρέπει να αλλάξουν

σε ένα μοντέλο με μία μόνο διεργασία.

Ο σκοπός της συγκεκριμένης εργασίας είναι να υλοποιήσει, κρατώντας το single-proccess χαρακτηριστικό των unikernels, ένα μηχανισμό για την υποστήριξη των εφαρμογών που απαιτούν παραπάνω από μία διεργασία.

Αρχικά έγινε μία μελέτη γύρω από τα υπάρχοντα unikernel frameworks, παρατηρώντας τα χαρακτηριστικά του κάθε ενός. Στη συνέχεια σε ένα από αυτά τα unikernels (rumprun) υλοποιήθηκαν οι δύο παρακάτω λειτουργίες:

- ένας μηχανισμός για επικοινωνία μεταξύ των vms, ο οποίος είναι στα πρότυπα της κλήσης συστήματος pipe (POSIX). Ουσιαστικά πρόκειται για την υλοποίηση της pipe σε επίπεδο εικονικών μηχανών.
- ένας μηχανισμός για την υποστήριξη της κλήσης συστήματος fork από τους unikernels. Όταν μία εφαρμογή θα χρησιμοποιεί τη συγκεκριμένη κλήση συστήματος, θα δημιουργείται μία νέα εικονική μηχανή κλώνος της αρχικής και θα ξεκινά την εκτέλεση της, ακριβώς μετά την κλήση συστήματος fork.

1.2 Οργάνωση Κειμένου

Στη συνέχεια παρουσιάζεται αναλυτικά η περιγραφή του σχεδιασμού και της υλοποίησης των δύο λειτουργιών pipe, fork, γίνεται μία ανασκόπηση στα ήδη υπάρχοντα unikernel frameworks και περιγράφεται το απαραίτητο θεωρητικό υπόβαθρο.

Πιο συγκεκριμένα, στο Κεφάλαιο 2 καλύπτεται το απαραίτητο θεωρητικό υπόβαθρο, κάνοντας μία μικρή αναφορά για το cloud computing, μία εισαγωγή στην εικονικοποίηση και στα λειτουργικά συστήματα.

Στο Κεφάλαιο 3 παρουσιάζονται τα unikernel frameworks που μελετήθηκαν, τα χαρακτηριστικά τους, η φιλοσοφία τους κ.λ.π.

Στο Κεφάλαιο 4 γίνεται αναλυτική περιγραφή του σχεδιασμού και της υλοποίησης των δύο λειτουργιών που αναφέρθηκαν προηγουμένως στο σκοπότης εργασίας (pipe,fork).

Τέλος, στο Κεφάλαιο 5 αναφέρονται τα τελικά συμπεράσματα της παρούσας μελέτης όπως επίσης και πιθανές μελλοντικές επεκτάσεις αυτής της διπλωματικής εργασίας.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

2.1 Cloud computing

Εδώ και αρκετό καιρό γίνεται συχνά ανφορά στο ”Cloud”, αλλά και σε αυτή την εργασία έχει αναφερθεί αρκετές φορές ο όρος ”cloud computing”. Παρά τη δημοφιλία του όρου, δεν υπάρχει ακριβής ορισμός. Εντούτοις, θα χρησιμοποιηθεί ο ορισμός που έχει δωθεί από το NIST (National Institute of Standards and Technology of USA) [11]. Το cloud computing (υπολογιστικό νέφος με ελληνικούς όρους) είναι η κατ’ αίτηση διαδικτυακή κεντρική διάθεση υπολογιστικών πόρων (όπως δίκτυο, εξυπηρετητές, εφαρμογές και υπηρεσίες) με υψηλή ευελιξία, ελάχιστη προσπάθεια από τον χρήστη και υψηλή αυτοματοποίηση. Ωστόσο, είναι συχνό φαινόμενο να αναφερόμαστε με το συγκεκριμένο όρο τόσο στις υπηρεσίες που είναι διαθέσιμες μέσω διαδικτύου με τη μορφή μίας υπηρεσίας ιστού όσο και στο υλικό και λογισμικό που απαρτίζουν την υποδομή που προσφέρει αυτές τις υπηρεσίες.

2.1.1 Χαρακτηριστικά του cloud computing

Πέρα από τον ορισμό το NIST [11], περιγράφει και τα ακόλουθα βασικά χαρακτηριστικά του cloud computing:

- παροχή υπηρεσίας κατ’ απαίτηση: Οι χρήστες έχουν τη δυνατότητα να τροποποιήσουν τους πόρους που προσφέρονται, χωρίς την ανθρώπινη διαμεσολάβηση από την πλευρά του παρόχου του cloud.
- ευρυζωνική δικτυακή πρόσβαση: Οι υπηρεσίες είναι διαθέσιμες από το διαδίκτυο.
- Διάθεση πόρων σε περισσότερους χρήστες: Οι πόροι μπορούν

να διατεθούν ανάμεσα σε πολλούς χρήστες χωρίς να δημιουργούνται προβλήματα.

- ταχεία ελαστικότητα/επεκτασιμότητα: Οι πόροι μπορούν να ανακατεμηθούν, είτε αυτόματα είτε κατ' απαίτηση, χωρίς περιορισμούς και χωρίς να επηρεάζεται η ομαλή λειτουργία άλλων υπηρεσιών.
- μετρήσιμη υπηρεσία: Η χρήση των πόρων καταγράφεται και παρουσιάζεται τόσο στον πάροχο όσο και στον πελάτη.

2.1.2 Μοντέλα υπηρεσιών

Σύμφωνα με το NIST [11], τα μοντέλα υπηρεσιών του cloud είναι τα εξής:

Λογισμικό ως Υπηρεσία (SaaS)

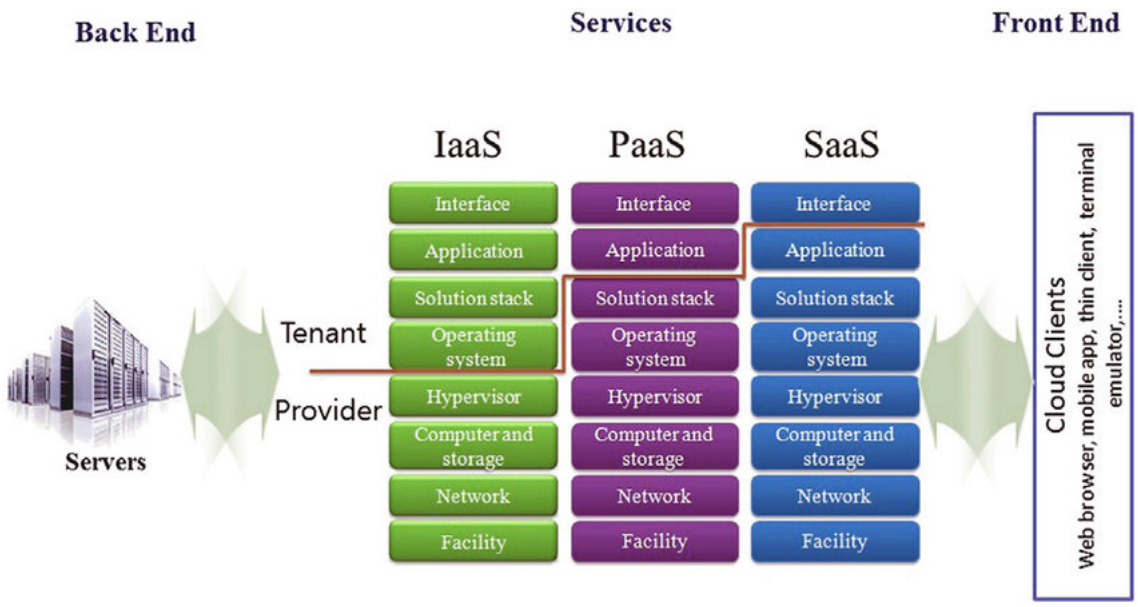
Η υπηρεσία που παρέχεται στο χρήστη είναι αυτή μία εφαρμογής που τρέχει στην υποδομή του παρόχου. Ο χρήστης δε χρειάζεται να γνωρίζει τίποτα σχετικά με την υποδομή που υποστηρίζει την εφαρμογή, ούτε έχει τη δυνατότητα να τροποποιήσει πολλά πράγματα πέρα ίσως από κάποιες ρυθμίσεις που προσφέρει η εφαρμογή. Οι εφαρμογές αυτές είναι προσβάσιμες μέσα από κάποιο λογιμικό πελάτη που προσφέρει ο πάροχος ή ακόμα και από ένα πειηγητή ιστού (web browser).

Πλατφόρμα ως υπηρεσία (PaaS)

Σε αυτή την περίπτωση ο χρήστης έχει τη δυνατότητα της δημιουργίας εφαρμογών ή/και περιβάλλοντων εφαρμογών, χρησιμοποιώντας προγραμματιστικά εργαλεία και υπηρεσίες που παρέχονται από τον πάροχο. Ο χρήστης δεν μπορεί να ελέγξει και να διαχειριστεί την υποδομή του cloud, ωστόσο έχει τον έλεγχο των αναπτυγμένων εφαρμογών και ενδεχομένως των ρυθμίσεων διαμόρφωσης για το περιβάλλον φιλοξενίας αυτών των εφαρμογών.

Υποδομή ως υπηρεσία (IaaS)

Οι παροχές προς το χρήστη είναι υπολογιστικοί πόροι όπως επεξεργαστές, αποθηκευτικός χώρος και δίκτυο στα οποία μπορεί αναπτύξει και να τρέξει λειτουργικά συστήματα και εφαρμογές. Παρά τη δυνατότητα να τροποποιήσει την ποσότητα των πόρων, ο χρήστης δεν έχει παραπάνω έλεγχο ως προς την υποδομή του cloud.



Σχήμα 2.1: Cloud services

Όπως βλέπουμε και από το σχήμα 2.1, όσο περισσότερη ευελιξία προσφέρει ένα μοντέλα τόσο περισσότερο έλεγχο έχει και ο χρήστης στις υπηρεσίες που λαμβάνει. Για παράδειγμα στην περίπτωση SaaS, ο χρήστης έχει το λιγότερο έλεγχο, σε αντίθεση με την περίπτωση IaaS όπου ο χρήστης μπορεί πλέον να αποκτήσει έλεγχο ακόμα και στο λειτουργικό σύστημα.

2.2 Εικονικοποίηση

Στην επιστήμη των υπολογιστών ο όρος εικονικοποίηση (virtualization) περιγράφει ένα μηχανισμό αφαίρεσης, που με τη χρήση ”εικονικών υπολογιστικών πόρων”, έχει ως σκοπό την απόκρυψη λεπτομερειών για την υλοποίηση ή την κατάσταση των φυσικών πόρων. Με το μηχανισμό αυτό ένας φυσικός πόρος μπορεί να παρουσιαστεί ως μία πλειάδα εικονικών φυσικών πόρων, ή αντίστροφα μία πλειάδα φυσικών πόρων να παρουσιαστούν ως ένας ενιαίος εικονικός πόρος. Ανάλογα σε ποιο επίπεδο στη στοίβα του λογισμικού υλοποιείται η εικονικοποίηση, υπάρχουν και τα αντίστοιχα πλεονεκτήματα, αλλά σε γενικές γραμμές κάποια από τα πλεονεκτήματα της εικονικοποίησης είναι:

- Καλύτερη αξιοποίηση και διαμοιρασμός των πόρων.
- Ασφάλεια μεσω της απομόνωσης των υπηρεσιών.

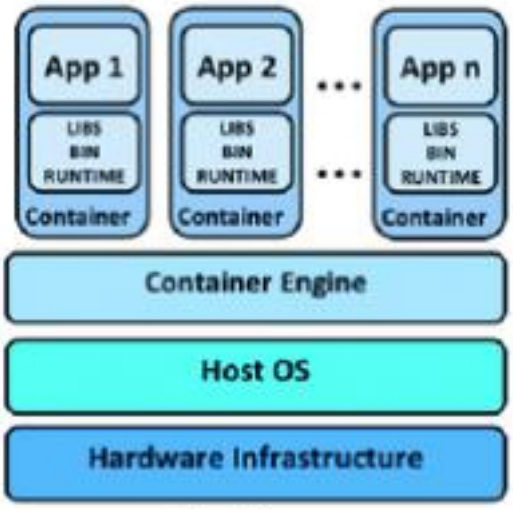
- Δυνατότητα προσομοίωσης περιβαλλόντων που δεν είναι φυσικά διαθέσιμα.
- Δυνατότητα αποθήκευσης, μεταφοράς και επαναφοράς της κατάστασης των υπηρεσιών που προσφέρονται μέσω της εικονικοποίησης.

Η εικονικοποίηση μπορεί να υλοποιηθεί σε διάφορα επίπεδα, όπως στο δίκτυο, στον αποθηκευτικό χώρο, στο hardware, στο λειτουργικό σύστημα κ.α. Η συγκεκριμένη εργασία ασχολείται κυρίως με την εικονικοποίηση σε επίπεδο υλικού, ωστόσο είναι συχό φαινόμενο να συγκρίνονται οι unikernels με τα containers. Για το λόγο αυτό θα γίνει μία σύντομη παρουσίαση της εικονικοποίησης σε επίπεδο λειτουργικού συστήματος, ενώ μετά από αυτή την περιγραφή, όταν χρησιμοποιείται ο όρος εικονικοποίηση θα εννοείται εικονικοποίηση επιπέδου υλικού.

2.2.1 Εικονικοποίηση σε επίπεδο λειτουργικού συστήματος

Η εικονικοποίηση σε επίπεδο λειτουργικού συστήματος είναι νεότερη σε σχέση με αυτή σε επίπεδο υλικού, ωστόσο έχει καταφέρει να συγκεντρώσει αρκετή προσοχή τα τελευταία χρόνια. Ο πυρήνας του λειτουργικού συστήματος επιτρέπει την ύπαρξη παραπάνω από ενα, απομονωμέα userspace instances τα οποία ονομάζονται containers, virtualization engines ή jails. Όπως φαίνεται στο σχήμα 2.2 όλα τα containers μοιράζονται τον ίδιο πυρήνα πάνω από τον οποίο βρίσκεται το στρώμα που είναι υπεύθυνο για την εικονικοποίηση.

Το γεγονός ότι τα containers μοιράζονται τον ίδιο πυρήνα κάνει το μέγεθο τους αρκετά μικρό, αφού δε χρειάζεται να συμπεριλάβουν κάποιο λειτουργικό σύστημα. Επιπλέον, οι χρόνοι εκκίνησης τους είναι αρκετά μικροί, χωρίς να χρειάζεται να καταναλωθεί πολύ μνήμη για τη λειτουργεία των containers. Επιπροσθέτως, τα containers είναι ανεξάρτητα από το φυσικό μηχάνημα, καθώς η εικονικοποίηση γίνεται με τη χρήση του κατάλληλου λογισμικού χωρίς να χρειάζεται κάποια υποστήριξη από το hardware. Τέλος, τα containers επιτρέπουν το εύκολο πακετάρισμα τόσο των εφαρμογών όσο και των κατάλληλων ρυθμίσεων, που σε συνδυασμό με την ανεξαρτησία από το hardware, δίνουν τη δυνατότητα για την εύκολη μεταφορά τους σε διαφορετικά μηχανήματα.



Σχήμα 2.2: Εικονικοποίηση σε επίπεδο λειτουργικού συστήματος

Δυστυχώς, όλα αυτά τα πλεονεκτήματα έρχονται με κάποιο κόστος, με το μεγαλύτερο να είναι αυτό της ασφάλειας. Αν και οι υποστηρικτές των containers θεωρούν ότι με τις κατάλληλες ρυθμίσεις τα containers μπορούν να γίνουν ασφαλή, δεν προσφέρουν την ίδια ασφάλεια με την εικονικοποίηση σε επίπεδο υλικού [6]. Τα containers δεν έχουν τη δυνατότητα να προσφέρουν τόσο ισχυρή απομόνωση όσο προσφέρει η εικονικοποίηση σε επίπεδο υλικού. Τα αποτελέσματα από την παραβίαση της απομόνωσης μπορεί να είναι από την απόκτηση πληροφοριών σχετικά με το host μηχάνημα ή άλλων containers, μέχρι την απόκτηση ελέγχου του λειτουργικού συστήματος στον host [5].

Η σύγκριση τψν δύο ειδών εικονικοποίησης είναι ένα ανοιχτό ζήτημα το οποίο δύσκολα θα λυθεί σύντομα [4]. Άλλωστε και τα δύο είδη, αναπτύσσονται ταχύτατα, προσφέροντας συνεχώς νέες δυνατότητες και μειώνοντας τα προβλήματα που υπάρχουν. Σε μία προσπάθεια να ενοποιηθούν τα θετικα των δύο ειδών, είναι συχνή η χρήση εικονικών μηχανών μέσα στις οποίες τρέχουν containers. Με αυτό τον τρόπο βελτιώνεται η ασφάλεια σε βάρος όμως της απόδοσης, καθώς πλέον χρειάζονται πόροι και για την εκτέλεση του απαραίτητου λογισμικού που θα επιτρέψει την υπάρξη εικονικών μηχανών.

2.2.2 Εικονικοποίηση σε επίπεδο υλικού

Η εικονικοποίηση σε επίπεδο υλικού, ή απλά εικονικοποίηση για το υπόλοιπο της συγκεκριμένης εργασίας, επιτρέπει τη λειτουργία ενός ολόκληρου υπολογιστικού συστήματος μέσα από έναν άλλον. Τα εμφωλευμένα υπολογιστικά συστήματα αναφέρονται ως guests,

ενώ το υπολογιστικό σύστημα στο οποίο πραγματοποιείται η εικονικοποίηση αναφέρεται ως host. Ένας guest εκτελείται μέσα σε μία εικονική μηχανή (virtual machine - VM), την οποία οι Popek και Goldberg [12] ορίζουν ως εξής: “ένα αποδοτικό και απομονωμένο αντίγραφο μιας πραγματικής μηχανής”. Κάθε εικονική μηχανή είναι ανεξάρτητη τόσο από το host, όσο και άπο άλλες τυχόν εικονικές μηχανές που μπορεί να υπάρχουν. Με αυτού του είδους την εικονικοποίηση, κάθε εικονική μηχανή έχει την ψαυδαίσθηση ότι κατέχει αποκλειστικά τους πόρους που της έχουν διατεθεί.

Απαραίτητο συστατικό για να επιτευχθούν όλα τα παραπάνω είναι ο επόπτης (hypervisor). Ο επόπτης, σύμφωνα με τους Popek και Goldberg έχει τρία συγκεκριμένα χαρακτηριστικά [12]:

1. ο επόπτης παρέχει ένα περιβάλλον όμοιο με το πραγματικό υπολογιστικό σύστημα,
2. τα προγράμματα που τρέχουν στο περιβάλλον που δημιουργείται θα πρέπει να έχουν στη χειρότερη περίπτωση μικρές μειώσεις στην ταχύτητα εκτέλεσης,
3. ο επόπτης έχει πλήρη έλεγχο των πόρων.

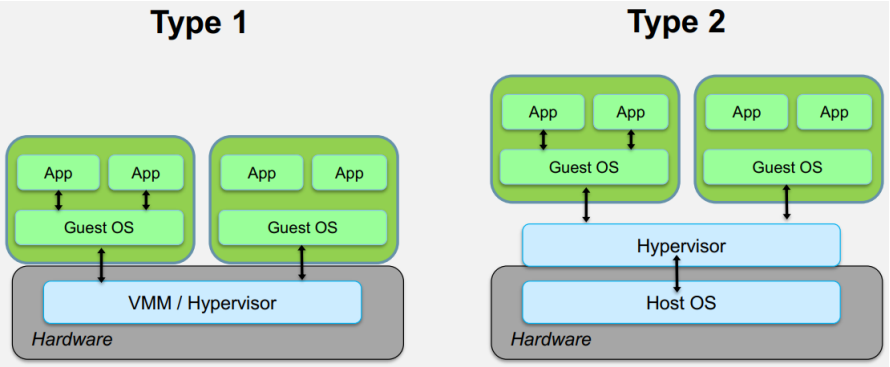
Ο ρόλος του επόπτη μοιάζει με το ρόλο του λειτουργικού συστήματος, με τις εικονικές μηχανές να έχουν το ρόλο των διεργασιών. Ο επόπτης χρονοδρομολογεί τις εικονικές μηχανές στη CPU, μοιράζει γενικότερα τους διαθέσιμους πόρους στα εικονικά μηχανήματα, εκτελεί εκ μέρους των εικονικών μηχανών ”προνομιούχες εντολές” και είναι αυτός που διαχειρίζεται τις εικονικές μηχανές (δημιουργία, εκκίνηση, τερματισμός κ.λ.π).

Όπως βλέπουμε στο σχήμα 2.3 υπάρχουν δύο διαφορετικά είδη εποπτών, ανάλογα με το που βρίσκονται στο σύστημα.

- **Type 1 - Bare metal/native**
  Σε αυτή την περίπτωση ο επόπτης εκτελείται πάνω από το υλικό του συστήματος, χωρίς να παρεμβάλλεται κάποιο λειτουργικό σύστημα. Είναι υπεύθυνος για τη χρονοδρομολόγηση των εικονικών μηχανών, τη διαχείριση της μνήμης και των πόρων καθώς και άλλων λειτουργιών. Χρησιμοποιεί δικούς του drivers για τη διαχείριση του υλικού και για την εξυπηρέτηση των I/O λειτουργιών των guest.

- **Type 2 - Hosted**
  Σε αυτή την περίπτωση ο επόπτης εκτελείται πάνω από ένα λειτουργικό σύστημα, σαν ένα συνηθισμένο πρόγραμμα στο userspace. Αποτελούν ένα ενδιάμεσο στρώμα μεταξύ του host και του guest, δημιουργώντας το κατάλληλο περιβάλλον για τις εικονικές μηχανές. Δε χρειάζονται ειδικοί drivers για το υλικό, καθώς αυτοί παρέχονται από το λειτουργικό σύστημα του host. Από την

άλλη, οι I/O λειτουργίες και διάφορες άλλες ”προνομιούχες εντολές” του guest θα πρέπει να περάσουν μέσα από τον hypervisor και μετά να εξυπηρετηθούν από τον host, δημιοργώντας επιπλέον κόστος στην εκτέλεση τους.



Σχήμα 2.3: Εικονικοποίηση σε επίπεδο υλικού

Εκτός από το διαχωρισμό των εποπτών, γίνεται και διαχωρισμός και στις τεχνικές που χρησιμοποιούντα για να επιτευχθεί η εικονικοποίηση. Οι τεχνικές αυτές βασίζονται τόσο στη δυνατότητα που προσφέρει ειδικό υλικό για την υποστήριξη της εικονικοποίησης, όσο και στο βαθμό συνεργασίας μεταξύ του επόπτη με το εικονικό μηχάνημα. Θα αναφερθούμε σε τρεις από αυτές τις τεχνικές, την πλήρη εικονικοποίηση (full virtualization), την παραεικονικοποίηση (paravirtualization) και τέλος την εικονικοποίηση υποστηριζόμενη από το υλικό (hardware-assisted virtualization).

Full virtualization

Στην πλήρη εικονικοποίηση ο guest δεν έχει υποστεί καμία αλλαγή και εκτελείται όπως και στην περίπτωση που εκτελείται απευθείας σε ένα φυσικό υπολογιστικό σύστημα. Απο τη μεριά του ελεγκτή, θα πρέπει να γίνει πλήρης προσομοίωση του υλικού ώστε να δίνει την ψευδαίσθηση στον guest ότι επικοινωνούν απευθείας με το υλικό. Ωστόσο δημιουργείται ένα πρόβλημα, το λειτουργικό σύστημα του guest εκτελείται σε unprivileged mode, οπότε δεν μπορεί να εκτελέσει προνομιούχες εντολές. Η λύση σε αυτό το πρόβλημα είναι η τεχνική trap-and-emulate [1], κατά την οποία όταν ο guest προσπαθεί να εκτελέσει μία προνομιούχα εντολή, δημιουργείται μία εξαίρεση (trap) στον hypervisor, καθώς ο guest βρίσκεται σε unprivileged mode. Ύστερα, ο hypervisor παίρνει τον έλεγχο και προσομοιώνει ή εκτελεί την προνομοιούχα εντολή για λογαριασμό του guest.

Η συγκεκριμένη λύση δεν είναι αρκετή για όλες τις αρχιτεκτονικές. Μία από αυτές τις αρχιτεκτονικές είναι και η x86. Στην x86 αρχιτεκτονική ορισμένες εντολές δεν προκαλούν trap όταν εκτελούνται

σε unprivileged mode, παρά το γεγονός ότι για την εκτέλεση τους είναι ανάγκη να βρίσκονται σε privileged mode (non-virtualizable instructions). Μία από αυτές τις εντολές είναι η popf, η οποία ενώ αλλάζει τόσο τα ALU flags όσο και τα flags του επεξεργαστή δεν προκαλεί κάποιο trap με αποτέλεσμα να αγνοούνται οι αλλαγές στα flags του επεξεργαστή. Για την αντιμετόπιση του συγκεκριμένου προβλήματος χρησιμοποείται από τους επόπτες η τεχνική του binary translation [1]. Όταν ο guest βρίσκεται σε unprivileged mode όλες οι εντολές εκτελούνται κανονικά, ενώ όταν μεταβεί στον πυρήνα και σε privileged mode τότε ο hypervisor μεταφράζει τις εντολές που δεν προκαλούν trap σε ένα νέο σύνολο εντολών που θα επιφέρουν τις επιθυμητές αλλαγές στην εικονική μηχανή.

Μπορούμε να παρατηρήσουμε εύκολα ότι και στις δύο περιπτώσεις η απόδοση θα μειωθεί σημαντικά, λόγω της ανάμειξης του hypervisor. Ιδιαίτερα στην περίπτωση του binary translation το κόστος της παρακολούθησης και μετάφρασης των εντολών του guest είναι αρκετά μεγάλο.

Paravirtualization

Στην παραεικονικοποίηση ο guest γνωρίζει ότι δεν εκτελείται απευθείας στο υλικό και ”συνεργάζεται” κατάλληλα με τον hypervisor. Απαιτείται λοιπόν, η τροποποίηση του guest, ώστε να αντικατασταθούν οι non-virtualizable εντολές, με κατάλληλες εντολές (hypercalls) που δίνουν τον έλεγχο στο hypervisor και αυτός με τη σειρά του διεκπαιρεώνει την αντίστοιχη λειτουργία. Επιπλέον ο hypervisor μπορεί να υποστηρίξει και κάποιες άλλες εντολές, που χρησιμοποιούνται για άλλες σημαντικές λειτουργίες του πυρήνα, όπως διαχείρηση μνήμης, διαχείριση διακοπών κ.α.

Με αυτό τον τρόπο μειώνεται σημαντικά το κόστος της εικονικοποίησης, αφού ο guest γνωρίζει ότι εκτελείται σε εικονικό περιβάλλον και μπορεί να βελτιστοποιηθεί για τις συγκεκριμένες συνθήκες. Παράλληλα γίνεται δυνατόν να εικονικοποιηθούν αρχιτεκτονικές όπως η x86 χωρίς σημαντική μείωση της απόδοσης αποφεύγοντας το binary translation. Από την άλλη, ο guest τροποποιείται αρκετά για να μπορέσει να ”συνεργαστεί” με τον επόπτη. Συνεπώς, μειώνεται η δυνατότητα μεταφοράς του σε διαφορετικά συστήματα (π.χ. διαφορετικό επόπτη), ενώ ταυτόχρονα αυξάνει το κόστος συντήρησης του καθώς οποιεσδήποτε αλλάγές ή αναβαθμίσεις στον πυρήνα του guest θα πρέπει να τροποποιηθούν κατάλληλα για την υποστήριξη της παραεικονικοποίησης.

Hardware-assisted virtualization

Η συγκεκριμένη μέθοδος ουσιαστικά έχει τα ίδια χαρακτηριστικά με τη μέθοδο της πλήρης εικονικοποίησης. Ο guest δε χρειάζεται να υποστεί καμία αλλαγή, ενώ εισάγεται κατάλληλο hardware το οποίο

διευκολύνει την εικονικοποίηση. Πρόκειται για επεκτάσεις των επεξεργαστών με στόχο την αύξηση της απόδοσης και της ασφάλειας της εικονικοποίησης. Αυτές οι επεκτάσεις είναι γνωστές ως επεκτάσεις εικονικοποίησης (virtualization extensions).

Ένα από τα βασικά στοιχεία αυτών των επεκτάσεων είναι η δημιουργία ενός νέου επιπέδου εκτέλεσης, αυτό του guest στο οποίο μπορούν να εκτελεστούν τόσο privileged όσο και unprivileged εντολές. Συνεπώς, δε χρειάζεται πλέον να γίνονται trap οι privileged εντολές του guest, αλλά αυτές μπορούν να εκτελεστούν απευθείας, χωρίς μάλιστα να επηρεάζεται ο host από την εκτέλεση τους. Οι επεκτάσεις αφορούν ακόμα τη διαχείριση μνήμης, αλλά και των διακοπών. Πλέον η εκτέλεση του guest συνεχίζεται αδιάκοπα μέχρι την ικανοποίηση κάποιας συνθήκες, όπως για παράδειγμα ένα page fault, όπου τότε ο έλεγχος μεταφέρεται στον host (VM exit).

Με αυτό τον τρόπο επιτυγχάνεται υψηλή απόδοση για την εικονικοποίηση χωρίς να είναι απαραίτητες οι αλλαγές στον guest. Ο πιο σημαντικός παράγοντας απόδοσης για την εν λόγω τεχνική είναι το πλήθος των VM exits, καθώς πρόκειται για μία χρονοβόρα διαδικασία [2]. Μάλιστα πολλές από αυτές τις επεκτάσεις έχουν δημιοργηθεί με ακριβώς αυτό το σκοπό, δηλαδή τη μείωση των VM exits.

2.2.3 QEMU/KVM

Το Qemu [3] είναι ένας επόπτης τύπου 2 και πρόκειται για έναν από τους πιο γνωστούς επόπτες. Με την τεχνική του full virtualization και συγκεκριμένα με τη μέθοδο του dynamic binary translation, υποστηρίζει τη φιλοξενεία πολλών λειτουργικών συστημάτων χωρίς να απαιτείται κάποια αλλαγή σε αυτά. Πέρα από τη λειτουργία του επόπτη, μπορεί να λειτουργήσει και ως προσομοιωτής υλικού. Το Qemu έχει τη δυνατότητα να προσομοιώσει διαφορετικές αρχιτεκτονικές από αυτή που διαθέτει ο host, πληθώρα από κάρτες δικτύου, σκληρούς δίσκους, περιφερειακές συσκευές κ.λ.π.. Με αυτό τον τρόπο μπορεί να χρησιμοποιηθεί και για εφαρμογές που έχουν υλοποιηθεί για διαφορετική αρχιτεκτονική να εκτελεστούν στην αρχιτεκτονική που διαθέτει ο host.

Οι εικονικές μηχανές που εκτελούνται στον ίδιο host, συχνά επικοινωνούν μεταξύ τους. Συνήθως πρόκειται για πλήρη υπολογιστικά συστήματα, συνεπώς για την επικοινωνία μεταξύ τους θα μπορούσε να χρησιμοποιηθεί ό,τι χρησιμοποιειταί μεταξύ φυσικών υπολογιστικών συστημάτων (π.χ. TCP/IP sockets). Μάλιστα ειδικά στη συγκεκριμένη περίπτωση οι επόπτες μπορούν να αντιληφθούν ότι πρόκειται για επικοινωνία μεταξύ εικονικών μηχανών που ελέγχουν οπότε και χρησιμοποιούν διάφορες βελτιστοποιήσεις, προκειμένου να μη χρειαστεί να περάσουν τα πακέτα πέρα από το host. Εντούτοις, συχνά πολλές εικονικές μηχανές δε χρειάζονται δίκτυο και δημιουργεί-

ται η ανάγκη για επικοινωνία μεταξύ άλλων εικονικών μηχανών λη ακόμα και με το host. Επιπλεόν πολλές φορές είναι χρήσιμο εικονικές μηχανές να μοιράζονται μνήμη μεταξύ τους . Έτσι έχουν δημιουργηθεί αρκετές μέθοδοι για τον διαμοιρασμό μνήμης και ενδοεπικοινωνίας μεταξύ εικονικών μηχανών, αλλά και μεταξύ του host και των εικονικών μηχανών [14].

Ένας από αυτούς τους μηχανισμούς είναι ο nahanni ή ivshmem (inter-VM shared memory) [10]. Το ivshmem είναι ένας μηχανισμός για το διαμοιρασμό μνήμης του host με τις εικονικές μηχανές που εκτελούνται στο host. Μπορεί να χρησιμοποιηθεί τόσο για διαμοιρασμό μνήμης μεταξύ host και guest, όσο και μεταξύ των guests. Το Qemu επιτρέπει τη χρήση αυτού τού μηχανισμού μέσω μίας PCI συσκευής την οποία πρέπει να υποστηρίξουν οι guests για να χρησιμοποιήσουν τη συγκεκριμένη περιοχή μνήμης.

Πολλές φορές το Qemu χρησιμοποιείται από άλλους επόπτες για την εξομοίωση του υλικού, όπως για παράδειγμα γίνεται από το Xen. Επιπροσθέτως συχνά χρησιμοποιείται σε συνδυασμό με το kvm (στην περίπτωση υποστήριξης hardware-assisted virtualization), πετυχαίνοντας υψηλή απόδοση. Ωστόσο, από μόνο του το kvm δε λειτουργεί ως επόπτης, αλλά ανοίγει μία διεπαφή (/dev/kvm) μέσω της οποίας μπορούν να χρησιμοποιηθούν οι διάφορες λειτουργίες που προσφέρει. Το kvm είναι ένα kernel module του Linux, που μετατρέπει τον πυρήνα σε εναν επόπτη. Σε αυτό τον συνδυασμό (Qemu/Kvm) κάθε εικονική μηχανή εκτελείται ως μία διεργασία.

2.3 Λειτουργικά συστήματα

Το λειτουργικό σύστημα είναι ένα πρόγραμμα το οποίο εκτελείται σε ένα υπολογιστικό σύστημα. Η ειδοποιός διαφορά του με οποιοδήποτε άλλο πρόγραμμα είναι ότι το πρώτο υπάρχει για να εξυπηρετεί το δεύτερο. Το λειτουργικό σύστημα διαχειρίζεται το υλικό του υπολογιστικού συστήματος και παρέχει στους πργαμματιστές των εφαρμογών ένα σαφές αφηρημένο σύνολο πόρων, αντί για το μπερδεμένο σύνολο που υπάρχει στο υπολογιστικό σύστημα. Χωρίς αυτό, ο προγραμματιστής μίας οποιασδήποτε εφαρμογής θα έπρεπε να γράψει και τον κατάλληο κώδικα για το υλικό που χρησιμοποιεί η συγκεκριμένη εφαρμογή (π.χ. αποθήκευση σε σκληρό δίσκο). Ως εκ τούτου θα σπαταλούσε αρκετό χρόνο σε μη βασικές λειτουργίες του προγράμματος. Επιπροσθέτως πολλές από αυτές τις λειτουργίες που αφορούν το υλικό μπορεί να έχουν ήδη υλοποιηθεί από άλλα προγράμματα, σπαταλώντας έτσι χρόνο για ήδη υπάρχουσες λειτουργίες.

Η πληθώρα και οι σημαντικές διαφορές στο σκοπό και στις δυνατότητες των υπολογιστικών συστημάτων οδήγησε στη δημιουργία διαφορετικών ειδών λειτουργικών συστημάτων. Για παράδειγμα,

ένα λειτουργικό σύστημα που έχει δημιουργηθεί με σκοπό την εκτέλεση του σε προσωπικούς υπολογιστές διαφέρει αρκετά από ένα λειτουργικό σύσστημα που χρησιμοποιείται σε ενσωματωμένα συστήματα. Το πιο γνωστό είδος λειτουργικών συστημάτων είναι αυτά του γενικού σκοπού, τα οποία έχουν ως στόχο να μπορούν να εκτελεστούν σε διάφορα υπολογιστικά συστήματα προσφέροντας όσο το δυνατόν περισσότερες λειτουργίες.

Ένας διαφορετικός τρόπος κατηγοριοποίησης των λειτουργικών συστημάτων γίνεται με τη δομή τους. Θα αναφερθούν οι κυριότερες από αυτές τις δομές, μονολιθικά συστήματα, πολυεπίπεδα συστήματα, μικροπυρήνες.

- μονολιθικά συστήματα: η πιο διαδεδομένη οργάνωση, στην οποία ολόκληρουόκληρο το λειτουργικό συστημα εκτελείται ως ένα μοναδικό πρόγραμμα σε κατάσταση πυρήνα. Ουσιαστικά πρόκειται για ένα μεγάλο εκτελέσιμο δυαδικό πρόγραμμα. Κάθε διαδικασία του συστήματος μπορεί να καλέσει οποιαδήποτε άλλη.

- πολυεπίπεδα συστήματα: το λειτουργικό σύστημα οργανώνενται σε μία ιεραρχία επιπέδων, όπου το καθένα δομείται πάνω στο αμέσως κατώτερο του. Κάθε επίπεδο αναλαμβάνει μία διαφορετική λειτουργία και μπορεί να χρησιμοποιείσει μόνο λειτουργίες που υποστηρίζουν κατώτερα επίπεδα.

- μικροπυρήνες: το λειτουργικό σύστημα διαιρείται σε μικρές καλά ορισμένες υπομονάδες, από τις οποίες μόνο μία, ο μικροπυρήνας, εκτελείται σε κατάσταση πυρήνα ενώ οι υπόλοιπες εκτελούνται σε σχετικά λιγότερο ισχυρές κοινές διεργασίες χρήστη.

Σε ένα μονολιθικό λειτουργικό συτημα διακρίνονται δύο ξεχωριστά επίπεδα εκτέλεσης κώδικα. Κάθε επίπεδο έχει διαοφορετικούς χώρους μνήμης και διαφορετικά δικαιώματα πρόσβασης στο υλικό. Τα δύο αυτά επίπεδα είναι ο χώρος χρήστη (userspace), στο οποίο εκτελούνται οι διεργασίες του χρήστη και ο χώρος πυρήνα (kernelspace), στο οποίο εκτελείται ο κώδικας του πυρήνα. Προκειμένου μία διεργασία να εκτελέσει μία λειτουργία για την οποία δε διαθέτει τα ανάλογα δικαιώματα, υπάρχει ένας μηχανισμός μέσω του οποίου η διεργασία ζητά από τον πυρήνα να εκτλέσει τη συγκεκριμένη λειτουργία για λογαριασμό της. Ο μηχανισμός αυτός ονομάζεται κλήση συτήματος (system call).

Ένα από τα πρώτα και πιο ιστορικά λειτουργικά συστήματα είναι το UNIX. Το UNIX επιλέχτηκε ως η βάση για μία τυποποιημένη διεπαφή συστήματος, ωστόσο υπήρχαν και δημιουργήθηκαν πολλές παραλλαγές του. Συνεπώς, δημιουργήθηκε η ανάγκη για τη δημιουργία ενός κοινού Standard, ώστε να διατηρείται η συμβατότητα μεταξύ

αυτών των διάφορων εκδόσεων του UNIX. Το standard που δημιουργήθηκε ήταν το POSIX, το οποίο ορίζει μία ελάχιστη διασύνδεση κλήσεων συστήματος που πρέπει να υποστηρίζουν τα συμβατά συστήματα UNIX. Μάλιστα το POSIX χρησιμοποιείται ακόμα και στις μέρες μας, ενώ έχουν δημιουργηθεί και κάποιες παραλλαγές του. Το UNIX επηρέασε σε μεγάλο βαθμό και κάποια από τα πιο δημοφικη σύγχρονα λειοτυργικά συστήματα όπως το Linux και αυτά της BSD οικονγένειας. Ένα από αυτά τα λειτουργικά συστήματα είναι και το NetBSD, το οποίο είναι ένα γενικού σκοπού λειτουργικό σύστημα. Χρησιμοποιείται συνήθως σε servers και λιγότερο σε προσωπικούς υπολογιστές, ενώ υποστηρίζει αρκετές αριτεκτονικές.

2.4 Μεταγλώττιση

Κεφάλαιο 3

Unikernel frameworks

Σε αυτό το κεφάλαιο θα γίνει μία εισαγωγή στην έννοια των unikernels και το σκοπό τους, ενώ θα παρουσιαστούν ορισμένα unikernel frameworks, τα οποία μελετήθηκαν κατά τη διάρκεια αυτής της εργασίας.

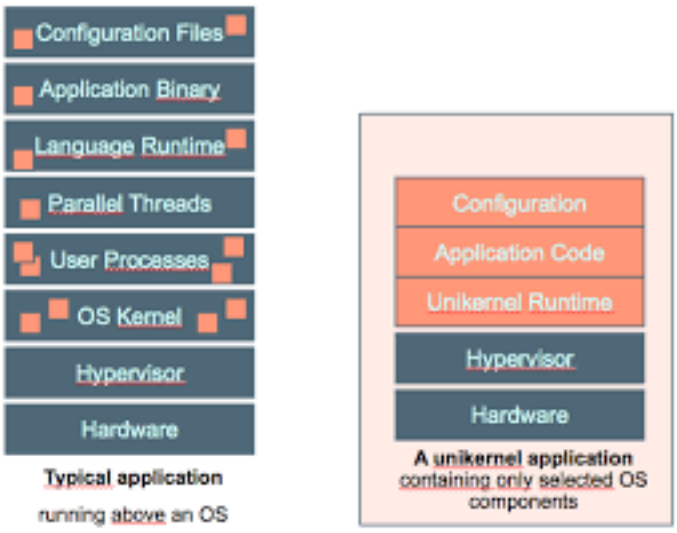
Ένας τυπικός ορισμός των unikernels αναφέρεται στη σελίδα unikernel.org και είναι ο εξής:

Τα Unikernels είναι εξειδικευμένες εικόνες μηχανής, με ένα μοναδικό χώρο διευθύνσεων, τα οποία κατασκευάζονται χρησιμοποιώντας library operating systems

Αναλύοντας τον παραπάνω ορισμό μπορούμε εύκολα να συμπεράνουμε τα βασικά χαρακτηριστικά των unikernels:

- εξειδικευμένες εικονικές μηχανές: κάθε unikernel μπορεί να είναι διαφορετικός από τους υπόλοιπους. Κάθε ένας αφορά μία συγκεκριμένη εφαρμογή και φτιάχνεται γύρω από αυτήν.
- ένας μοναδικός χώρος διευθύνσεων: στην περίπτωση των unikernels δεν υπάρχει ο διαχωρισμός μεταξύ userspace και kernelspace, όπως στα συμβατικά λειτουργικά συστήματα.
- library operating systems [13]: πρόκειται για λειτουργικά συστήματα τα οποία παρέχουν τις υπηρεσίες που υποστηρίζουν (networking κ.λ.π.) στη μορφή βιβλιοθηκών, οι οποίες στη συνέχεια μπορούν να χρησιμοποιηθούν (link) από τις εφαρμογές.

Με λίγα λόγια ένας unikernel αποτελείται από τον κώδικα της εφαρμογής την οποία θα εκτελέσει, τα απαραίτητα κομμάτια του λειτουργικού συστήματος που η συγκεκριμένη εφαρμογή χρειάζεται και τις κατάλληλες παραμετροποιήσεις. Τα unikernels δεδομένου ότι αφορούν μία και μόνο εφαρμογή δεν υποστηρίζουν περισσότερες από μία διεργασίες, ούτε περισσότερους από έναν χρήστες. Αυτόματα όλο το κομμάτι των συμβατικών λειτουργικών συστημάτων που αφορούν τη



Σχήμα 3.1: Unikerne vs OS software stack

διαχείρηση και υποστήριξη πολλών διεργασίων και χρηστών καθίσταται άχρηστο και δεν συμπεριλαμβάνεται. Τα πράγματα είναι διαφορετικά όσον αφορά τα νήματα, καθώς άλλα unikernel frameworks τα υποστηρίζουν ενώ άλλα όχι.

Στην εικόνα 3.1 φαίνεται η διαφορά μεταξύ ενός συμβατικού λειτουργικού συστήματος και ενός unikernel. Τα παραπάνω ενοποιούνται σε μία εικόνα μηχανής που μπορεί να εκτελεστεί από έναν επόπτη. Γίνεται εύκολα αντιληπτό ότι το μέγεθος της τελικής εικόνας θα είναι πολύ μικρότερο από αυτή ενός συμβατικού λειτουργικού συστήματος. Μία ακόμη συνέπεια αυτού είναι ότι οι χρόνοι εκκίνησης θα είναι αρκετά μικρότεροι, αφού πλέον δε χρειάζεται να φορτωθούν και να εκκινήσουν όλες οι υπηρεσίες που προσφέρει ένα λειτουργικό σύστημα.

Από άποψη ασφάλειας, ο σχεδιασμός των unikernels από μόνος του προσφέρει σημαντική ασφάλεια. Αρχικά το μικρό μέγεθος των unikernels συνεπάγεται αυτόματα και μικρότερο εύρος επίθεσης από κάποιον κακόβουλο χρήστη. Επιπλέον πρόκειται για εξειδικευμένες εικόνες που αφορούν μία και μόνο εφαρμογή, οπότε στην περίπτωση που αποκτηθεί πρόσβαση στην εικονική μηχανή, το περιθώριο εκμετάλλευσης θα είναι αρκετά μικρό. Τέλος, υπάρχει απομόνωση μεταξύ των εικονικών μηχανών η οποία προσφέρεται από τον επόπτη.

Η έλλειψη διαχωρισμού kernelspace και userspace, μειώνει το κόστος μετάβασης από τον ένα χώρο διευθύνσεων στον άλλο, κάνοντας έτσι τις κλήσεις συστήματος να λειτουργούν σαν απλές κλήσεις συναρτήσεων. Ως αποτέλεσμα, η ταχύτητα εκτέλεσης αυξάνει σημαντικά, καθώς δε χρειάζονται πλέον οι μεταβάσεις από τον ένα χώρο διευθύνσεων στον άλλο, μία αρκετά δαπανηρή λειτουργία.

Η έννοια των library operating systems δεν είναι καινούρια, ωστόσο δεν μπορούσαν να χρησιμοποιηθούν λόγω του μεγάλου πλήθους των συσκευών και συνεπώς των drivers που θα έπρεπε να υποστηρίζουν. Στις μέρες μας όμως, οι επόπτες έχουν περιορίσει σημαντικά το πλήθος των συσκευών κάνοντας εφικτή τη χρήση των συγκεκριμένων λειτουργικών συστημάτων.

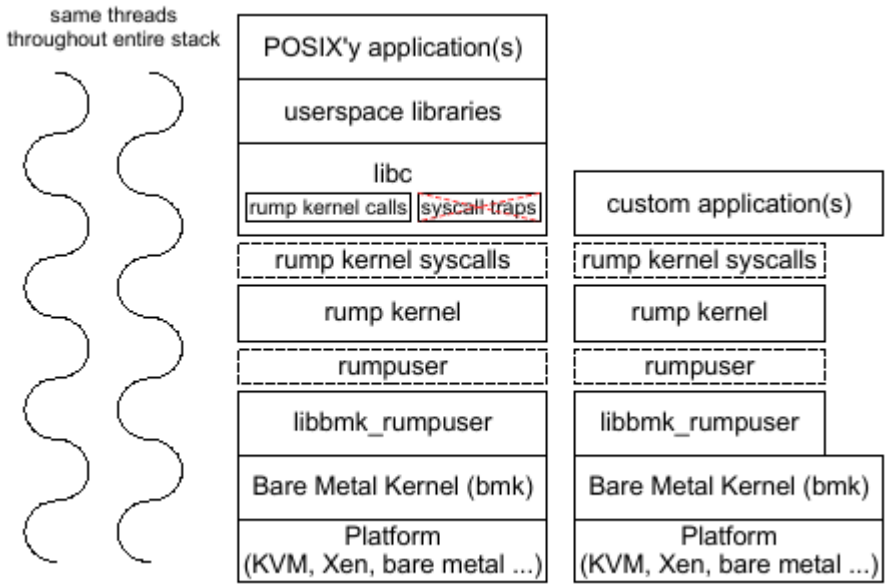
Αφ' ετέρου η απομάκρυνση όλων αυτών των υπηρεσιών από το λειτουργικό σύστημα, δημιουργεί αρκετούς περιορισμούς. Οι περισσότερες εφαρμογές έχουν φτιαχτεί για συμβατικά λειτυργικά συστήματα λαμβάνοντας υπόψιν και χρησιμοποιοώντας τις συγκεκριμένες υπηρεσίες. Απαιτείται, λοιπόν, να γίνουν σημαντικές αλλαγές στις εφαρμογές για να μπορέσουν να εκτελεστούν σε unikernels. Επιπλέον γίνεται πιο δύσκολη η αποσφαλμάτωση των unikernels δεδομένου των λίγοστών υπηρεσιών που προσφέρουν.

3.1 Rumprun

Το rumprun είναι ένα unikernel framework, το οποίο έχει χτιστεί με βάση τα συστατικά των drivers που προσφέρουν οι rump kernels. Το rumprun μπορεί να προσφέρει ένα POSIX-like περιβάλλον, το οποίο μπορεί να χρησιμοποιηθεί από POSIX εφαρμογές ώστε να μετατραπούν χωρίς καμία αλλαγή σε unikernel εικόνες. Αυτός ήταν και ένας από τους βασικούς στόχους του εγχειρήματος, η δυνατότητα δηλαδή σε POSIX κώδικα να μπορεί να εκτελεστεί χωρίς να υποστεί καμία αλλαγή. Στην εικόνα 3.2 φαίνεται η δομή του rumprun για POSIX εφαρμογές (αριστερά) και προσαρμοσμένες εφαρμογές (δεξιά). Προφανώς το δεξί μοντέλο φαίνεται αρκετά μικρότερο, αλλά είναι απαραίτητες να γίνουν συγκεκριμένες αλλαγές στην εφαρμογή. Ο συγκεκριμένος unikernel μπορεί να εκτελεστεί χρησιμοποιώντας Xen ή KVM, ενώ υποστηρίζει arm και x86 αρχιτεκτονικές. Ο κώδικας του rumprun είναι διαθέσιμος στο `http://repo.rumpkernel.org/rumprun`.

Όπως κάθε unikernel framework το rumprun υποστηρίζει μία και μόνο διεργασία. Μολαταύτα υποστηρίζει POSIX threads, δίνοντας τη δυνατότητα για την ύπαρξη περισσοτέρων από ένα νήματα. Μάλιστα ο χρονοδρομολογητής που διαθέτει είναι cooperative, οπότε αν ένα νήμα αποκτήσει τον έλεγχο δνε πρόκειται να διακοπεί από το χρονδορομοληγτή αλλά θα αφήσει το έλεγχο όταν εκείνο τερματίσει ή πρόκειται να μπλοκάρει. Κάποιοι ακόμα περιορισμοί, είναι η μη υποστήριξη εικονικής μνήμης (virtual memory) και σημάτων (signals), Συνεπώς εφαρμογές που χρησιμοποιούν σήματα ή κλήσεις συστήματος όπως η mmap() δε θα λειτουργούν χωρίς αλλαγές στον κώδικα τους.

Ένας rumprun unikernel αποτελείται από τα απαραίτητα κομ-



Σχήμα 3.2: Rumprun software stack

μάτια των rump kernels που χρειάζεται η εφαρμογή και την ίδια την εφαρμογή. Για τη δημιουργία του unikernel γίνεται πάντα cross-compilation, οπότε δε χρειάζεται κάποιος ήδη ΄ετοιμος rumprun unikernels, αλλά μία σειρά από εργαλεία. Η διαδικασία έχει ως εξής:

1. Αρχικά γίνεται η μεταγλώττιση του κώδικα της εφαρμογής χρησιμοποιώντας έναν cross-compiler.

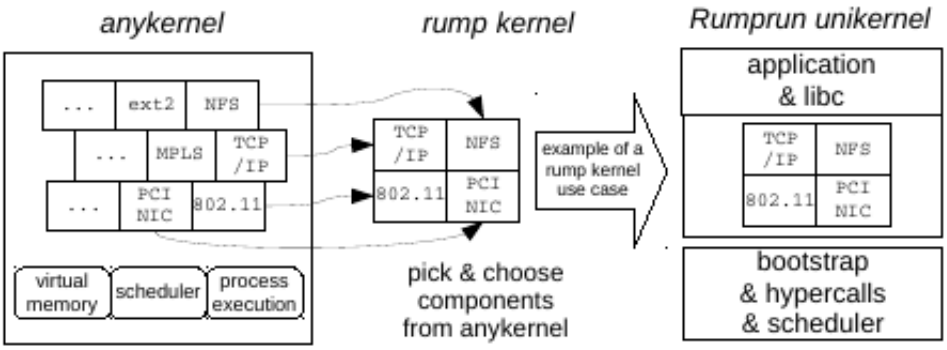
2. Ύστερα, αντί για linking γίνεται pseudo-linking όπως ονομάζουν τη διαδικασία κατά την οποία απλά ελέγχονται αν ικανοποιούνται όλες οι εξαρτήσεις συμβόλων, χωρίς να γίνεται κάποια σύνδεση με κάποιο συστατικό του λειτουργικού συστήματος.

3. Τέλος, το παράγωγο του προηγούμενου βήματος γίνεται ”bake”, όπου εισάγονται και τα κομμάτια του λειτουργικού συστήματος που απαιτούνται, όπως έχουν οριστεί κατά την εκτέλεση της εντολής.

Όπως έχει αναφερθεί ο rumprun unikernel βασίζεται στα rump kernels [8]. Τα rump kernels παρέχουν drivers του NetBSD ως φορητά εξαρτήματα με τα οποία μπορείς να εκτελέσεις εφαρμογές χωρίς να ναι απαραίτητη η ύπαρξη λειτουργικού συστήματος [9]. Η αρχική ιδέα ήταν να υπάρχει η δυνατότητα να εκτελούνται αμετάβλητοι drivers του NetBSD ως απλά προγράμματα σε userspace, ώστε να είναι πιο εύκολος ο έλεγχος και η ανάπτυξη NetBSD drivers. Ένας

rump kernel είναι ένας πυρήνας χρονομερισμού (timesharing) από τον οποίο έχουν αφαιρεθεί ορισμένα κομμάτια [7]. Τα κομμάτια που κρατήθηκαν είναι οι drivers και οι απαραίτητες ρουτίνες που χρειάζονται οι συγκεκριμένοι drivers για να λειτουργήσουν (συχχρονισμός, κατανομή μνήμης κ.λ.π.). Τα κομμάτια που αφαιρέθηκαν είναι αυτά που αφορούν τις διεργασίες, την εικονική μνήμη κ.α.. Επιπλέον τα rump kernels έχουν και ένα καλώς ορισμένο στρώμα φορητότητας, ώστε να είναι εύκολο να ενσωματωθούν σε διάφορα περιβάλλοντα.

Πίσω από τα rump kernels υπάρχει μία ακόμα έννοια αυτή του anykernel [7]. Ο συγκεκριμένος όρος δημιουργήθηκε ως απάντηση στην όλο και αυξανόμενο αριθμό μοντέλων λειτουργικών συστημάτων (monolithic, exokernel, mikrokernel, unikernel κ.λ.π.). Ένα πολύ μεγάλο ποσοστό ενός λειτουργικού συστήματος ανεξάρτητα από το μοντέλο που χρησιμοποιεί αποτελείται από drivers. Ο όρος anykernel περιγράφει ένα κώδικα βάσης (codebase) τύπου κώδικα πυρήνα από τον οποίο οι οδηγοί μπορούν να εξαχθούν και να ενσωματωθούν σε οποιοδήποτε μοντέλο λειτουργικού συστήματος, χωρίς να χρειάζονται αλλαγές και συντήρηση. Στην εικόνα 3.3 φαίνεται η σχέση μεταξύ των εννοιών anykernel, rump kernels και rumprun.



Σχήμα 3.3: Σχέση μεταξύ των ενοιών anykernel, rump kernel και rumprun

3.2 OSv

dsfdsgfdsgdfs

3.3 Click OS

dsfdsgfdsgdfs

3.4 Lkl

dsfdsgfdsgdfs

3.5 Include OS

dsfdsgfdsgdfs

3.6 Mirage OS

dsfdsgfdsgdfs

Κεφάλαιο 4

Σχεδιασμός και υλοποίηση

μπλα μπλα μπλα

4.1 Mpla

dsfdsgfdsgdfs

4.2 Mpla

dsfdsgfdsgdfs

4.2.1 Mpla

dsfdsgfdsgdfs

Κεφάλαιο 5

Επίλογος

μπλα μπλα μπλα

5.1 Mpla

dsfdsgfdsgdfs

5.2 Mpla

dsfdsgfdsgdfs

5.2.1 Mpla

dsfdsgfdsgdfs

mpla [5]

Βιβλιογραφία

[1] Keith Adams and Ole Agesen. A comparison of software and hardware techniques for x86 virtualization. *ACM SIGARCH Computer Architecture News*, 34(5):2–13, 2006.

[2] Ole Agesen, Jim Mattson, Radu Rugina, and Jeffrey Sheldon. Software techniques for avoiding hardware virtualization exits. In *USENIX Annual Technical Conference*, pages 373–385, 2012.

[3] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, volume 41, page 46, 2005.

[4] Michael Eder. Hypervisor-vs. container-based virtualization. *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, 1, 2016.

[5] Xing Gao, Zhongshu Gu, Mehmet Kayaalp, Dimitrios Pendarakis, and Haining Wang. Containerleaks: emerging security threats of information leakages in container clouds. In *Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on*, pages 237–248. IEEE, 2017.

[6] Major Hayden and Richard Carbone. Securing linux containers. *GIAC (GCUX) Gold Certification, Creative Commons Attribution-ShareAlike 4.0 International License*, 19, 2015.

[7] Antti Kantee. *The Design and Implementation of the Anykernel and Rump Kernels*. PhD thesis, doctoral dissertation. Department of Computer Science and Engineering, Aalto University, 2012.

[8] Antti Kantee. On rump kernels and the rumprun unikernel, 2015.

[9] Antti Kantee and Justin Cormack. Rump kernels no os? no problem! *USENIX; login: magazine*, 2014.

[10] A Cameron Macdonell et al. *Shared-memory optimizations for virtual machines*. University of Alberta Edmonton, Canada, 2011.

[11] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[12] Gerald J Popek and Robert P Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974.

[13] Donald E Porter, Silas Boyd-Wickizer, Jon Howell, Reuben Olinsky, and Galen C Hunt. Rethinking the library os from the top down. In *ACM SIGPLAN Notices*, volume 46, pages 291–304. ACM, 2011.

[14] Yi Ren, Ling Liu, Qi Zhang, Qingbo Wu, Jianbo Guan, Jinzhu Kong, Huadong Dai, and Lisong Shao. Shared-memory optimizations for inter-virtual-machine communication. *ACM Computing Surveys (CSUR)*, 48(4):49, 2016.