ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

V4Vsockets: Μηχανισμός αποδοτικής ενδο-επικοινωνίας εικονικών μηχανών χαμηλής επιβάρυνσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννα - Μαρία Αλιφιεράκη

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2014

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

V4Vsocketss: Μηχανισμός αποδοτικής ενδο-επικοινωνίας εικονικών μηχανών χαμηλής επιβάρυνσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννα - Μαρία Αλιφιεράκη

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 24η Ιουλίου 2014.

.....	.....	.....
Ν. Κοζύρης Καθηγητής ΕΜΠ	Δ. Σούντρης Επ. Καθηγητής ΕΜΠ	Γ. Γκούμας Λέκτορας ΕΜΠ

Αθήνα, Ιούλιος 2014.

.....

Ιωάννα - Μαρία Αλιφιεράκη
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Στις μέρες μας, οι υποδομές Cloud Computing προσφέρουν μεγάλη ευελιξία και απομόνωση για την εκτέλεση ενός μεγάλου πλήθους εφαρμογών και υπηρεσιών. Οι υποδομές αυτές αποτελούνται κατά κανόνα από συστοιχίες υπολογιστών (clusters), με πολυπύρηνους επεξεργαστές, στις οποίες αναπτύσσεται ένα περιβάλλον εικονικών μηχανών, το οποίο αναλαμβάνει την ασφάλεια, απομόνωση και τον κατάλληλο διαμοιρασμό των πόρων στις διάφορες υπηρεσίες. Οι συστοιχίες αυτές προσφέρουν πολύ μεγάλη επεξεργαστική ισχύ, γεγονός που τις καθιστά ιδανικές για πληθώρα εφαρμογών που απαιτούν υπολογιστική ισχύ (High Performance Computing applications). Ωστόσο ένα από τα σοβαρότερα προβλήματα εκτέλεσης εφαρμογών υψηλής επίδοσης σε εικονικά περιβάλλοντα είναι το κόστος της επικοινωνίας. Έχουν γίνει πολλές προσπάθειες ώστε να ενισχυθεί η απομόνωση μεταξύ των εικονικών μηχχανών που φιλοξενούνται στο ίδιο φυσικό μηχάνημα (co-resident VMs). Η απομόνωση είναι πολύ σημαντική παράμετρος από την σκοπιά της ασφάλειας αλλά πολλές φορές επιτυγχάνεται σε βάρος της απόδοσης. Η απομόνωση είναι πολύ σημαντική παράμετρος από την σκοπιά της ασφάλειας αλλά πολλές φορές επιτυγχάνεται σε βάρος της απόδοσης. Στην περίπτωση όπου εικονικές μηχανές που φιλοξενούνται στο ίδιο φυσικό μηχάνημα χρειάζεται να επικοινωνήσουν, η ισχυρή απομόνωση προκαλεί επιβραδύνει την επικοινωνία. Οι Cloud providers δε δίνουν τη δυνατότητα στους χρήστες να επιλέγουν τη φυσική τοποθεσία εκτέλεσης των εικονικών μηχανών τους, με αποτέλεσμα κάποιες από αυτές να συνυπάρχουν στο ίδιο φυσικό μηχάνημα. Έτσι, αν δεν υπάρχει πρόβλεψη για αυτήν την περίπτωση, η επικοινωνία μεταξύ εικονικών μηχανών που βρίσκονται στο ίδιο μηχάνημα δε γίνεται βέλτιστα. Στόχος της παρούσας εργασίας είναι η μελέτη και αποτίμηση μεθόδων επικοινωνίας εικονικών μηχανών που συνυπάρχουν στο ίδιο φυσικό μηχάνημα χωρίς την αλλαγή της προγραμματιστικής διεπαφής. Συγκεκριμένα μελετάται ένας νέος μηχανισμός επικοινωνίας εικονικών μηχανημάτων (virtual machines – VMs) που βρίσκονται στο ίδιο φυσικό μηχάνημα. Ο μηχανισμός λέγεται V4V (virtual for virtual) και το εικονικό περιβάλλον δημιουργείται από τον ελεγκτή εικονικών μηχανών Xen. Χτίζουμε πάνω σε αυτόν τον μηχανισμό και δημιουργούμε το V4Vsockets ένα πλαίσιο το οποίο είναι συμβατό με την προγραμματιστική διεπαφή Socket αλλά παρακάμπτει τα επίπεδα του πρωτοκόλλου TCP/IP. Μετράμε την επίδοση τόσο του V4Vsockets όσο και την επίδοση του κλασσικού τρόπου επικοινωνίας που προσφέρει το Xen και τα αποτελέσματα δείχνουν ταχύτερη μεταφορά δεδομένων για το V4Vsockets.

Λέξεις-Κλειδιά : v4vsockets, ενδοεπικοινωνία εικονικών μηχανών, εικονοποίηση, Xen

Abstract

Nowadays, with the advent of virtualization techniques, Cloud Computing infrastructures are becoming a great trend, providing flexibility, dedicated execution and isolation to a vast number of services. These infrastructures, built on clusters of multicores, offer huge processing power, ideal for mass deployment of compute-intensive applications. However, bridging the gap between I/O techniques in virtualized environments and application demands seems to be a major challenge. Therefore, lowering the cost of communication in virtualized environments is an intriguing task, especially in a high-performance computing context. Great effort has been put on strengthening the isolation barrier between co-existing VMs. Isolation is an important requirement from a security point of view; in some cases, however, it is achieved at the expense of efficiency. When applications hosted on co-resident VMs need to communicate, this isolation barrier implies communication overhead. Cloud providers do not give users the opportunity to choose the physical location of their virtual machines, and therefore some VMs may coexist on the same physical machine. Thus, if this case is not taken into account, a communication intensive application which requires data transfer between two or more co-resident VMs will present suboptimal performance. The purpose of this thesis is to study, evaluate and compare communication methods between VMs hosted on the same physical machine. A major requirement is that the communication API remains intact. Specifically, we examine a novel mechanism using the Xen hypervisor, called Virtual-to-Virtual^2 (V4V), which provides inter-domain communication, and serves as the transport layer of our approach. We use TCP/IP protocol semantics and build on this mechanism, creating V4Vsockets, a socket-compliant interface to V4V that supports socket operations bypassing all transport layers of the underlying stack (IP layers, software bridges etc.). We evaluate the performance of V4Vsockets compared to conventional TCP/UDP sockets and our experimental results seem extremely promising. V4Vsockets show improved performance when transferring both small and large messages.

Keywords : v4vsockets, inter-domain communication, virtualization, Xen

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στο Εργαστήριο Υπολογιστικών Συστημάτων της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου, υπό την επίβλεψη του Καθηγητή Νεκτάριου Κοζύρη.

Καταρχάς, θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Νεκτάριο Κοζύρη, τόσο για την εποπτεία του κατά την εκπόνηση της εργασίας μου, όσο και για τις γνώσεις και την έμπνευση που μου προσέφερε με τη διδασκαλία του, καθ' όλη τη διάρκεια της φοίτησής μου.

Οι θερμότερες ευχαριστίες απευθύνονται στο Μεταδιδακτορικό Ερευνητή Αναστάσιο Νάνο για τη συνεχή του καθοδήγηση στη διάρκεια της εκπόνησης της διπλωματικής αυτής εργασίας, καθώς και για την ενθάρρυνση και αισιοδοξία που μου παρείχε. Η βοήθειά του ήταν ανεκτίμητη τόσο για τις καίριες παρεμβάσεις σε επίπεδο σχεδιασμού και υλοποίησης όσο και για τις τεχνικές γνώσεις που μου μετέδωσε. Χωρίς τη συμβολή του η ολοκλήρωση της διπλωματικής εργασίας δεν θα ήταν εφικτή.

Επίσης, επιβάλλεται να ευχαριστήσω τους φίλους και συμφοιτητές για την παρουσία τους και την πολύτιμη βοήθειά τους, σε προσωπικό και επιστημονικό επίπεδο. Χωρίς αυτούς, τα φοιτητικά χρόνια που πέρασαν θα ήταν λιγότερο όμορφα και πολύ πιο δύσκολα.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου και τις αδερφές μου, για την αγάπη τους, την αμέριστη υποστήριξή τους και την εμπιστοσύνη τους σε κάθε μου επιλογή, από τα πρώτα χρόνια της ζωής μου μέχρι σήμερα, αδιάκοπα και ακούραστα.

Ιωάννα - Μαρία Αλιφιεράκη

Περιεχόμενα

Κατάλογος σχημάτων

Κεφάλαιο 1

Εισαγωγή

Στις μέρες μας, οι υποδομές Cloud Computing προσφέρουν μεγάλη ευελιξία και απομόνωση για την εκτέλεση ενός μεγάλου πλήθους εφαρμογών και υπηρεσιών. Οι υποδομές αυτές αποτελούνται κατά κανόνα από συστοιχίες υπολογιστών (clusters), με πολυπύρηνους επεξεργαστές, στις οποίες αναπτύσσεται ένα περιβάλλον εικονικών μηχανών, το οποίο αναλαμβάνει την ασφάλεια, απομόνωση και τον κατάλληλο διαμοιρασμό των πόρων στις διάφορες υπηρεσίες. Οι συστοιχίες αυτές προσφέρουν πολύ μεγάλη επεξεργαστική ισχύ, γεγονός που τις καθιστά ιδανικές για πληθώρα εφαρμογών που απαιτούν υπολογιστική ισχύ (High Performance Computing applications). Ωστόσο ένα από τα σοβαρότερα προβλήματα εκτέλεσης εφαρμογών υψηλής επίδοσης σε εικονικά περιβάλλοντα είναι το κόστος της επικοινωνίας.

Σύμφωνα με τους Popek και Goldberg, “μία εικονική μηχανή είναι ένα αποδοτικό απομονωμένο αντίγραφο της πραγματικής μηχανής” [10]. Έχουν γίνει πολλές προσπάθειες ώστε να ενισχυθεί η απομόνωση μεταξύ των εικονικών μηχανών που φιλοξενούνται στο ίδιο φυσικό μηχάνημα (co-resident VMs). Η απομόνωση είναι πολύ σημαντική παράμετρος από την σκοπιά της ασφάλειας αλλά πολλές φορές επιτυγχάνεται σε βάρος της απόδοσης. Στην περίπτωση όπου εικονικές μηχανές που φιλοξενούνται στο ίδιο φυσικό μηχάνημα χρειάζεται να επικοινωνήσουν, η ισχυρή απομόνωση προκαλεί επιβραδύνει την επικοινωνία. Παραδείγματος χάρη, μία web υπηρεσία που τρέχει σε κάποιο εικονικό μηχάνημα ενδεχομένως να χρειάζεται να επικοινωνήσει με την βάση δεδομένων που τρέχει σε κάποιο άλλο εικονικό μηχάνημα προκειμένου να εξυπηρετηθούν οι συνδιαλλαγές των πελατών. Επιπλέον, οι διεργασίες των κατανεμημένων εφαρμογών υψηλής επίδοσης μπορεί να τρέχουν σε διαφορετικά εικονικά μηχανήματα και να απαιτείται η μεταξύ τους επικοινωνία. Οι Cloud providers δε δίνουν τη δυνατότητα στους χρήστες να επιλέγουν τη φυσική τοποθεσία εκτέλεσης των εικονικών μηχανών τους, με αποτέλεσμα κάποιες από αυτές να συνυπάρχουν στο ίδιο φυσικό μηχά-

νημα. Έτσι, αν δεν υπάρχει πρόβλεψη για αυτήν την περίπτωση, η επικοινωνία μεταξύ εικονικών μηχανών που βρίσκονται στο ίδιο μηχάνημα δε γίνεται βέλτιστα.

Έχουν γίνει αρκετές μελέτες που στοχεύουν στην στην βελτιστοποίηση της επικοινωνίας μεταξύ εικονικών μηχανών που φιλοξενούνται στο ίδιο φυσικό μηχάνημα και έχουν προταθεί διάφορες τεχνικές όπως ο διαμοιρασμός μνήμης, η παράκαμψη των επιπέδων εικονοποίησης καθώς και η βελτιστοποίηση των δικτυακών διεπαφών. Η παρούσα εργασία εξετάζει το θέμα της ενδοεπικοινωνίας των εικονικών μηχανών μέσω ανταλλαγής μηνυμάτων, δίνοντας έμφαση στην διατήρηση της προγραμματιστικής διεπαφής.

1.1 Σκοπός

Στόχος της παρούσας εργασίας είναι η μελέτη και αποτίμηση μεθόδων επικοινωνίας εικονικών μηχανών που συνυπάρχουν στο ίδιο φυσικό μηχάνημα χωρίς την αλλαγή της προγραμματιστικής διεπαφής. Ιδιαίτερη έμφαση δίνεται στη διατήρηση της διεπαφής επικοινωνίας (Sockets) για μεγαλύτερη ευκολία στην υιοθέτηση συμβατικών εφαρμογών.

Συγκεκριμένα μελετάται ένας μηχανισμός επικοινωνίας εικονικών μηχανημάτων (virtual machines – VMs) που βρίσκονται στο ίδιο φυσικό μηχάνημα. Ο μηχανισμός λέγεται V4V (virtual for virtual) και το εικονικό περιβάλλον δημιουργείται από τον ελεγκτή εικονικών μηχανών Xen. Χτίζουμε πάνω σε αυτόν τον μηχανισμό και δημιουργούμε το V4Vsockets μία βιβλιοθήκη χώρου χρήστη η οποία είναι συμβατή με την προγραμματιστική διεπαφή Socket αλλά παρακάμπτει τα επίπεδα του πρωτοκόλλου TCP/IP.

Τέλος, εκτός από την μελέτη του μηχανισμού πραγματοποιείται μέτρηση της απόδοσής του και σύγκριση του με τον κλασσικό τρόπο επικοινωνίας μέσω των TCP/IP sockets. Η αποτίμηση της μεθόδου γίνεται με εκτέλεση μετρο-προγραμμάτων ανταλλαγής μηνυμάτων.

1.2 Οργάνωση Κειμένου

Στη συνέχεια παρουσιάζεται η μεθοδολογία αξιολόγησης της επίδοσης του V4Vsockets, η περιγραφή του σχεδιασμού και υλοποίησής του, γίνεται μια σύντομη αναφορά στην σχετική βιβλιογραφία και περιγράφεται το απαραίτητο θεωρητικό υπόβαθρο.

Πιο συγκεκριμένα στο Κεφάλαιο 2 καλύπτεται το απαραίτητο θεωρητικό υπόβαθρο, το οποίο περιλαμβάνει μία πολύ σύντομη εισαγωγή στο λειτουργικό σύστημα Linux και τους οδηγούς χαρακτήρων,

περιγράφεται το πρωτόκολλο TCP/IP και η προγραμματιστική διεπαφή Socket και τέλος καλύπτονται θέματα σχετικά την εικονοποίηση και το Xen.

Στο Κεφάλαιο 3 γίνεται μία σύντομη αναφορά στην σχετική βιβλιογραφία, δηλαδή αναφέρουμε μελέτες που έχουν γίνει στο παρελθόν και οι οποίες επιχειρούν να βελτιστοποιήσουν την επικοινωνία εικονικών μηχανών που βρίσκονται στο ίδιο φυσικό μηχάνημα.

Στο Κεφάλαιο 4 γίνεται αναλυτική περιγραφή του σχεδιασμού και της υλοποίησης του V4Vsockets και του V4V και δίδεται έμφαση στα σημεία όπου επηρεάζουν την επίδοση του μηχανισμού.

Στο Κεφάλαιο 5 γίνεται η αξιολόγηση της επίδοσης του V4Vsockets. Παρουσιάζονται διαγράμματα όπου φαίνεται η σύγκριση του V4Vsockets με συμβατικούς τρόπους επικοινωνίας, οι διάφοροι παράμετροι που επηρεάζουν την επίδοση του μηχανισμού καθώς επίσης η κλιμακωσιμότητά του.

Τέλος, στο Κεφάλαιο 6 αναφέρονται τα τελικά συμπεράσματα της παρούσας μελέτης όπως επίσης και πιθανές μελλοντικές επεκτάσεις αυτής της διπλωματικής εργασίας.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

2.1 Linux

Το Linux είναι ένα σύγχρονο λειτουργικό σύστημα ανοιχτού κώδικα, που βασίζεται στις βασικές σχεδιαστικές αρχές και την παράδοση του Unix. Αναπτύχθηκε αρχικά από τον Linus Torvalds για τον επεξεργαστή i386 της Intel αλλά έχει μεταφερθεί έως σήμερα σε πολλές διαφορετικές αρχιτεκτονικές (IA-64, MIPS, Alpha, SPARC, PowerPC κ.α.).

Το Linux προσφέρει την δυνατότητα ταυτόχρονης προστατευμένης εκτέλεσης πολλών διαφορετικών διεργασιών, που ανήκουν σε διαφορετικούς χρήστες, είναι δηλαδή multi-tasking και multi-user. Με τον όρο “προστατευμένη” εκτέλεση εννοείται ότι κάθε διεργασία εκτελείται σε απομόνωση από τις υπόλοιπες, σε δικό της εικονικό χώρο διευθύνσεων και χωρίς καμιά δυνατότητα άμεσης πρόσβασης στο υλικό του υπολογιστικού συστήματος. Έτσι, πιθανή δυσλειτουργία της δεν επηρεάζει την εξέλιξη των υπολοίπων που εκτελούνται στο σύστημα. Επιπλέον, κάθε διεργασία ανήκει σε συγκεκριμένο χρήστη, οπότε έχει και ανάλογα δικαιώματα σε πόρους του συστήματος.

Το λειτουργικό σύστημα διαχειρίζεται τους πόρους του συστήματος και συντονίζει την πρόσβαση των διεργασιών σε αυτούς. Το Linux είναι ένα μονολιθικό λειτουργικό σύστημα, όπως και τα περισσότερα συστήματα που βασίζονται στο Unix : οι περισσότερες λειτουργίες του υλοποιούνται σε ένα πρόγραμμα, που ονομάζεται *πυρήνας* του Λ.Σ, το οποίο εκτελείται σε ένα ενιαίο χώρο διευθύνσεων και έχει πλήρη ελεύθερη πρόσβαση στο υλικό (*προνομιούχος κατάσταση - privileged mode*). Ο πυρήνας του Linux αναλαμβάνει το σύνολο σχεδόν των λειτουργιών που χαρακτηρίζουν ένα σύγχρονο Λ.Σ.: Διαχείριση CPU, χρονοδρομολόγηση διεργασιών, διαχείριση μνήμης, διαχείριση συσκευών I/O, διαχείριση συστημάτων αρχείων, διαδικτύωση. Στον πυρήνα του Linux βρίσκονται επίσης και οι οδηγοί συσκευών (device drivers), που επιτρέπουν τον έλεγχο των διαφόρων

περιφερειακών συσκευών.

Σε ένα σύστημα που τρέχει Linux, υπάρχουν δύο εντελώς χωριστά *επίπεδα εκτέλεσης* κώδικα, που αναφέρονται τόσο σε διαφορετικούς χώρους μνήμης όσο και σε διαφορετικά δικαιώματα πρόσβασης στο υλικό: από τη μία είναι ο *χώρος χρήστη* (userspace), στον οποίο ζουν και εκτελούνται οι διεργασίες χρήστη και από την άλλη είναι ο *χώρος πυρήνα* (kernelspace), στον οποίο εκτελείται ο κώδικας του πυρήνα. Η διάκριση των δύο χώρων και των διαφορετικών δικαιωμάτων πρόσβασης επιβάλλεται με μηχανισμούς της CPU.

Προκειμένου μια διεργασία να μπορέσει να εκτελέσει μια λειτουργία που δεν έχει δικαίωμα να πραγματοποιήσει (πχ πρόσβαση σε κάποια συσκευή), υπάρχει ένας μηχανισμός μέσω του οποίου μια διεργασία ζητάει από τον πυρήνα να ολοκληρώσει κάποια λειτουργία εκ μέρους της. Αυτός είναι ο μηχανισμός των *κλήσεων συστήματος (systems calls)*, ο οποίος λειτουργεί ως εξής: κατά την εκτέλεση της, η διεργασία ετοιμάζει και υποβάλλει κατάλληλη αίτηση στο λειτουργικό σύστημα. Τότε, η CPU ξεκινά να εκτελεί κώδικα πυρήνα, ο οποίος ελέγχει την ορθότητα των δεδομένων εισόδου και - αν η διεργασία έχει τα ανάλογα δικαιώματα - πραγματοποιεί τη ζητούμενη λειτουργία και επιστρέφει τα αποτελέσματά της πίσω στον χώρο χρήστη. Ο έλεγχος επανέρχεται τότε στον κώδικα της διεργασίας.

Το Linux υποστηρίζει τη δυναμική εισαγωγή νέου κώδικα στο χώρο διευθύνσεων του πυρήνα, μέσω του μηχανισμού των *τμημάτων πυρήνα (kernel modules)*. Τα modules είναι τμήματα κώδικα τα οποία ενσωματώνονται με τον πυρήνα κατά τη λειτουργία του και επεκτείνουν τις δυνατότητές του.

2.1.1 Οδηγοί Συσκευών στο Linux

Ένα σύγχρονο λειτουργικό σύστημα χρειάζεται να υποστηρίζει πολλές διαφορετικές κατηγορίες περιφερειακών συσκευών (π.χ. κάρτες ήχου, κάρτες γραφικών, προσαρμογείς δικτύου, ελεγκτές μέσων αποθήκευσης) αλλά και πολλές διαφορετικές συσκευές σε κάθε κατηγορία. Κάθε μία από αυτές ελέγχεται πιθανά με διαφορετικές εντολές, οπότε χρειάζεται ακριβής γνώση του τρόπου προγραμματισμού της για τη σωστή λειτουργία της και ειδικός κώδικας για τον χειρισμό της.

Προκειμένου να μην απαιτούνται αλλαγές στον κώδικα των εφαρμογών των χρηστών για να υποστηρίζονται νέες περιφερειακές συσκευές, το λειτουργικό σύστημα *αποκρύπτει* τις λεπτομέρειες του ελέγχου συσκευών, πίσω από τον μηχανισμό των κλήσεων συστήματος. Ο κώδικας οργανώνεται σε επίπεδα, με τις εφαρμογές να να χρησιμοποιούν υψηλότερου επιπέδου *αφαιρέσεις (abstractions)*, όπως “αρχείο”, ή “TCP/IP socket” για να επικοινωνήσουν με τον έξω κόσμο.

Ωστόσο και ο ίδιος ο κώδικας του πυρήνα είναι σε μεγάλο βαθμό ανεξάρτητος από τη συσκευή που χρησιμοποιείται κάθε φορά, γιατί ο πυρήνας είναι οργανωμένος σε στρώματα: κάποια είναι ανεξάρτητα από το υλικό (π.χ. ο χρονοδρομολογητής, τα συστήματα αρχείων, η υποστήριξη TCP/IP), άλλα όχι.

Οι λεπτομέρειες του προγραμματισμού και ελέγχου μιας συσκευής περικλείονται μέσα σε τμήματα κώδικα που ονομάζεται *οδηγός συσκευής* και έχει συγκεκριμένο τρόπο (προγραμματιστική διεπαφή - interface) επικοινωνίας με τον υπόλοιπο πυρήνα. Έτσι μπορεί να αντικατασταθεί όταν υπάρξει ανάγκη, χωρίς να χρειάζονται αλλαγές στα υπόλοιπα τμήματα του πυρήνα.

Ωστόσο, δεν μπορούν όλοι οι οδηγοί συσκευών να αντιμετωπιστούν με τον ίδιο τρόπο από τον πυρήνα, γιατί δεν ταιριάζει η ίδια προγραμματιστική διεπαφή σε όλες τις συσκευές. Ο πυρήνας ομαδοποιεί τους οδηγούς συσκευών σε ένα μικρό αριθμό από κατηγορίες. Μέσα σε μια κατηγορία, όλοι οι οδηγοί υλοποιούν το ίδιο interface και αντιμετωπίζονται με όμοιο τρόπο. Έτσι, για να προστεθεί υποστήριξη για μια νέα συσκευή υλικού, χρειάζεται να επιλεγεί μία από τις υπάρχουσες κατηγορίες, στην οποία ταιριάζει καλύτερα η λειτουργία της και να γραφεί κατάλληλος οδηγός για το αντίστοιχο interface.

2.1.2 Κατηγορίες οδηγών στο Linux

Ο πυρήνας του Linux διακρίνει τρεις κύριες κατηγορίες οδηγών συσκευών. Για κάθε μία προβλέπεται ανάλογο interface, δηλαδή ένας συνδυασμός δεδομένων και λειτουργιών που χρειάζεται να υλοποιεί ο οδηγός.

Συσκευές χαρακτήρων: Στην κατηγορία αυτή ανήκουν συσκευές που αντιμετωπίζονται από τον πυρήνα ως ακολουθίες χαρακτήρων. Η προσπέλαση σε αυτές γίνεται με ένα interface παρόμοιο με την πρόσβαση αρχείων στο δίσκο: οι κυριότερες λειτουργίες που υλοποιούνται από τον οδηγό της συσκευής είναι η ανάγνωση δεδομένων από τη συσκευή, αντίστοιχη της read() από αρχείο, η εγγραφή δεδομένων στη συσκευή, αντίστοιχη της write(), και η μετακίνηση του σημείου ανάγνωσης/εγγραφής, αντίστοιχη της lseek(). Οι κυριότερες συσκευές υλικού που ταιριάζουν σε αυτό το μοντέλο είναι οι σειριακές και παράλληλες θύρες επικοινωνίας, ποντίκια, τερματικά χρηστών, κάρτες ήχου, ταινίες για αντίγραφα εφεδρείας.

Συσκευές μπλοκ: Πρόκειται για συσκευές στις οποίες η βασική μονάδα I/O είναι το μπολ δεδομένων, μια ομάδα σταθερού αριθμού bytes. Κάθε συσκευή μοντελοποιείται ως μια συλλογή από αριθμημένα μπλοκ, στα οποία μπορεί να γίνει προσπέλαση με τυχαία σειρά. Το interface με τον πυρήνα είναι εντελώς διαφορετικό από αυτό των συσκευών χαρακτήρα. Ο πυρήνας χρησιμοποιεί τον οδηγό για να πραγ-

ματοποιήσει δύο κύριες λειτουργίες: την ανάγνωση κάποιου μπλοκ με αριθμό n και η εγγραφή των δεδομένων του σε κάποια θέση μνήμης και η εγγραφή δεδομένων από κάποια θέση μνήμης στο μπλοκ με αριθμό n. Ως συσκευές μπλοκ μοντελοποιούνται συνήθως σκληροί δίσκοι ATA/SATA/ASCI και μονάδες CD/DVD-ROM/DVD-RW.

Συσκευές δικτύου: Ως συσκευές δικτύου μοντελοποιούνται περιφερειακά τα οποία μπορούν να στείλουν και να λάβουν πακέτα δεδομένων από και προς άλλα υπολογιστικά συστήματα Η κατηγορία αυτή περιλαμβάνει πραγματικές, φυσικές, κάρτες δικτύου(π.χ. eth0, eth1 για τον πρώτο και δεύτερο προσαρμογέα δικτύου Ethernet) αλλά και εικονικούς προσαρμογείς δικτύου, όπως τους lo και ppp0 που χρησιμοποιούνται για πακέτα που το μηχάνημα στέλνει στον εαυτό του και για πακέτα που ανταλλάσσονται πάνω από σειριακές γραμμές μέσω του πρωτοκόλλου PPP, αντίστοιχα.

2.1.3 Συσκευές Χαρακτήρων

Οι οδηγοί χαρακτήρων, όπως και οι συσκευές μπλοκ, είναι προσβάσιμες από τα προγράμματα χρήστη απευθείας, μέσω ειδικών αρχείων (“κόμβων”) στο σύστημα αρχείων. Όπως δηλαδή μια εφαρμογή μπορεί να ανοίξει ένα οποιοδήποτε αρχείο για εγγραφή ή ανάγνωση, οπότε τα δεδομένα ανταλλάσσονται με το σύστημα αρχείων, έτσι μπορεί να ανοίξει και μια σειρά από ειδικά αρχεία, ώστε να χρησιμοποιήσει μια συσκευή χαρακτήρων.

Τα ειδικά αυτά αρχεία δεν περιέχουν δεδομένα που αποθηκεύονται στο σύστημα αρχείων, αλλά αποτελούν πύλες πρόσβασης σε συσκευές του συστήματος μέσω των καθιερωμένων κλήσεων συστήματος open(), read(), write(), close() κτλ. Λόγω της ειδικής τους σημασίας βρίσκονται συνήθως αποθηκευμένα με κατάλληλα ονόματα κάτω από τον κατάλογο /dev.

Ωστόσο, αυτό που κάνει αυτά τα αρχεία ειδικά δεν είναι ούτε η θέση τους, ούτε το όνομά τους, αλλά δύο αριθμοί, οι οποίο προσδιορίζουν το αρχείο και καθορίζουν ποια συσκευή αφορά κάθε κόμβος. Οι αριθμοί αυτοί ονομάζονται major και minor number. Ο major number χρησιμοποιείται από τον πυρήνα για να προσδιορίσει τον οδηγό συσκευής που αφορά ο συγκεκριμένος κόμβος. Για παράδειγμα, όλοι οι δίσκοι SCSI αντιστοιχούν στον ίδιο major number. Ο minor number δεν αφορά τον κώδικα του πυρήνα αλλά τον ίδιο τον οδηγό της συσκευής και χρησιμοποιείται ώστε να μπορεί να διακρίνει ποια συγκεκριμένη συσκευή από όλες όσες μπορεί να χειριστεί αφορά το συγκεκριμένο αρχείο.

Προκειμένου ο οδηγός συσκευής να μπορεί να επικοινωνήσει με τον πυρήνα πρέπει να υλοποιεί μια συγκεκριμένη διεπαφή (interface) προς τον πυρήνα. Το interface αυτό περιγράφεται στην συνέχεια.

Όλοι οι οδηγοί παρέχουν ένα σύνολο από ρουτίνες. Κάθε οδηγός

παρέχει μία δομή struct file_operations που περιέχει δείκτες στις ρουτίνες αυτές. Κατά την αρχικοποίηση του οδηγού, αυτή η δομή "αγκιστρώνεται" σε ένα πίνακα που γνωρίζει ο πυρήνας. Κάθε φορά που γίνεται αίτηση για άνοιγμα ενός ειδικού αρχείου, ο αριθμός major του αρχείου χρησιμοποιείται από τον πυρήνα ως δείκτης σε αυτόν τον πίνακα, ώστε να εντοπίσει τον κατάλληλο οδηγό συσκευής και την ανάλογη δομή file_operations.

Εκτός από τη δομή file_operations, πολύ σημαντικό ρόλο στην αλληλεπίδραση του οδηγού συσκευής με τον πυρήνα παίζει και η δομή struct file. Η δομή αυτή αναπαριστά κάποιο ανοιχτό αρχείο μιας διεργασίας μέσα στον κώδικα του πυρήνα. Η δομή αυτή αντιστοιχίζεται, μέσω ενός δείκτη, με τη δομή file_operations που έχει δηλώσει ο οδηγός συσκευής στον πυρήνα. Έτσι, κάθε φορά που η διεργασία ζητά την εκτέλεση μιας κλήσης συστήματος επάνω στο ανοιχτό αρχείο, ο πυρήνας βρίσκει τη δομή file που αντιστοιχεί στο ανοιχτό αρχείο, ακολουθεί το σύνδεσμο προς τη δομή file_operations του οδηγού συσκευής, εν συνεχεία ακολουθεί το δείκτη της δομής που αναφέρεται στην συγκεκριμένη λειτουργία και φτάνει έτσι στην ρουτίνα του οδηγού συσκευής την οποία εκτελεί.

2.2 TCP/IP socket

2.2.1 Το πρωτόκολλο TCP/IP

Το *"TCP/IP"* είναι μία συλλογή πρωτοκόλλων επικοινωνίας στα οποία βασίζεται το Διαδίκτυο αλλά και μεγάλο ποσοστό των εμπορικών δικτύων. Η ονομασία TCP/IP προέρχεται από τις συντομογραφίες των δύο κυριότερων πρωτοκόλλων που περιέχει: το TCP ή Transmission Control Protocol(Πρωτόκολλο Ελέγχου Μετάδοσης) και το IP ή Internet Protocol(Πρωτόκολλο Διαδικτύου). Τα πρωτόκολλα αυτά αναπτύχθηκαν στα μέσα της δεκαετίας του 1970, όταν το Υπουργείο Άμυνας των ΗΠΑ ενδιαφέρθηκε να εγκαταστήσει ένα δίκτυο μεταγωγής πακέτων που θα καθιστούσε εύκολη την επικοινωνία μεταξύ διαφορετικού τύπου υπολογιστικών συστημάτων σε ιδρύματα έρευνας.

Το TCP/IP προσφέρει "από άκρο σε άκρο" επικοινωνία και καθορίζει ζητήματα σχετικά με την διαμόρφωση των δεδομένων, την διευθυνσιοδότηση, την μετάδοση, την δρομολόγηση και την άφιξη κάθε πακέτου στον προορισμό του. Το TCP/IP είναι οργανωμένο σε τέσσερα αφαιρετικά επίπεδα: επίπεδο εφαρμογής (application layer), επίπεδο μεταφοράς (transport layer), επίπεδο δικτύου (internet layer) και επίπεδο συνδέσμου (link layers). Ο κώδικας που υλοποιεί αυτά τα επίπεδα είναι γνωστός ως στοίβα πρωτοκόλλου (protocol stack). Τα επίπεδα αυτά είναι *διαφανή (transparent)*, το οποίο σημαίνει ότι

κάθε επίπεδο κρύβει τις λεπτομέρειες υλοποίησης και την πολυπλοκότητα του από τα κατώτερα επίπεδα. Δύο γειτονικά επίπεδα επικοινωνούν μεταξύ τους μέσω μιας συγκεκριμένης προγραμματιστικής διεπαφής(API). Παρακάτω περιγράφεται συνοπτικά ο ρόλος του κάθε επιπέδου στο TCP/IP.

- **Επίπεδο συνδέσμου**
  Το χαμηλότερο επίπεδο του TCP/IP πρωτοκόλλου είναι το επίπεδο συνδέσμου το οποίο αποτελείται από την διεπαφή υλικού (κάρτα δικτύου) και τον οδηγό συσκευής προς το φυσικό μέσο (πχ τηλεφωνική γραμμή, καλώδιο οπτικής ίνας). Ο ρόλος αυτού του επιπέδου είναι να μεταφέρει τα δεδομένα μέσω ενός φυσικού μέσου σε ένα δίκτυο.

  Προκειμένου να μεταφέρει δεδομένα, το επίπεδο συνδέσμου ενθυλακώνει τα δεδομενογράμματα (datagrams) από το επίπεδο δικτύου σε “πακέτα” που λέγονται πλαίσια (frames). Το επίπεδο συνδέσμου προσθέτει σε κάθε πλαίσιο μία επικεφαλίδα που περιέχει πληροφορίες σχετικά με την διεύθυνση προορισμού και το μέγεθος του πλαισίου. Σε ορισμένες περιπτώσεις αυτό το επίπεδο ενδέχεται να ανιχνεύει λάθη, να κάνει επαναμετάδοση όπως επίσης να διασπά τα πλαίσια σε μικρότερα πλαίσια και να τα συνενώνει όταν φτάσουν στον αποδέκτη.

- **Επίπεδο δικτύου**
  Ρόλος του επιπέδου δικτύου είναι να μεταφέρει δεδομένα από τον ξενιστή αποστολέα (source host) στον ξενιστή αποδέκτη (destination host). Το επίπεδο αυτό επιτελεί διάφορες εργασίες μεταξύ των οποίων κατακερματισμό δεδομένων (αν χρειάζεται) σε πακέτα όσο μικρά χρειάζεται για να μεταδοθούν μέσω του επιπέδου συνδέσμου, την δρομολόγηση των των πακέτων στο δίκτυο και την προσφορά υπηρεσιών στο επίπεδο μεταφοράς. Στο πρωτόκολλο TCP/IP, το κύριο πρωτόκολλο του επιπέδου δικτύου είναι το πρωτόκολλο IP.

  Το πρωτόκολλο IP μεταφέρει δεδομένα με τη μορφή δεδομενογραμμάτων - πακέτων (datagrams - packets). Κάθε πακέτο που αποστέλλεται μεταξύ δύο υπολογιστών δρομολογείται ανεξάρτητα μέσα στο δίκτυο και ενδέχεται να ακολουθήσει διαφορετική διαδρομή για να φτάσει στον προορισμό του. Κάθε IP πακέτο περιέχει μία επικεφαλίδα, η οποία περιέχει την διεύθυνση προορισμού, προκειμένου να είναι εφικτή η δρομολόγηση του πακέτου, καθώς επίσης και την διεύθυνση του αποστολέα, έτσι ώστε ο παραλήπτης να γνωρίζει την πηγή του πακέτου.

  Επίσης το πρωτόκολλο IP είναι ασυνδεσμικό (connectionless) πρωτόκολλο, αφού δεν προσφέρει την έννοια ενός εικονικού κυκλώματος που συνδέει δύο υπολογιστές. Είναι επίσης αναξιό-

πιστο (unreliable) καθώς κάνει την “καλύτερη προσπάθεια” για να μεταφέρει ένα πακέτο από την πηγή στον προορισμό, αλλά δεν εγγυάται ότι τα πακέτα θα φτάσουν με την σειρά με την οποία μεταδόθηκαν, ότι δεν θα υπάρχουν αντίγραφα ούτε αν θα φτάσουν τελικά στον προορισμό. Η αξιοπιστία προσφέρεται είτε χρησιμοποιώντας κάποιο αξιόπιστο πρωτόκολλο στο επίπεδο μεταφοράς (πχ TCP) είτε από την ίδια την εφαρμογή.

- **Επίπεδο μεταφοράς**
  Στο πρωτόκολλο TCP/IP χρησιμοποιούνται ευρέως δύο πρωτόκολλα στο επίπεδο μεταφοράς. Το *Πρωτόκολλο Δεδομενογράμματος Χρήστη (UDP - User Datagram Protocol)* το οποίο χρησιμοποιείται για τα datagram sockets και το *Πρωτόκολλο Ελέγχου Μετάδοσης (TCP - Transmission Control Protocol)*. Στη συνέχεια περιγράφουμε τα *Port Numbers* και πως συνδέονται εννοιολογικά με τα δύο αυτά πρωτόκολλα.

  Port Numbers
  Ο ρόλος του επιπέδου μεταφοράς είναι μα προσφέρει από άκρο σε άκρο επικοινωνία μεταξύ εφαρμογών που φιλοξενούνται σε διαφορετικούς υπολογιστές. Προκειμένου να προσφέρει αυτήν την υπηρεσία, το επίπεδο μεταφοράς πρέπει να μπορεί να διακρίνει τις διάφορες εφαρμογές που τρέχουν στον ίδιο υπολογιστή και οι οποίες επικοινωνούν με άλλες εφαρμογές στον ίδιο ή σε διαφορετικό υπολογιστή. Στο TCP και το UDP αυτό γίνεται μέσω ενός 16-bit αριθμού που λέγεται port number. Τα port numbers διακρίνονται σε δύο κατηγορίες : τα well-known, registered ή privileges ports και τα ephemeral ports. Στην πρώτη κατηγορία ανήκουν port numbers τα οποία είναι μόνιμα εκχωρημένες σε γνωστές εφαρμογές. Για παράδειγμα, η εφαρμογή ssh έχει port number 22 και η http 80. Τα ephemeral ports χρησιμοποιούνται στην περίπτωση που η εφαρμογή δεν επιλέγει κάποιο συγκεκριμένο port number και τότε το TCP και το UDP εκχωρούν στο socket της εφαρμογής κάποιο port number προκειμένου να μπορέσει το επίπεδο δικτύου να αναγνωρίσει τα άκρα της επικοινωνίας.

  User Datagram Protocol (UDP)
  Το πρωτόκολλο UDP προσθέτει δύο επιπλέον γνωρίσματα στο IP(επίπεδο δικτύου): το port number και το άθροισμα ελέγχου δεδομένων (data checksum) ώστε να προσφέρει δυνατότητα ελέγχου στα μεταδιδόμενα δεδομένα. Το UDP, όπως και το IP, είναι ασυνδεσμικό και αναξιόπιστο. Αν μια εφαρμογή που τρέχει πάνω από το UDP πρωτόκολλο απαιτεί αξιοπιστία, τότε είναι ευθύνη της εφαρμογής να εξασφαλίσει την αξιοπιστία.

  Transmission Control Protocol (TCP)
  Το πρωτόκολλο TCP προσφέρει αξιόπιστη, συνδεσμική, δικα-

τευθυντική (bidirectional), byte-stream επικοινωνία μεταξύ δύο άκρων (εφαρμογών). Προκειμένου να προσφέρει τα παραπάνω χαρακτηριστικά το TCP πρέπει να να επιτελεί τα παρακάτω έργα :

- *Εγκαθίδρυση Σύνδεσης*
  Προτού ξεκινήσει η επικοινωνία, το TCP εγκαθιδρύει ένα κανάλι επικοινωνίας μεταξύ των δύο άκρων επικοινωνίας. Κατά τη διάρκεια της εγκαθίδρυσης της σύνδεσης, ο αποστολέας και ο παραλήπτης μπορούν να ανταλλάξουν επιλογές προκειμένου να ορίσουν τις παραμέτρους για τη σύνδεση.
- *Πακετάρισμα δεδομένων σε τεμάχια*
  Τα δεδομένα κατακερματίζονται σε κομμάτια, το καθένα από τα οποία περιέχει ένα άθροισμα ελέγχου (checksum) ώστε να είναι εφικτός ο εντοπισμός των λαθών κατά τη μετάδοση. Κάθε κομμάτι δεδομένων μεταδίδεται σε ένα IP δεδομενόγραμμα (IP datagram).
- *Γνωστοποίηση, Επαναμετάδοση και timeouts*
  Όταν ένα TCP πακέτο φτάνει στον προορισμό του χωρίς λάθη, ο παραλήπτης στέλνει μία θετική γνωστοποίηση στον αποστολέα, ενημερώνοντας τον ότι έλαβε επιτυχώς τα δεδομένα. Αν το πακέτο φτάσει με λάθη, τότε απορρίπτεται και δεν αποστέλλεται γνωστοποίηση. Ο αποστολέας κάθε φορά που στέλνει ένα TCP πακέτο, ξεκινάει έναν μετρητή προκειμένου να ελέγξει αν τα πακέτα φτάνουν στον προορισμό τους. Αν δεν λάβει γνωστοποίηση μέχρι να λήξει ο μετρητής, τότε θεωρεί ότι το πακέτο δεν μεταδόθηκε σωστά και το στέλνει ξανά.
- *Διασφάλιση ακολουθίας πακέτων*
  Σε κάθε byte Που στέλνεται μέσω μιας TCP σύνδεσης εκχωρείται ένας αριθμός μιας λογικής ακολουθίας. Αυτός ο αριθμός υποδεικνύει την θέση του συγκεκριμένου byte μέσα στη ροή δεδομένων. Όταν μεταδίδεται ένα TCP πακέτο, συμπεριλαμβάνει ένα πεδίο το οποίο περιέχει τον ακολουθιακό αριθμό του πρώτου byte μέσα στο πακέτο.

  Η εκχώρηση ακολουθιακών αριθμών στα πακέτα εξυπηρετεί διάφορους σκοπούς όπως:

  * Ο ακολουθιακός αριθμός επιτρέπει στα TCP πακέτα να μπαίνουν στην σωστή σειρά όταν φτάσουν στον προορισμό και έπειτα να προωθούνται σαν byte stream στο επίπεδο εφαρμογής. (Κάθε στιγμή, ενδέχεται να αποστέλλονται πολλαπλά TCP πακέτα μεταξύ αποστολέα και παραλήπτη και δεν υπάρχει εγγύηση ότι φτάνουν με τη σωστή σειρά.)

    * Το μήνυμα γνωστοποίησης που στέλλεται από τον αποδέκτη στον παραλήπτη μπορεί να χρησιμοποιήσει τον ακολουθιακό αριθμό για να αναγνωρίσει ποιο TCP πακέτο παρέλαβε.
    * Επίσης ο παραλήπτης μπορεί να χρησιμοποιήσει τον ακολουθιακό αριθμό προκείμενου να εξαλείψει τα διπλά πακέτα. Διπλά πακέτα μπορούν να προκύψουν είτε εξαιτίας διπλών IP δεδομενογραμμάτων είτε εξαιτίας του αλγορίθμου επαναμετάδοσης του TCP, ο οποίος μπορεί να στείλει ξανά πακέτα που έχουν αποσταλεί επιτυχώς στην περίπτωση που η γνωστοποίηση για τα συγκεκριμένα πακέτα δεν έχει φτάσει.

  – *Έλεγχος ροής*
  Ο έλεγχος ροής αποτρέπει έναν γρήγορο αποστολέα να "πλημμυρίσει" έναν αργό παραλήπτη. Για να υλοποιήσει τον έλεγχο ροής, ο παραλήπτης διατηρεί έναν buffer για τα εισερχόμενα δεδομένα. Τα δεδομένα συσσωρεύονται σε αυτόν τον buffer καθώς λαμβάνονται και απομακρύνονται από τον buffer όταν η εφαρμογή τα διαβάσει. Επίσης, με κάθε γνωστοποίηση, ο παραλήπτης ενημερώνει τον αποστολέα σχετικά με το πόσος χώρος υπάρχει διαθέσιμος στον buffer.

  – *Έλεγχος συμφόρησης*
  Οι αλγόριθμοι ελέγχου συμφόρησης του TCP είναι σχεδιασμένοι έτσι ώστε να αποτρέψουν έναν γρήγορο αποστολέα να "πλημμυρίσει" το δίκτυο. Εάν ο αποστολεάς στέλνει πακέτα πιο γρήγορα από όσο μπορεί να τα διαχειριστεί κάποιος ενδιάμεσος δρομολογητής (router), τότε ο δρομολογητής θα αρχίσει να απορρίπτει τα πακέτα. Αυτό μπορεί να οδηγήσει σε αύξηση του ρυθμού απώλειας πακέτων και επομένως να προκαλέσει μείωση της επίδοσης, αν ο αποστολέας εξακολουθεί να απαναμεταδίδει πακέτα με τον ίδιο ρυθμό.

- **Επίπεδο εφαρμογής**
  Το επίπεδο εφαρμογής περιλαμβάνει πρωτόκολλα που χρησιμοποιούνται από εφαρμογές που προσφέρουν υπηρεσίες χρήστη η ανταλλάσσουν δεδομένα μέσω δικτυακών συνδέσεων που εγκαθιδρύουν πρωτόκολλα χαμηλότερων επιπέδων. Ωστόσο το επίπεδο εφαρμογής ενδέχεται να παρέχει βασικές δικτυακές υπηρεσίες όπως πρωτόκολλα δρομολόγησης. Παραδείγματα πρωτοκόλλων του επιπέδου εφαρμογής είναι το Hypertext Transfer Protocol (HTTP), το File Transfer Protocol (FTP), το Simple Mail Transfer Protocol (SMTP) και το Dynamic Host Configuration Protocol (DHCP). Τα δεδομένα κωδικοποιούνται σύμφωνα με το επίπεδο εφαρμογής και ενθυλακώνονται σε τεμάχια στο επί-

πεδο μεταφοράς (όπως TCP ή UDP μυνήματα), τα οποία με τη σειρά τους χρησιμοποιούν πρωτόκολλα χαμηλότερων επιπέδων για πραγματοποιηθεί τελικά η μετάδοση.

Το μοντέλο IP δεν λαμβάνει υπόψη τη μορφοποίηση των δεδομένων ή τον τρόπο παρουσίασής τους ούτε καθορίζει επιπρόσθετα επίπεδα μεταξύ του επιπέδου εφαρμογής και μεταφοράς, όπως συμβαίνει με το μοντέλο OSI [12]. Αυτές οι λειτουργίες επαφίενται στις βιβλιοθήκες και στη προγραμματιστική διεπαφή (API). Τα πρωτόκολλα του επιπέδου εφαρμογής αντιμετωπίζουν τα πρωτόκολλα του επιπέδου μεταφοράς, καθώς και τα χαμηλότερα επίπεδα, σαν “μαύρα κουτιά” τα οποία παρέχουν μια σταθερή σύνδεση μέσω της οποίας επιτυγχάνεται η επικοινωνία. Ωστόσο, πολλές φορές οι εφαρμογές γνωρίζουν κάποια σημαντικά χαρακτηριστικά της σύνδεσης του επιπέδου μεταφοράς όπως τη διεύθυνση IP και τα port numbers.

2.2.2 Προγραμματιστική Διεπαφή socket

Τα sockets (υποδοχείς) είναι μία μέθοδος διαδιεργασιακής επικοινωνίας (IPC), όπου επιτρέπει την ανταλλαγή δεδομένων μεταξύ εφαρμογών που βρίσκονται στο στον ίδιο ή σε διαφορετικούς υπολογιστές. Η πρώτη διαδεδομένη υλοποίηση του socket API αναπτύχθηκε το 1983 στο 4.2BDS, και έκτοτε έχει μεταφερθεί πρακτικά σε όλα τα UNIX συστήματα καθώς και στα περισσότερα λειτουργικά συστήματα.

Στη συνέχεια περιγράφονται περιληπτικά οι τύποι των sockets, τους τομείς/πεδία επικοινωνίας (communication domains) και οι βασικές κλήσεις συστήματος που σχετίζονται με τα sockets καθώς και τα βασικά χαρακτηριστικά των stream και datagram sockets.

Communication Domains

Ένα χαρακτηριστικό των sockets είναι το πεδίο επικοινωνίας (communication domain). Για κάθε socket το πεδίο επικοινωνίας καθορίζει την μέθοδο αναγνώρισης του socket (π.χ. πως πρέπει να είναι διαμορφωμένη η διεύθυνση του socket), όπως επίσης και το εύρος της επικοινωνίας (π.χ. αν η επικοινωνία διεξάγεται μεταξύ εφαρμογών στον ίδιο υπολογιστή ή σε διαφορετικούς που συνδέονται μέσω δικτύου).

Όλα τα σύγχρονα λειτουργικά συστήματα υποστηρίζουν τουλάχιστον τα παρακάτω πεδία.

- UNIX domain (AF_UNIX) : Αυτό το πεδίο επιτρέπει την επικοινωνία μεταξύ εφαρμογών που βρίσκονται στον ίδιο υπολογιστή.

- IPv4 domain (AF_INET) : Επιτρέπει την επικοινωνία μεταξύ εφαρμογών που βρίσκονται σε διαφορετικούς υπολογιστές και συνδέονται μέσω δικτύου IPv4.
- IPv6 domain (AF_INET6) : Επιτρέπει την επικοινωνία μεταξύ εφαρμογών που βρίσκονται σε διαφορετικούς υπολογιστές και συνδέονται μέσω δικτύου IPv6.

Socket Types

Όλεχοι υλοποιήσεις των sockets υποστηρίζουν τουλάχιστον δύο τύπους socket : stream και datagram.

Οι *stream sockets (SOCK_STREAM)* προσφέρουν αξιόπιστη (reliable), δικατευθυντική (bidirectional, byte-stream επικοινωνία. Με αυτούς τους όρους εννοούμε τα παρακάτω:

- *Αξιόπιστη* σημαίνει ότι είναι σίγουρο ότι είτε τα δεδομένα θα φτάσουν στον προορισμό τους άθικτα, ακριβώς όπως μεταδόθηκαν από τον αποστολέα, είτε θα υπάρξει ειδοποίηση κάποιας πιθανής αστοχίας κατά τη μετάδοση.
- *Δικατευθυντική* σημαίνει ότι τα δεδομένα μπορούν να μεταδοθούν από και προς τις δύο κατευθύνσεις μεταξύ των sockets.
- *Byte-stream* σημαίνει ότι δεν υπάρχει η έννοια του μηνύματος δεδομένου μεγέθους.

Τα stream sockets λειτουργούν ως ένα συνδεδεμένο ζευγάρι πάνω από ένα κανάλι επικοινωνίας. Για αυτόν τον λόγο, αυτού του τύπου sockets περιγράφονται ως *συνδεσμικά (connection-oriented)*. Τέλος τα stream sockets συχνά διαχωρίζονται σε *ενεργά* και *παθητικά*. Ενεργό θεωρείται το socket που συνδέεται σε κάποιο άλλο, δηλαδή κάνει connect(), ενώ ως παθητικό εννοούμε το socket που δέχεται συνδέσεις από άλλα και σημειώνεται ως παθητικό όταν κληθεί η listen().

Τα *datagram sockets (SOCK_DGRAM)* επιτρέπουν την ανταλλαγή δεδομένων με τη μορφή μηνυμάτων που ονομάζονται δεδομενογράμματα (datagrams). Σε αυτόν τον τύπο sockets υπάρχουν όρια που σχετίζονται με το μέγεθος του μηνύματος αλλά δεν υπάρχει αξιοπιστία. Τα μηνύματα ενδέχεται να φτάσουν εκτός σειράς, σε διπλότυπα ή να μην φτάσουν καθόλου στον προορισμό.

Τα datagram sockets έχουν την έννοια της ασυνδεσμικής (connectionless) επικοινωνίας. Σε αντίθεση με τα stream sockets τα datagram sockets δεν χρειάζεται να είναι συνδεδεμένα το ένα με το άλλο προκειμένου να χρησιμοποιηθούν. Τέλος στο Internet, τα datagram sockets χρησιμοποιούν το User Datagram Protocol (UDP), ενώ τα stream sockets το Transmission Control Protocol (TCP).

Socket System Calls

Οι κύριες κλήσεις συστήματος που σχετίζονται με την προγραμματιστική διεπαφή Socket είναι οι εξής:

- ***Socket***
  Η κλήση συστήματος socket() δημιουργεί ένα νέο socket. Σε περίπτωση επιτυχίας επιστρέφει έναν περιγραφητή αρχείου (file descriptor) που αναφέρεται στο συγκεκριμένο socket και χρησιμοποιείται από επόμενες κλήσεις συστήματος, ενώ σε περίπτωση αποτυχίας επιστρέφει -1. Ο τρόπος κλήσης της είναι

  ```
  int socket(int domain, int type, int protocol);
  ```

  Το όρισμα domain καθορίζει το πεδίο επικοινωνίας (communication domain) του socket και το όρισμα type προσδιορίζει τον τύπο του socket, όπου στις περισσότερες περιπτώσεις είναι είτε SOCK_STREAM είτε SOCK_DGRAM. Το όρισμα protocol έχει μη μηδενική τιμή για τύπους socket που δεν μελετώνται στην παρούσα εργασία.

- ***Bind***
  Η κλήση συστήματος bind αντιστοιχίζει ένα socket σε μία διεύθυνση. Σε περίπτωση επιτυχίας επιστρέφει 0 αλλιώς -1. Ο τρόπος κλήσης της είναι

  ```
  int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
  ```

  Το όρισμα sockfd είναι ο περιγραφητής αρχείου που επέστρεψε η κλήση socket(). Το όρισμα addr είναι ένας pointer σε μία δομή που προσδιορίζει την διεύθυνση με την οποία θα δεθεί το socket. Ο τύπος της δομής που περνιέται ως όρισμα σε αυτήν την δομή εξαρτάται από το πεδίο επικοινωνίας του socket. Τέλος το όρισμα addrlen αντιπροσωπεύει το μέγεθος της δομής addr.

- ***Listen***

  Η κλήση συστήματος listen() επιτρέπει σε ένα stream socket να δεχτεί συνδέσεις από άλλα stream sockets. Σε περίπτωση επιτυχίας επιστρέφει 0 αλλιώς -1. Ο τρόπος κλήσης της είναι

  ```
  int listen(int sockfd, int backlog);
  ```

  Η κλήση listen() μαρκάρει το stream socket για το οποίο κλήθηκε ως παθητικό. Στη συνέχεια αυτό το socket μπορεί να δεχθεί συνδέσεις από άλλα ενεργά stream sockets. Το όρισμα sockfd αναφέρεται στον file descriptor του socket για το οποίο καλείται η listen(). Το όρισμα backlog είναι ένας ακέραιος που μας χρησιμεύει σε περίπτωση που ένα άλλο socket προσπαθήσει να συνδεθεί (connect()) σε αυτό το socket πριν προλάβει να εκτελεστεί η accept().

- ***Accept***

  Η κλήση συστήματος accept() δέχεται μία εισερχόμενη σύνδεση πάνω σε ένα socket για το οποίο έχει κληθεί η listen(). Αν δεν υπάρχουν συνδέσεις που να περιμένουν η accept() μπλοκάρει μέχρι να έρθει μια αίτηση για σύνδεση. Σε περίπτωση επιτυχίας επιστρέφει έναν νέο file descriptor, ενώ σε περίπτωση αποτυχίας -1. Ο τρόπος κλήσης της είναι

  ```
  int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
  ```

  Το όρισμα sockfd αναφέρεται στον file descriptor που επέστρεψε η socket(), ενώ τα άλλα δύο ορίσματα επιστρέφουν την διεύθυνση του peer socket. Το όρισμα addr δείχνει σε μία δομή όπου επιστρέφεται η διεύθυνση του peer και το addrlen δείχνει σε έναν ακέραιο, ο οποίος περιέχει το μέγεθος της δομής addr.

  Το βασικό σημείο σχετικά την accept() είναι ότι δημιουργεί ένα νέο socket και είναι αυτό το καινούριο socket που συνδέεται με το peer socket που εκτελεί connect(). Ο file descriptor που επιστρέφεται για το συνδεδεμένο socket είναι αποτέλεσμα της κλήσης accept(). Το socket που “ακούει” (listening socket (sockfd)) παραμένει ανοιχτό και μπορεί να χρησιμοποιηθεί για επόμενες συνδέσεις. Μία τυπική εφαρμογή server δημιουργεί ένα listening socket, το δένει (bind) σε μία γνωστή διεύθυνση και έπειτα χειρίζεται τις αιτήσεις του client δεχόμενη συνδέσεις μέσω αυτού του socket.

- ***Connect***

  Η κλήση συστήματος connect() συνδέει το ενεργό socket, στο οποίο αναφερόμαστε μέσω του ορίσματος sockfd, με το listening socket , του οποίου η διεύθυνση προσδιορίζεται από τα ορίσματα addr και addrlen. Σε περίπτωση επιτυχίας επιστρέφει 0 αλλιώς -1. Ο τρόπος κλήσης της είναι

  ```
  int connect(int sockfd, struct sockaddr *addr, socklen_t addrlen);
  ```

  Τα ορίσματα addr και addrlen προσδιορίζονται και λειτουργούν όπως τα αντίστοιχα ορίσματα της κλήσης bind(). Αν η connect() αποτύχει και εμείς επιθυμούμε να προσπαθήσουμε ξανά, το πρότυπο του SUSv3 ορίζει ότι πρέπει να κλείσουμε το socket και να ξαναπροσπαθήσουμε την σύνδεση με ένα νέο socket.

2.3 Virtualization

2.3.1 Εικονικές Μηχανές

Εικονική μηχανή είναι η υλοποίηση μιας μηχανής σε λογισμικό, που εκτελεί προγράμματα όπως μια φυσική μηχανή. Οι Popek και Goldberg [10] έδωσαν τον εξής ορισμό: "Εικονική μηχανή θεωρείται ένα αποδοτικό και απομονωμένο αντίγραφο μιας πραγματικής μηχανής". Ωστόσο η έννοια αυτή σήμερα περιλαμβάνει υλοποιήσεις που δεν αποτελούν αντίγραφα πραγματικών μηχανών. Πάνω σε αυτό το διαχωρισμό βασίζετια η κατηγοριοποίηση των εικονικών μηχανών σε εικονικές μηχανές συστήματος και εικονικές μηχανές διεργασίας.

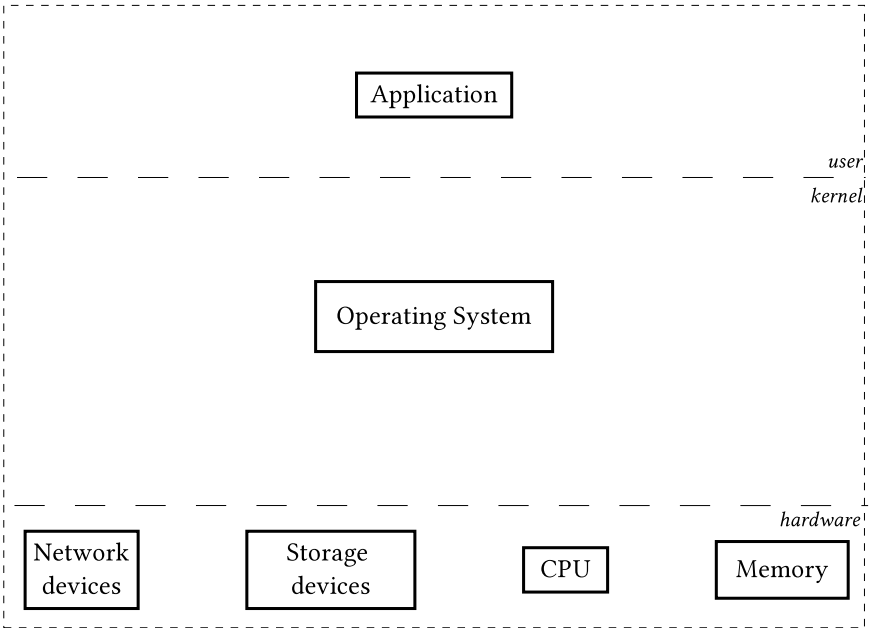
Μια εικονική μηχανή διεργασίας εκτελείται σαν μια συνηθισμένη εφαρμογή, μέσα σε ένα λειτουργικό σύστημα και υποστηρίζει μόνο ένα πρόγραμμα. Υλοποιείται με τη χρήση διερμηνέα και σκοπός της είναι να παρέχει ένα προγραμματιστικό περιβάλλον ανεξάρτητο του λειτουργικού συστήματος και του υλικού. Χαρακτηριστικό παράδειγμα αποτελεί το Java virtual machine, το οποίο διερμηνεύει τον μεταγλωττισμένο κώδικα της Java για έναν επεξεργαστή, έτσι ώστε να μπορούν να εκτελεστούν οι εντολές κάθε προγράμματος Java. Με αυτόν τον τρόπο οι προγραμματιστές μπορούν να συντάσσουν προγράμματα Java σε οποιοδήποτε υπολογιστή, αφού σε αυτόν θα υπάρχει η κατάλληλη εικονική μηχανή που θα διερμηνεύσει τον τελικό κώδικα που υποστηρίζει ο αντίστοιχος επεξεργαστής.

Από την άλλη πλευρά, η εικονική μηχανή συστήματος είναι ουσιαστικά ένα υπολογιστικό σύστημα εμφωλευμένο μέσα σε ένα άλλο υπολογιστικό σύστημα και λειτουργεί σαν να του ανήκουν όλοι οι πόροι του συστήματος, αν και την ίδια στιγμή μπορεί να συνυπάρχει με ένα ή περισσότερα συστήματα. Πιο συγκεκριμένα, η εικονική μηχανή συστήματος, που στη συνέχεια του κειμένου θα αποκαλείται απλώς "εικονική μηχανή" (virtual machine – VM), αποτελεί μία προσομοίωση ενός πραγματικού υπολογιστικού συστήματος κατά τη δημιουργία της οποίας ο χρήστης μπορεί να ορίσει γνωστές παραμέτρους συστήματος, όπως τον αριθμό των (εικονικών) πυρήνων, το μέγεθος της μνήμης, το μέγεθος της cache, τη διεύθυνση MAC της κάρτας δικτύου καθώς και τη διεύθυνση IP των διεπαφών της. Επίσης υποστηρίζει ένα λειτουργικό σύστημα καθώς και την εκτέλεση οποιαδήποτε εφαρμογής του χρήστη. Για παράδειγμα, σε ένα πραγματικό σύστημα που περιέχει δύο πυρήνες και 4GB μνήμη μπορούν να δημιουργηθούν δύο εικονικές μηχανές με δύο (εικονικούς) πυρήνες και 4GB μνήμη η κάθε μία, που η μία να υποστηρίζει το λειτουργικό Linux, η άλλη λειτουργικό Windows και να λειτουργούν ταυτόχρονα χωρίς να παρεμβαίνουν οι εκτελέσεις των εφαρμογών της μίας σε αυτές της άλλης.

Η ταυτόχρονη συνύπαρξη των εικονικών μηχανών επιτυγχάνεται

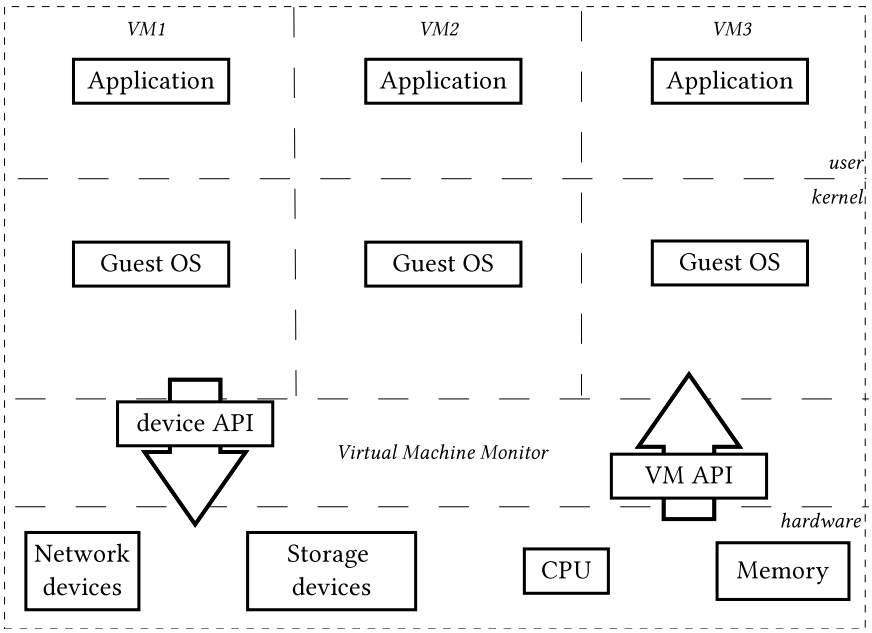
με χρήση του “ελεγκτή εικονικής μηχανής” (virtual machine monitor - VMM). Ο ελεγκτής εικονικής μηχανής είναι ουσιαστικά το σύστημα που πολυπλέκει την πρόσβαση στο υλικό για κάθε εικονική μηχανή μεσολαβώντας στην επικοινωνία των λειτουργικών συστημάτων των εικονικών μηχανών με το υλικό. Μερικές από τις λειτουργίες του είναι η διασφάλιση του απομονωμένου περιβάλλοντος εκτέλεσης των εικονικών μηχανών, η πολύπλεξη των αιτημάτων για πρόσβαση στους πόρους του συστήματος από τα φιλοξενούμενα λειτουργικά συστήματα και η αντιστοίχηση των διακοπών υλικού σε διακοπές λογισμικού έτσι ώστε να ειδοποιείται το κατάλληλο φιλοξενούμενο λειτουργικό σύστημα για διάφορα συμβάντα του υλικού.

Στο Σχήμα 2.1, αναπαριστάται η ιεραρχική τοποθέτηση των τμημάτων του λογισμικού σε σχέση με το υλικό ενός υπολογιστή που δεν περιέχει εικονικό περιβάλλον. Σε αυτή τη περίπτωση το λειτουργικό σύστημα συνδέεται άμεσα με το υλικό, καθώς περιέχει τους οδηγούς του και επικοινωνεί με αυτό μέσω διακοπών. Επίσης, έχει τον πλήρη έλεγχο των πόρων του υπολογιστή, όπως τη μνήμη, οργανώνει την εκτέλεση των εφαρμογών που δημιουργούνται από τον χρήστη και δρα ως διεπαφή μεταξύ των εφαρμογών και του υλικού. Με άλλα λόγια, το λειτουργικό σύστημα κατέχει τον πλήρη έλεγχο του συστήματος και βρίσκεται, όπως λέγεται, στο δακτυλίδι 0 (ring 0).



Σχήμα 2.1: Συμβατικό Υπολογιστικό Σύστημα

Στο Σχήμα 2.2 παρουσιάζεται η παραπάνω ιεραρχία σε εικονικό περιβάλλον. Σε αυτή τη περίπτωση ο ελεγκτής εικονικής μηχανής κατέχει τον πλήρη έλεγχο του συστήματος. Αυτό σημαίνει ότι εκτελεί τόσες λειτουργίες όσες εκτελούσε το λειτουργικό σύστημα στο

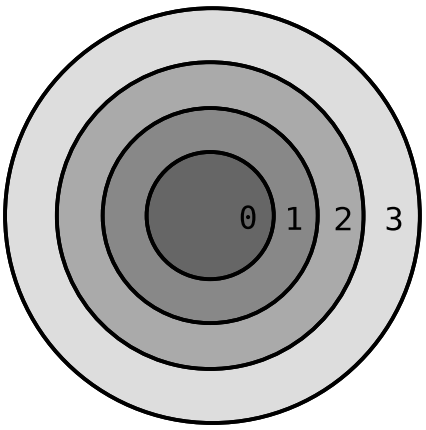


Σχήμα 2.2: Ελεγκτής εικονικών μηχανών

σχήμα 2.1 και επιπλέον οργανώνει την εκτέλεση των εικονικών μηχανών που δημιουργούνται. Μέρος της οργάνωσης αυτής είναι ο τρόπος με τον οποίο οι εικονικές μηχανές επικοινωνούν με το υλικό αλλά και ο τρόπος με τον οποίο αυτές μοιράζονται τους πόρους του υπολογιστικού συστήματος. Από τη πλευρά τους, τα φιλοξενούμενα λειτουργικά συστήματα (guest operating systems) οργανώνουν την εκτέλεση των εφαρμογών τους και επικοινωνούν με τον ελεγκτή εικονικής μηχανής σε ζητήματα πρόσβασης στο πραγματικό υλικό.

Τα σύγχρονα λειτουργικά συστήματα δεν επιτρέπουν στις εφαρμογές χώρου χρήστη να εκτελούν λειτουργίες αυξημένων δυνατοτήτων. Για παράδειγμα, μόνο το λειτουργικό σύστημα μπορεί να φορτώσει οδηγούς συσκευών ή να προσπελάσει απευθείας το υλικό. Για να περιοριστούν όλες οι εφαρμογές υπό εκτέλεση μόνο σε ένα υποσύνολο των πόρων, το λειτουργικό σύστημα και ο επεξεργαστής συνεργάζονται χρησιμοποιώντας τα επίπεδα δισκαιωμάτων Σχήμα 2.3.

Η επεξεργαστική μονάδα εκτελεί εντολές σε ένα από αυτά τα επίπεδα. Το επίπεδο (δακτυλίδι) 0 έχει τα περισσότερα δικαιώματα και το επίπεδο 3 τα λιγότερα. Οι πόροι που προστατεύονται μέσω των επιπέδων είναι: η μνήμη, οι θύρες εισόδου/εξόδου και οι εντολές του επεξεργαστή. Το λειτουργικό σύστημα εκτελείται στο επίπεδο 0 (κατάσταση πυρήνα - kernel mode), οι εφαρμογές του χρήστη στο επίπεδο 3 (κατάσταση χρήστη - user mode) και τα επίπεδα 1 και 2 παραμένουν αχρησιμοποίητα. Όμως, σε έναν υπολογιστή με εικονικό περιβάλλον, ο ελεγκτής εικονικής μηχανής θα πρέπει να μπορεί να προσπελάσει την μνήμη, τον επεξεργαστή και τις συσκευές εισόδου-



Σχήμα 2.3: CPU privilege levels

/εξόδου. Αυτό σημαίνει ότι θα πρέπει να εκτελείται στο επίπεδο με τα περισσότερα δικαιώματα (επίπεδο 0). Από την άλλη πλευρά, τα φιλοξενούμενα λειτουργικά συστήματα περιμένουν να έχουν κι αυτά πρόσβαση σε κάποιους από τους πόρους. Επειδή, όμως, μόνο ένας πυρήνας μπορεί να βρίσκεται στο επίπεδο 0, τα φιλοξενούμενα λειτουργικά συστήματα θα πρέπει να εκτελούνται είτε σε άλλο επίπεδο, με λιγότερα δικαιώματα, (επίπεδο 1) είτε θα πρέπει να τροποποιηθούν για να εκτελούνται στο επίπεδο 3.

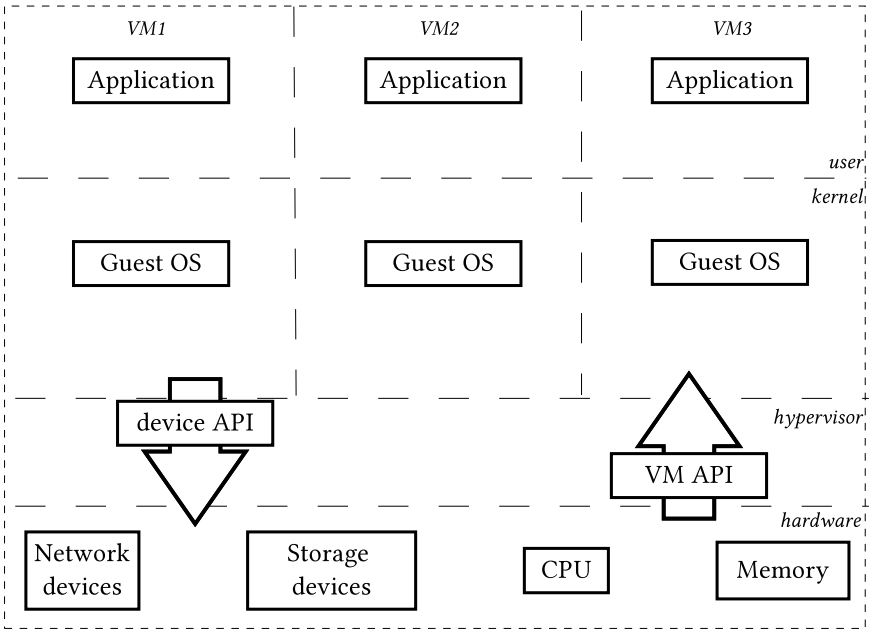
Ο τρόπος με τον οποίο δομείται ένα εικονικό περιβάλλον δεν είναι μοναδικός και εξαρτάται κυρίως από τον σχεδιασμό του ελεγκτή εικονικής μηχανής. Οι δύο βασικές παράμετροι σχεδιασμού είναι οι εξής:

- η άμεση επαφή του ελεγκτή εικονικής μηχανής με το υλικό (σε αντίθεση με την παρεμβολή ενός λειτουργικού συστήματος που θα πολυπλέκει την πρόσβαση)

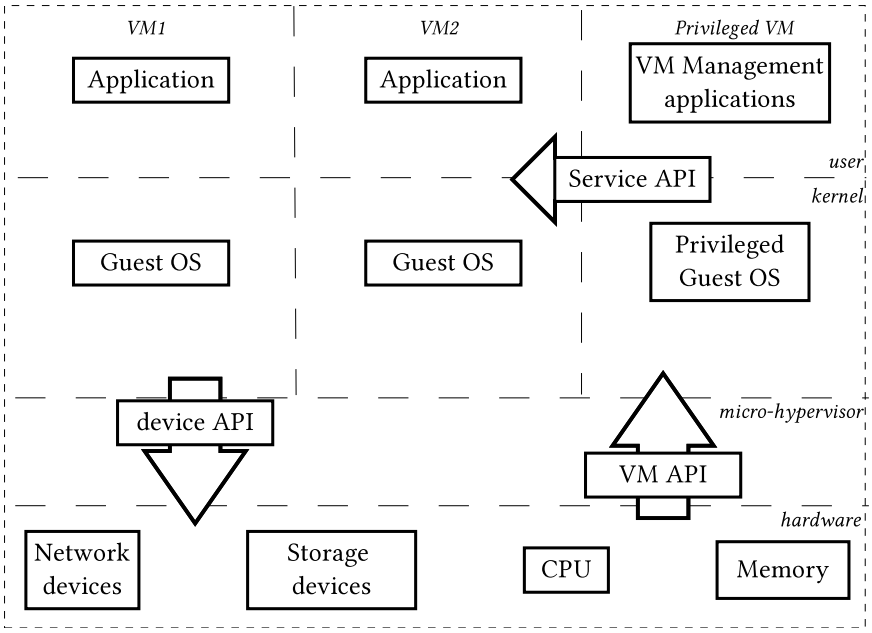
- η πλήρης προσομοίωση του υλικού για πρόσβαση από τις εικονικές μηχανές

Βάσει των παραπάνω παραμέτρων, προκύπτουν δύο τύποι virtualization: ο τύπος I και ο τύπος II. Στον τύπο I ανήκουν οι ελεγκτές εικονικής μηχανής που τουλάχιστον ένα μέρος τους βρίσκεται σε άμεση επαφή με το υλικό. Σε αυτόν τον τύπο υπάρχουν δύο υποτύποι ελεγκτών εικονικής μηχανής: ο επιβλέπων (Hypervisor), που εκτελείται εξολοκλήρου πάνω στο υλικό (Σχήμα 2.4) και το λειτουργικό σύστημα υπηρεσίας (Service OS), που ουσιαστικά λειτουργεί ως μια εικονική μηχανή αυξημένων δυνατοτήτων (Σχήμα 2.5).

Στον τύπο II ανήκουν οι ελεγκτές εικονικής μηχανής που εκτελούνται πάνω από ένα ήδη εγκατεστημένο λειτουργικό σύστημα (Host OS). Ο ελεγκτής και το υπάρχον λειτουργικό σύστημα βρίσκονται

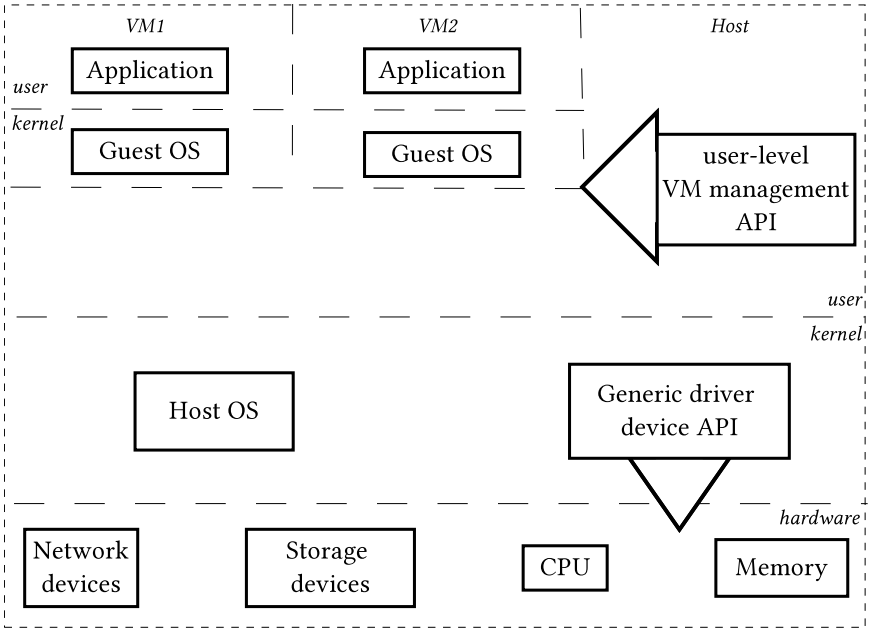


Σχήμα 2.4: Hypervisor



Σχήμα 2.5: Service OS

στο δακτυλίδι 0 (ring 0). Μία αίτηση για πρόσβαση των εικονικών μηχανών στις συσκευές Ε/Ε, ανακατευθύνεται μέσω του ελεγκτή (VMM) στους εικονικούς οδηγούς, που βρίσκονται στον ελεγκτή επιπέδου χρήστη (User Level Monitor) και εκείνοι με τη σειρά τους την κατευθύνουν προς τους πραγματικούς οδηγούς του λειτουργικού συστήματος. Στο Σχήμα 2.6 παρουσιάζεται η δομή του.

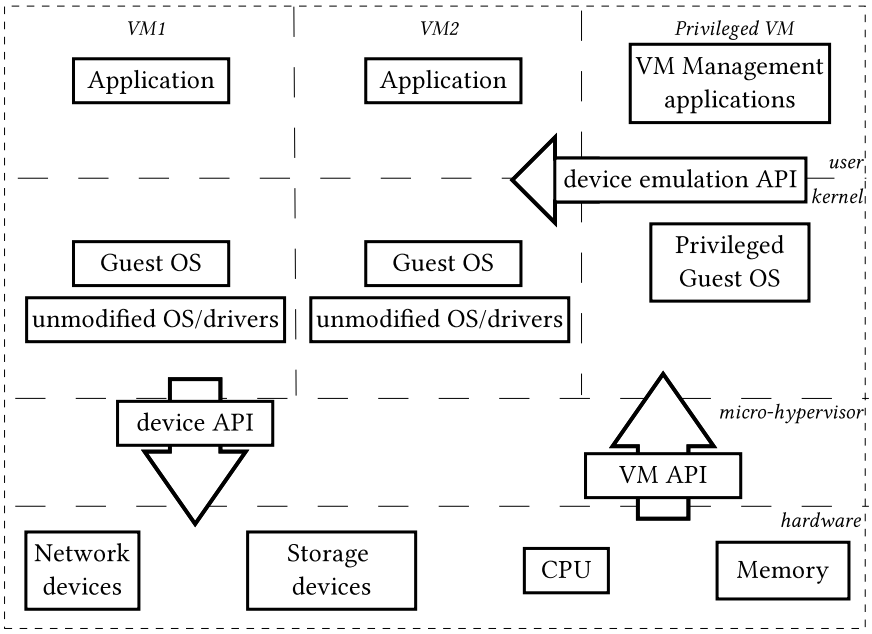


Σχήμα 2.6: Host OS

Όσο απομακρύνεται το λογισμικό του ελεγκτή από το υλικό, τόσο μικραίνει η εξάρτησή του από αυτό και τόσο μεγαλώνει η φορητότητά του. Αυτό οφείλεται στο ότι οι οδηγοί μεταβιβάζουν μέρος των λειτουργιών τους στον ελεγκτή των εικονικών μηχανών και τείνουν να αποκτήσουν τον χαρακτήρα των οδηγών ενός απλού λειτουργικού συστήματος. Πιο συγκεκριμένα, οι οδηγοί ενός ελεγκτή τύπου Ι θα πρέπει να επικοινωνούν με επιτυχία με τις αντίστοιχες συσκευές, να οργανώνουν κατάλληλα τα αιτήματα από τις υπάρχουσες εικονικές μηχανές και να ανακατευθύνουν τις διακοπές και τα δεδομένα, που προέρχονται από τις συσκευές, στην κατάλληλη εικονική μηχανή. Σε αυτή τη περίπτωση, οι οδηγοί έχουν δημιουργηθεί ειδικά για το συγκεκριμένο υλικό και έτσι η φορητότητά τους σε άλλο υλικό θα είναι περιορισμένη. Στον ελεγκτή τύπου Ι, οι οδηγοί έχουν μεταφερθεί στο λειτουργικό σύστημα υπηρεσίας, με το οποίο πρέπει να επικοινωνούν τα υπόλοιπα φιλοξενούμενα λειτουργικά συστήματα έτσι ώστε να αποκτήσουν πρόσβαση στο υλικό. Το λειτουργικό σύστημα υπηρεσίας, όμως, είναι ένα κοινό, ελαφρώς τροποποιημένο λειτουργικό σύστημα. Αυτό σημαίνει ότι ως προς την επικοινωνία με το υλικό,

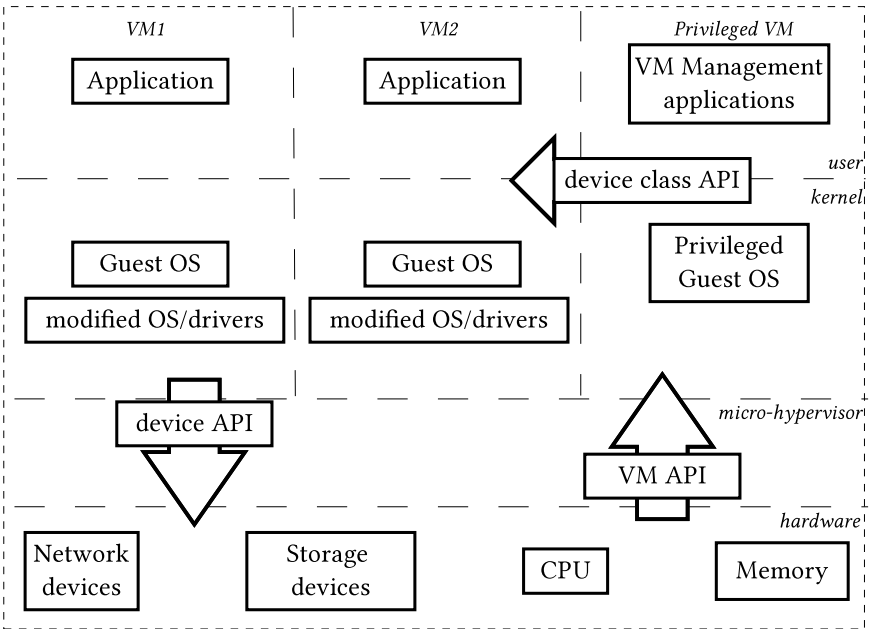
οι οδηγοί θα είναι πανομοιότυποι με αυτούς που υπάρχουν στα δημοφιλή λειτουργικά συστήματα, ενώ ως προς την επικοινωνία με τις υπόλοιπες εικονικές μηχανές θα υπάρχουν τροποποιήσεις. Το γεγονός αυτό, καθιστά τον τύπο αυτό πιο ευέλικτο, αφού βασίζεται σε οδηγούς ευρείας χρήσης. Τέλος, στον ελεγκτή τύπου II οι οδηγοί βρίσκονται στο ήδη υπάρχον λειτουργικό σύστημα. Αυτό σημαίνει ότι είναι ανεξάρτητοι του εικονικού περιβάλλοντος και ταυτίζονται πλήρως με τους συμβατικούς οδηγούς των κοινών λειτουργικών συστημάτων. Οι λειτουργίες της οργάνωσης των αιτημάτων και της ανακατεύθυνσης των δεδομένων και των διακοπών έχουν πλήρως μεταβιβαστεί στον ελεγκτή. Συνεπώς,όσο απομακρύνεται το λογισμικό του ελεγκτή από το υλικό, τόσο αυξάνεται η ευελιξία.

Όμως, ταυτόχρονα αυξάνεται η πολυπλοκότητα του συστήματος και μειώνεται η απόδοσή του. Αυτό συμβαίνει διότι προστίθενται επιπλέον επίπεδα ανάμεσα σε μία εικονική μηχανή και στο πραγματικό υλικό. Στον τύπο II, η εικονική μηχανή συνομιλεί απευθείας με τον ελεγκτή και τους οδηγούς. Στον τύπο I, η εικονική μηχανή συνομιλεί πρώτα με την την εικονική μηχανή υπηρεσίας και έπειτα αυτή επικοινωνεί με το υλικό. Τέλος, στον τύπο II, η εικονική μηχανή επικοινωνεί με τον ελεγκτή, ο ελεγκτής με τον ελεγκτή επιπέδου χρήστη κι εκείνος με το υλικό.



Σχήμα 2.7: Full Virtualization

Δημιουργούνται έτσι δύο κατηγορίες τεχνικών virtualization: η πλήρης (full virtualization) και η μερική (paravirtualization). Οι τεχνικές της πρώτης κατηγορίας (Σχήμα 2.7), απαιτούν ελεγκτές που προ-



Σχήμα 2.8: Paravirtualization

σομοιώνουν πλήρως το υλικό προς τη πλευρά των εικονικών μηχανών, ενώ οι τεχνικές της δεύτερης κατηγορίας (Σχήμα 2.8) απαιτούν ελεγκτές που παρουσιάζουν εικονικούς οδηγούς στη πλευρά των εικονικών μηχανών και οργανώνουν την συνομιλία τους με τους πραγματικούς.

Στην τεχνική πλήρους virtualization, όπως αναφέρθηκε παραπάνω, ο ελεγκτής προσομοιώνει πλήρως το υλικό. Αυτό σημαίνει ότι χρησιμοποιεί το λογισμικό για να προσομοιώσει τις λειτουργίες του υλικού έτσι, ώστε να δίνεται στα φιλοξενούμενα λειτουργικά συστήματα η ψευδαίσθηση ότι επικοινωνούν απευθείας με τις συσκευές, αν και στη πραγματικότητα παρεμβάλλεται ο ελεγκτής. Σε αυτή την περίπτωση λοιπόν, η λειτουργία του ελεγκτή είναι να ανακατευθύνει και να οργανώνει τις κλήσεις των οδηγών των φιλοξενούμενων λειτουργικών συστημάτων στις πραγματικές συσκευές. Η τεχνική αυτή επιτρέπει στις εικονικές μηχανές να έχουν μη τροποποιημένα λειτουργικά συστήματα και μη τροποποιημένους οδηγούς. Δηλαδή, θα μπορούσε να φιλοξενηθεί οποιοδήποτε Λ/Σ, αφού θα δημιουργείται η εντύπωση στο Λ/Σ ότι εκτελείται πάνω σε πραγματικό υλικό και όχι μέσα σε εικονικό περιβάλλον. Από την άλλη πλευρά, η τεχνική αυτή παρουσιάζει μειωμένη επίδοση, αφού η διαδικασία της προσομοίωσης του υλικού είναι απαιτητική σε πόρους και χρονοβόρα.

Από την άλλη πλευρά, με το paravirtualization, τα φιλοξενούμενα λειτουργικά συστήματα έχουν επίγνωση ότι εκτελούνται μέσα σε εικονικό περιβάλλον. Τα ίδια, αλλά και οι οδηγοί που περιέχουν, έχουν τροποποιηθεί σε σχέση με τα κοινά λειτουργικά συστήματα

έτσι ώστε να επικοινωνούν με τις κατάλληλες διεπαφές του ελεγκτή. Με άλλα λόγια, η κλήση ενός οδηγού στην αντίστοιχη συσκευή αντικαθίσταται από μια κλήση στην αντίστοιχη διεπαφή του ελεγκτή και από μια κλήση της διεπαφής αυτής στη συσκευή. Συνεπώς, η τεχνική αυτή είναι πιο αποδοτική, αφού ο ελεγκτής απλώς αποκρίνεται σε κλήσεις των φιλοξενούμενων εικονικών μηχανών, αλλά απαιτεί την τροποποίηση των λειτουργικών συστημάτων και των οδηγών, κάτι που είναι λιγότερο ευέλικτο και πολλές φορές κοστίζει σε χρόνο εκτέλεσης καθώς και υλοποίησης. Η τεχνική paravirtualization μπορεί να αποφευχθεί, αν ο επεξεργαστής, το υποσύστημα Ε/Ε καθώς και η εκάστοτε συσκευή Ε/Ε του υπολογιστικού συστήματος υποστηρίζουν στο υλικό επεκτάσεις virtualization (όπως τα extended instruction sets των επεξεργαστών Intel/AMD, τα υποσυστήματα IOMMU, καθώς και οι συσκευές που υποστηρίζουν I/O Virtualization).

2.3.2 Xen

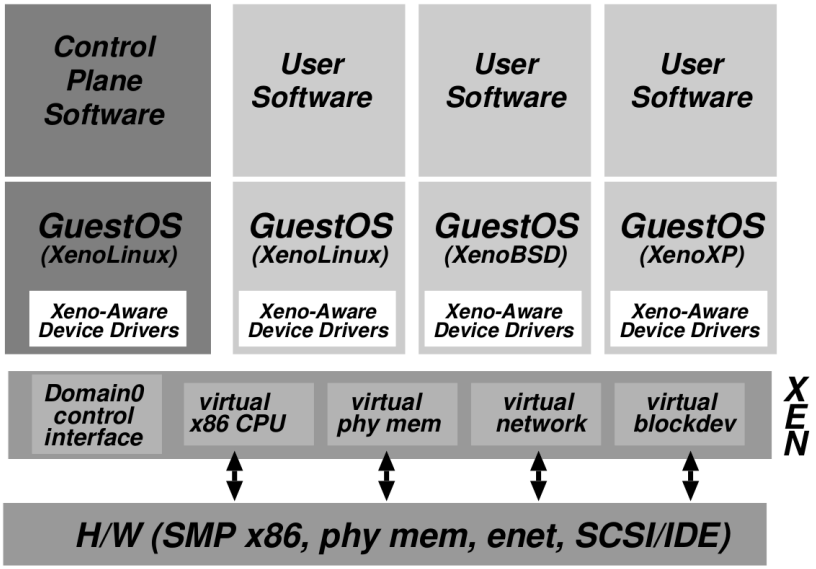
Αρχιτεκτονική

Στο Σχήμα 2.9 παρουσιάζεται η αρχιτεκτονική του Xen. Το Xen είναι ελεγκτής τύπου I και, πιο συγκεκριμένα, ανήκει στον υποτύπο “λειτουργικό σύστημα υπηρεσίας”. Το ρόλο της εικονικής μηχανής υπηρεσίας τον παίζει το driver domain, μια εσικονική μηχανή που ονομάζεται επίσης και Domain 0 (Dom0), ενώ η εικονικές μηχανές VM1..N είναι κοινές εικονικές μηχανές (DomUs), που επικοινωνούν με το Dom0 για τη προσπέλαση στο υλικό. Το Dom0 δημιουργείται κατά την εκκίνηση του συστήματος (boot time) και έχει δικαίωμα να χρησιμοποιεί τη διεπαφή ελέγχου του συστήματος (control interface), με την οποία μπορεί ανάμεσα στα άλλα να δημιουργεί και να τερματίζει εικονικές μηχανές, να δημιουργεί εικονικές διεπαφές δικτύου και να διαχειρίζεται τη πρόσβαση των εικονικών μηχανών στο υλικό.

Όσον αφορά τα επίπεδα δικαιωμάτων, ο ελεγκτής εκτελείται στο επίπεδο 0, το Dom0 καθώς και τα DomUs εκτελούνται στο επίπεδο 1 και οι εφαρμογές χρήστη στο επίπεδο 3. Όταν τα φιλοξενούμενα λειτουργικά συστήματα επιθυμούν να εκτελέσουν μια εντολή σε αυξημένα δικαιώματα, επικοινωνούν με τον Xen μέσω μιας υπερκλήσης (hypercall) και εκτελείται η εντολή από τη πλευρά του Xen. Για μια εικονική μηχανή, μια υπερκλήση είναι ότι είναι η κλήση συστήματος για μια εφαρμογή.

Διαχείριση Πόρων

Η αρχική ανάθεση μνήμης για κάθε εικονική μηχανή καθορίζεται τη στιγμή της δημιουργίας της. Αυτό σημαίνει ότι η μνήμη διαχωρίζεται στατικά μεταξύ των εικονικών μηχανών (DomU), παρέχοντας



Σχήμα 2.9: Αρχιτεκτονική του Xen

ισχυρή απομόνωση. Αν όμως ένα DomU χρειαστεί περισσότερη μνήμη μπορεί να διεκδικήσει επιπρόσθετες σελίδες από το Xen, μέχρι να φτάσει κάποιο καθορισμένο όριο. Αντίθετα, αν το DomU επιθυμεί να αποταμιεύσει μνήμη που δε χρησιμοποιεί μπορεί να απελευθερώσει σελίδες και να τις επιστρέψει στο Xen. Τα τροποποιημένα φιλοξενούμενα λειτουργικά συστήματα περιέχουν έναν ειδικό οδηγό, τον balloon driver, ο οποίος παρέχει τις δύο λειτουργίες που μόλις περιγράφηκαν. Επειδή το Xen δεν εγγυάται ότι θα παραχωρήσει συνεχόμενες περιοχές μνήμης (φυσική μνήμη, physical memory) στα φιλοξενούμενα λειτουργικά συστήματα, τα ίδια δημιουργούν την ψευδαίσθηση ότι κατέχουν συνεχόμενη φυσική μνήμη (ψευδο-φυσική μνήμη, pseudo-physical memory). Το Xen παρέχει αποδοτική αντιστοίχηση μεταξύ των διευθύνσεων αυτών, αφού συντηρεί έναν διαμοιραζόμενο πίνακα μεταξύ των Doms, μέσω του οποίου ελέγχει την εγκυρότητά τους. Θα πρέπει να τονιστεί, ότι τα φιλοξενούμενα λειτουργικά συστήματα έχουν επίγνωση ότι η μνήμη που τους ανατίθεται δεν είναι συνεχής, αποφεύγεται το επιπλέον κόστος συντήρησης δομών, όπως οι shadow page tables.

Η διαμοιραζόμενη πρόσβαση στις συσκευές εισόδου/εξόδου γίνεται μέσω των τροποποιημένων οδηγών, οι οποίοι ακολουθούν το μοντέλο του διαχωρισμένου οδηγού συσκευών (split driver model). Αυτό σημαίνει ότι ο οδηγός χωρίζεται σε δύο κομμάτια: το front-end που βρίσκεται στο φιλοξενούμενο λειτουργικό σύστημα ενός DomU και το back-end που βρίσκεται στο λειτουργικό σύστημα υπηρεσίας

του Dom0. Πιο συγκεκριμένα ένας οδηγός συσκευής του συστήματος Xen αποτελείται από τέσσερις βασικές μονάδες:

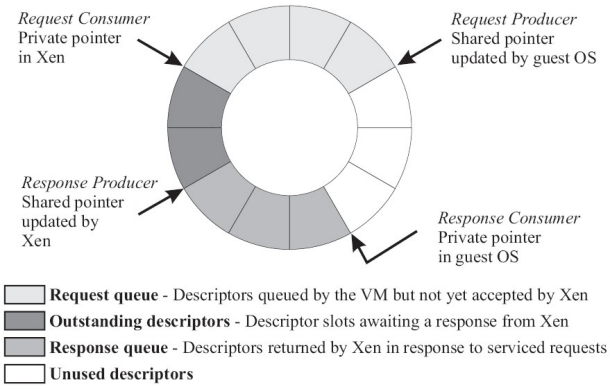
- Τον πραγματικό οδηγό (native driver).
- Το κάτω μισό του διαχωρισμένου οδηγού (backend driver).
- Τους μοιραζόμενους αποθηκευτικούς χώρους σε δομή δακτυλιδιού (ring buffers).
- Το πάνω μισό του διαχωρισμένου οδηγού (frontend driver).

Ο πραγματικός οδηγός, που βρίσκεται στο λειτουργικό σύστημα υπηρεσίας του Dom0 όπως έχει αναφερθεί, είναι ένας κοινός οδηγός που υπάρχει στα συμβατικά λειτουργικά συστήματα. Αυτό που αλλάζει είναι ότι ο ελεγκτής λαμβάνει τις διακοπές που προέρχονται από τη συσκευή και τις μετατρέπει σε γεγονότα του Xen (Xen events). Το κάτω μισό του διαχωρισμένου οδηγού επιτελεί δύο λειτουργίες: παρέχει πολυπλεξία και μία γενική διεπαφή του οδηγού. Το πρώτο επιτρέπει σε πάνω από μία εικονική μηχανή να χρησιμοποιεί τη συσκευή ενώ το δεύτερο παρέχει μια διεπαφή που είναι ανεξάρτητη από το υλικό, όπως η ανάγνωση και η εγγραφή δεδομένων σε μια συσκευή μπλοκ. Το πάνω μισό του διαχωρισμένου οδηγού είναι συνήθως πολύ απλό, αφού αρμοδιότητά του είναι να παρέχει στον χρήστη μια εικονική διεπαφή του οδηγού (τις διαθέσιμες λειτουργίες του οδηγού) και να μεταβιβάζει τις εντολές και τα δεδομένα του χρήστη από το DomU, στο οποίο βρίσκεται, στο Dom0. Η μεταβίβαση αυτή γίνεται μέσω των κοινών αποθηκευτικών χώρων (ring buffers), των καναλιών γεγονότων (event channels) και της βάσης δεδομένων XenStore του συστήματος Xen.

Επικοινωνία μεταξύ εικονικών μηχανών

Απαραίτητο στοιχείο της υλοποίησης του μοντέλου διαχωρισμένου οδηγού είναι η επικοινωνία μεταξύ του driver domain και των εικονικών μηχανών. Η μεταφορά των δεδομένων μεταξύ τους, δηλαδή η μεταφορά των εντολών του χρήστη αλλά και της καθαρής πληροφορίας γίνεται μέσω μιας διαμοιραζόμενης θέσης μνήμης που είναι σε δομή δακτυλιδιού.

Η δομή του δακτυλιδιού είναι μια κυκλική ουρά με τέσσερις δείκτες και δυο ειδών τύπου δεδομένων ως στοιχεία της (Σχήμα 2.10). Πιο συγκεκριμένα, τα στοιχεία του δακτυλίου μπορεί να είναι είτε αιτήματα (requests) είτε απαντήσεις (responses). Για παράδειγμα, αν ο ρόλος του δακτυλίου είναι να μεταφέρει δεδομένα από το DomU στο Dom0, τα αιτήματα είναι δομές (structs) που μεταφέρουν δεδομένα προς αυτή τη κατεύθυνση και οι απαντήσεις είναι δομές που



Σχήμα 2.10: Δομή του Xen Ring

μεταφέρουν αποκρίσεις των αιτημάτων προς την αντίστροφη κατεύθυνση. Τα ονόματα των τεσσάρων δεικτών που συμμετέχουν είναι: παραγωγός αιτημάτων (request producer - RP), καταναλωτής αιτημάτων (request consumer - RC), παραγωγός απαντήσεων (response producer - RP), καταναλωτής απαντήσεων (response consumer - RC). Σε αντιστοιχία με το παραπάνω παράδειγμα, ο πρώτος και ο τελευταίος θα ανήκουν στο DomU και οι άλλοι δύο στον Dom0. Όταν προστίθεται ένα αίτημα ή μια απάντηση ο αντίστοιχος παραγωγός δείχνει μια θέση μπροστά (όπου και τοποθετείται η νέα δομή) και όταν εξυπηρετείται ένα αίτημα ή μια απάντηση ο αντίστοιχος καταναλωτής προωθείται μια θέση μπροστά, δείχνοντας στην επόμενη δομή προς εξυπηρέτηση.

Η έννοια της διαμοιραζόμενης μνήμης σημαίνει ότι και τα δύο Dom που συμμετέχουν έχουν δικαίωμα να προσπελάσουν τη μνήμη αυτή ανά πάσα στιγμή. Για να οριστεί ένα μέγεθος μνήμης (συνήθως σελίδες) ως διαμοιραζόμενο χρησιμοποιείται ο μηχανισμός της παραχώρησης σελίδων (grant pages) του Xen, το οποίο συντηρεί έναν πίνακα παραχώρησης (grant table) όπου αναγράφεται ποια σελίδα ανήκει σε ποιο Dom. Συνήθως, λοιπόν, το DomU δεσμεύει μια σελίδα, την αρχικοποιεί ως δομή δακτυλίδι, ορίζει τις δομές των αιτημάτων και των απαντήσεων, την παραχωρεί στον Dom0 και έπειτα ο Dom0 την αποδέχεται.

Για την επικοινωνία μεταξύ των εικονικών μηχανών χρησιμοποιείται ο μηχανισμός XenStore. Το XenStore έχει μια δεντρική μορφή, στα φύλλα των οποίων υπάρχουν πεδία με τις αντίστοιχες τιμές τους. Τα πεδία αυτά αποτελούν παραμέτρους λειτουργίας των εικονικών μηχανών και είναι ορατά από όλα τα μέρη, βάσει ενός πίνακα δυνατοτήτων πρόσβασης. Επίσης κάθε εικονική μηχανή μπορεί να τοποθετήσει στους κόμβους που το ενδιαφέρουν έναν παρατηρητή (watch) με την αντίστοιχη συνάρτηση εξυπηρέτησης, ο οποίος θα πυροδοτείται κάθε φορά που μεταβάλλεται ένα από τα πεδία των φύλλων που ακο-

λουθούν. Μόλις πυροδοτείται ο παρατηρητής, η αντίστοιχη συνάρτηση εξυπηρέτησης εκτελείται. Στη συγκεκριμένη περίπτωση λοιπόν δημιουργείται ένα φύλλο στο XenStore στο οποίο θα αποθηκευτεί ο αριθμός της σελίδας που πρόκειται να παραχωρηθεί. Το Dom0 τοποθετεί έναν παρατηρητή σε αυτό το φύλλο και προσαρμόζει την κατάλληλη συνάρτηση εξυπηρέτησης. Το DomU ενημερώνει το φύλλο με τον αριθμό της σελίδας που παραχωρεί. Μόλις συμβεί αυτό, ο παρατηρητής ενεργοποιείται και εκτελείται η συνάρτηση εξυπηρέτησης. Μέσα στη συνάρτηση αυτή γίνεται η ανάγνωση του πεδίου και έπειτα η αποδοχή της σελίδας αυτής από το Dom0. Δεν μπορεί να χρησιμοποιηθεί η XenStore για την ανταλλαγή δεδομένων μεταξύ των Doms, δηλαδή δε μπορεί να υποκαταστήσει τον δακτύλιο, αφού στα φύλλα της αποθηκεύονται τιμές περιορισμένου μεγέθους και πάντα είναι τύπου συμβολοσειράς (string). Στη XenStore αποθηκεύονται μόνο λειτουργικές παράμετροι.

Τέλος, για την ενημέρωση και την εγκατάσταση διακοπών τόσο στο Dom0 όσο και στα DomUs δημιουργείται ένα σύστημα ενημέρωσης μεταξύ τους, που αποτελείται από έναν δίαυλο γεγονότων, δύο θύρες (μία σε κάθε άκρο του διαύλου) και δύο συναρτήσεις εξυπηρέτησης (μία σε κάθε θύρα του διαύλου). Στο συγκεκριμένο παράδειγμα, ο Dom0 δημιουργεί μία θύρα (port) στην οποία προσαρμόζει έναν δίαυλο γεγονότων, που η δεύτερη πόρτα δεν είναι ακόμα γνωστή και ενημερώνει το κατάλληλο φύλλο της XenStore με τον αριθμό της θύρας. Επειδή το DomU έχει ήδη τοποθετήσει έναν παρατηρητή στο φύλλο αυτό, ενεργοποιείται η αντίστοιχη συνάρτηση εξυπηρέτησης, στην οποία διαβάζεται το φύλλο αυτό, δημιουργείται η θύρα του DomU (η δεύτερη θύρα του διαύλου) και ενώνεται με τη θύρα του Dom0 μέσω του διαύλου. Όταν, λοιπόν, το DomU τοποθετεί ένα αίτημα στο δακτύλιο, στέλνει ένα γεγονός μέσω του διαύλου στο Dom0. Η συνάρτηση εξυπηρέτησης του γεγονότος ενεργοποιείται και το Dom0 καταναλώνει το αίτημα.

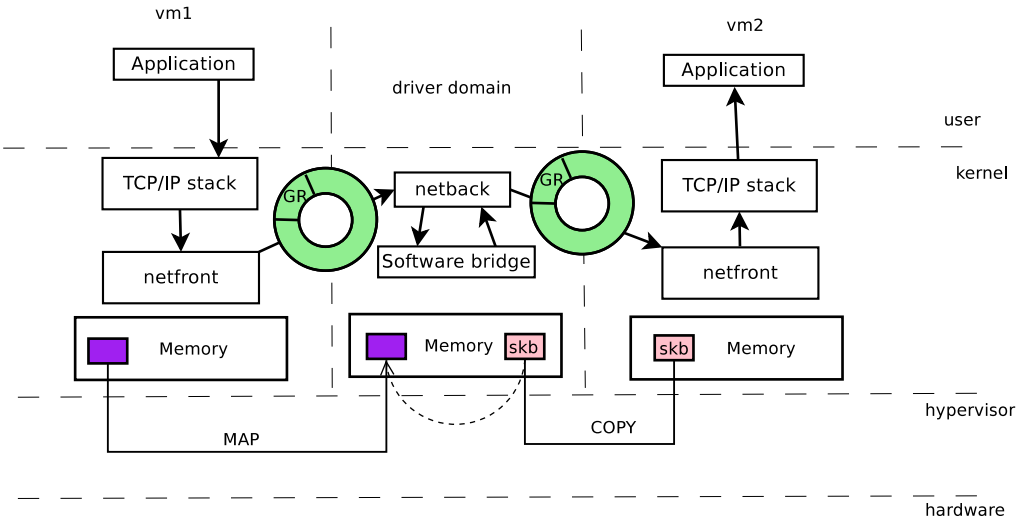
Επίδοση

Ο ελεγκτής εικονικής μηχανής Xen υιοθετεί τη τεχνική του μερικού virtualization. Μπορεί να επιτύχει απόδοση που πλησιάζει αυτή των φυσικών συστημάτων. Αυτό συμβαίνει διότι οι εικονικές μηχανές έχουν επίγνωση ότι εκτελούνται σε εικονικό περιβάλλον και έτσι υπάρχει δυνατότητα να παρακάμψουν επίπεδα που παρεμβάλλονται ανάμεσα σε αυτά και το υλικό. Με άλλα λόγια υπάρχει δυνατότητα άμεσης πρόσβασης στο υλικό, ειδικά αν αυτό υποστηρίζει εικονικά περιβάλλοντα.

Κεφάλαιο 3

Intranode Communication

Σε αυτό το κεφάλαιο παρουσιάζουμε μελέτες που έχουν γίνει στο παρελθόν και στοχεύουν στην βελτιστοποίηση της ενδοεπικοινωνίας εικονικών μηχανών καθώς επίσης και τον κλασσικό τρόπο με τον οποίο διεξάγεται η ενδοεπικοινωνία στο Xen. Το παρόν κεφάλαιο δομείται ως εξής: αρχικά παρουσιάζουμε τον μηχανισμό που παρέχεται από το Xen για την επικοινωνία των εικονικών μηχανών και στη συνέχεια παρουσιάζουμε διάφορες μελέτες που έχουν γίνει και σκοπό έχουν να πετύχουν την ενδοεπικοινωνία με μικρότερο κόστος από άποψη επίδοσης. Οι τεχνικές που έχουν προταθεί χωρίζονται σε τρεις κατηγορίες: την εκμετάλλευση κοινής μνήμης, την τροποποίηση αλγορίθμων χρονοδρομολόγησης ώστε να έχουμε ταχύτερη επικοινωνία και την βελτιστοποίηση της διεπαφής δικτύου.



Σχήμα 3.1: Ενδο-επικοινωνία εικονικών μηχανών στο Xen

3.1 Xen Virtual Network

Η προεπιλεγμένη μέθοδος για την ενδοεπικοινωνία εικονικών μηχανών είναι μέσω της κλασσικής διεπαφής δικτύου. Η υλοποίηση του εικονικού δικτύου του Xen δημιουργεί μία γέφυρα (bridge) πάνω από την φυσική διεπαφή του δικτύου. Για κάθε εικονικό μηχάνημα το Xen δημιουργεί μία εικονική διεπαφή δικτύου και την συνδέει με το bridge. Αυτή η διαδικασία είναι διαφανής προς τα εικονικά μηχανήματα και επιπλέον καθιστά τον Domain 0 υπεύθυνο για την διαχείριση της διεπαφής δικτύου.

Στο Σχήμα 3.1 φαίνεται η διαδρομή που ακολουθεί ένα πακέτο για να μεταδοθεί από τον χώρο χρήστη μιας εικονικής μηχανής στον χώρο χρήστη μιας άλλης. Προτού ξεκινήσει η επικοινωνία δημιουργείται ένα κομμάτι μοιραζόμενης μνήμης ανάμεσα στις εικονικές μηχανές, όπως περιγράψαμε στο προηγούμενο κεφάλαιο(grant table, I/O ring). Αρχικά, όταν η εφαρμογή στείλει το πακέτο, αυτό διασχίζει την στοίβα πρωτοκόλλου TCP/IP. Στο κατώτερο επίπεδο της στοίβας, αντί να βρει την φυσική διεπαφή δικτύου όπως θα έβρισκε αν δεν υπήρχε το εικονικό περιβάλλον, συναντά την εικονική διεπαφή δικτύου, η οποία ουσιαστικά πρόκειται για το front-end του split driver model (Netfront). Το netfront τοποθετεί την αναφορά του grant table (GR), που δείχνει στη σελίδα μνήμης που περιέχονται τα δεδομένα, μέσα στο μοιρζόμενο I/O ring. Έπειτα ειδοποιεί τον Domain 0, μέσω του event channel, να διβάσει το GR. Αφού ο Domain 0 πάρει το GR αντιστοιχίζει τη σελίδα μνήμης, την οποία δείχνει ο GR, σε δικό του χώρο μνήμης και βάζει τα δεδομένα σε μί δομή sk_buf, έτσι ώστε να μπορεί να τα διαχειριστεί η στοίβα δικτύου. Όταν ο Domain 0 καταλάβει ότι ο προορισμός των δεδομένων είναι ένα εικονικό μηχάνημα στην ίδια πλατφόρμα, ο backend driver παίρνει το GR από το I/O ring του δεύτερου εικονικού μηχανήματος, αντιγράφει τα δεδομένα από την δομή sk_buf στο χώρο μνήμης που δείχνει το GR (VM2) και έπειτα ειδοποιεί την δεύτερη εικονική μηχανή για την άφιξη των δεδομένων. Τέλος, αναλαμβάνει το netfront της εικονικής μηχανής να πάρει τα δεδομένα και να τα προωθήσει μέσω της στοίβας TCP/IP στον χώρο χρήστη.

Σύμφωνα με τους Menon et al. [6] έχει τρεις επιβαρύνσεις: α) το κόστος της επεξεργασίας από την στοίβα TCP/IP, β) η επιβάρυνση από την αντιστοίχηση των σελίδων (page flipping) και γ) το μεγάλο μονοπάτι επικοινωνίας. Στη συνέχεια του κεφαλαίου παρουσιάζονται διάφορες μελέτες που στόχο έχουν την βελτιστοποίηση της επίδοσης στην ενδοεπικοινωνία εικονικών μηχανών.

3.2 Σχετική Βιβλιογρφία

3.2.1 Μοιραζόμενη Μνήμη

XenLoop

Το XenLoop [11]βασίζεται σε ένα module του λειτουργικού συστήματος Linux το οποίο ενσωματώνεται στο επίπεδο δικτύου στον πυρήνα κάθε εικονικής μηχανής. Το module διακόπτει την πορεία κάθε εξερχόμενου πακέτου και το εξετάζει μέσω ενός μηχανισμού που προσφέρει το Linux (Linux netfilter hook). Επιπλέον το module περιέχει έναν πίνακα αντιστοίχησης με όλες τις εικονικές μηχανές που τρέχουν στο ίδιο φυσικό μηχάνημα. Στην περίπτωση όπου δύο εικονικά μηχανήματα που φιλοξενούνται στο ίδιο φυσικό μηχάνημα χρειάζεται να ανταλλάξουν δεδομένα, τα modules σε κάθε μηχάνημα φτιάχνουν ένα κανάλι επικοινωνίας διπλής κατεύθυνσης με χρήση κοινής μνήμης, έτσι ώστε να μην εμπλέκεται ο Domain0 στη διαδικασία της επικοινωνίας.

XenSockets

Το XenSockets [14]είναι ένα κανάλι επικοινωνίας μονής κατεύθυνσης ανάμεσα σε δύο εικονικά μηχανήματα. Υλοποιεί έναν μηχανισμό ενδοεπικοινωνίας βασιζόμενο στις αρχές των Unix domain sockets. Η υλοποίηση του XenSockets βασίζεται σε μοιραζόμενη μνήμη και αποφεύγει τον μηχανισμό page flipping του Xen. Ο αποστολέας δημιουργεί μια σελίδα μνήμης και την αντιστοιχίζει στον χώρο διευθύνσεων που έχει πρόσβαση ο αποδέκτης. Κάθε άκρο επικοινωνίας υλοποιεί δύο τύπους μοιραζόμενων σελίδων: μία σελίδα περιγραφής, που χρησιμοποιείται για τον έλεγχο της επικοινωνίας και έναν επιπλέον χώρο κοινής μνήμης που χρησιμοποιείται για την ανταλλαγή των δεδομένων. Η προγραμματιστική διεπαφή του XenSocket είναι μία καθαρή και απλή διεπαφή, ωστόσο οι εφαρμογή πρέπει να επαναμεταγλωττιστεί σύμφωνα με την νέα διεπαφή και έτσι δεν προσφέρεται συμβατότητα.

XWAY

Το XWAY [5] ορίζει μία εικονική συσκευή στο μοντέλο συσκευών του Xen καθώς και έναν οδηγό συσκευών μέσα σε κάθε εικονική μηχανή. Προκειμένου να δημιουργηθεί το κανάλι επικοινωνίας, οι οδηγοί συσκευών σε κάθε εικονική μηχανή πρέπει να δημιουργήσουν έναν χώρο κοινής μνήμης καθώς και ένα event channel. Για την δημιουργία των παραπάνω, το XWAY χρησιμοποιεί τις διεπαφές επικοινωνίας που προσφέρει το Xen: grant tables, xenstore και event

channels, ωστόσο δεν χρησιμοποιεί το page flipping για λόγους επίδοσης. Οι δύο οδηγοί συσκευών του XWAY ανταλλάσσουν τις αναφορές για την μοιραζόμενη μνήμη και για το event channel μέσω ενός ειδικού βοηθητικού daemon που τρέχει σε κάθε εικονική μηχανή. Το XWAY προσφέρει συμβατότητα για όλες τις εφαρμογές που είναι γραμμένες σύμφωνα με την προγραμματιστική διεπαφή Socket. Αυτό επιτρέπει στις εφαρμογές να επωφεληθούν από το γεγονός ότι φιλοξενούνται στο ίδιο φυσικό μηχάνημα, χωρίς να απαιτείται η επαναμεταγλώττισή τους με νέα προγραμματιστική διεπαφή.

Inter-OS Communication on Highly Parallel Multi-Core Architectures

Οι Youseff et al. [13] επίσης στηρίζουν την δουλειά τους στον μηχανισμό των grant tables του Xen, αλλά επιτρέπουν τον διαμοιρασμό μνήμης μεταξύ εικονικών μηχανών διαφορετικών επιπέδων εμπιστοσύνης. Αντί να χρησιμοποιήσουν την κλασσική προγραμματιστική διεπαφή των Socket, Οι συγγραφείς επιλέγουν να κάνουν απευθείας διαθέσιμη την κοινή μνήμη στις εικονικές μηχανές και τις εφαρμογές τους. Αυτή η επιλογή επιβαρύνει λιγότερο την επικοινωνία όμως η ασφάλεια και η αξιοπιστία επαφίεται στους προγραμματιστές. Δυνητικά, αυτή η υλοποίηση είναι ευάλωτη σε επιθέσεις DoS. Μία εφαρμογή θα μπορούσε να εξαντλήσει τους πόρους της πλατφόρμας απαιτώντας όλη τη μνήμη του συστήματος.

Virtual Machine Aware Communication Libraries for High Performance Computing

Η μελέτη των Huang et al. [4] βασίζεται στον διαμοιρασμό μνήμης και υλοποιεί μία προγραμματιστική διεπαφή παρόμοια με αυτή των Socket για το Xen. Κύριος στόχος των συγγραφέων είναι το High Performance Computing (HPC) και για αυτόν τον λόγο υλοποιούν μία βιβλιοθήκη ανταλλαγής μηνυμάτων (Message Passing Interface - MPI). Οι διεπαφές ανταλλαγής μηνυμάτων είναι ευρέως γνωστές στα περιβάλλοντα HPC. Η βιβλιοθήκη που υλοποιείται σε αυτήν τη μελέτη υποστηρίζει την ενδοεπικοινωνία εικονικών μηχανικών και επίσης χρησιμοποιεί την τεχνική VMM-bypass I/O, η είναι προηγούμενη μελέτη των ίδιων συγγραφέων[]. Η διαδικασία ενδοεπικοινωνίας λειτουργεί ως εξής: αρχικά, μια διεργασία χρησιμοποιεί την βιβλιοθήκη για να δεσμεύσει σελίδες μνήμης ως έναν κοινό χώρο με το άλλο άκρο επικοινωνίας. Στην συνέχει η βιβλιοθήκη καλεί τον αντίστοιχο driver για να παραχωρήσει στην εικονική μηχανή όπου στεγάζεται το άλλο άκρο επικοινωνίας, πρόσβαση σε αυτές τις σελίδες. Η κοινή μνήμη δημιουργείται χρησιμοποιώντας τον μηχανισμό των grant tables του Xen. Αφού δημιουργηθεί η κοινή μνήμη, μία διεπαφή

ανάλογη με αυτή των Socket χρησιμοποιείται για να διεξαχθεί η επικοινωνία, το οποίο απαιτεί την επαναμεταγλώττιση των εφαρμογών.

Efficient Shared Memory Passing for Inter-VM Communication

Στην μελέτη [2] εξετάζεται το ζήτημα της ενδοεπικοινωνίας εικονικών μηχανών για MPI εφαρμογές. Για να επιτευχθεί αυτό, εισάγεται μια εικονική συσκευή η οποία προσφέρει μία απλή διεπαφή ανταλλαγής μηνυμάτων στα φιλοξενούμενα λειτουργικά συστήματα. Η εικονική συσκευή ουσιαστικά είναι ένας κοινός χώρος μνήμης μέσω του οποίου γίνεται η επικοινωνία των εικονικών μηχανών. Η διεπαφή της εικονικής συσκευής δεν ορίζει κάποιο πρωτόκολλο επικοινωνίας αλλά αφήνει την δυνατότητα υλοποίησης κάποιου πρωτοκόλλου στην βιβλιοθήκη επικοινωνίας στον χώρο χρήστη. Έπειτα η διεπαφή αυτή χρησιμοποιείται για να υλοποιηθεί μια αποδοτική MPI βιβλιοθήκη για εικονικές μηχανές.

Πειραματική αποτίμηση και βελτιστοποίηση της επικοινωνίας εικονικών μηχανών που συνυπάρχουν στο ίδιο φυσικό μηχάνημα

Στην διπλωματική εργασία “Πειραματική αποτίμηση και βελτιστοποίηση της επικοινωνίας εικονικών μηχανών που συνυπάρχουν στο ίδιο φυσικό μηχάνημα”, ο συγγραφέας αποτιμά την επίδοση της μεταφοράς δεδομένων μεταξύ δύο εικονικών μηχανών στο ίδιο φυσικό μηχάνημα. Για την ανταλλαγή των δεδομένων χρησιμοποιούνται οι μηχανισμοί διαμοιρασμού μνήμης του Xen (grant table, I/O rings) καθώς και η βιβλιοθήκη “libvchan”. Παράλληλα με την πειραματική αποτίμηση της επικοινωνίας των εικονικών μηχανών, γίνεται και η βελτιστοποίηση της επίδοσης της επικοινωνίας αυτής, τροποποιώντας κατάλληλα τμήμα του κώδικα της βιβλιοθήκης libvchan.

3.2.2 Χρονοδρομολόγηση

Xen & Co: Communication-aware CPU Scheduling

Η βασική ιδέα της μελέτης [3] είναι η μεροληπτική χρονοδρομολόγηση των εικονικών μηχανών που επικοινωνούν έντονα έναντι αυτών που εκτελούν κυρίως υπολογισμούς. Όταν ο χρονοδρομολογητής πρέπει να επιλέξει ανάμεσα από πολλές εικονικές μηχανές επιλέγει αυτή με τα περισσότερα εκκρεμή πακέτα - είτε αυτή που έχει λάβει τα περισσότερα πακέτα είτε αυτή που πρόκειται να στείλει τα περισσότερα πακέτα. Στόχος της μελέτης είναι να μειωθεί η συνολική καθυστέρηση στις διαδικασίες Ε/Ε, που επιβάλλεται από την χρο-

νοδρομολόγηση. Επίσης, τίθεται ένα άνω όριο στο time-granularity ώστε να είναι δίκαιος ο διαμοιρασμός των CPUs.

Coexisting scheduling policies boosting I/O Virtual Machines

Στη μελέτη [1] υποστηρίζεται ότι αλλάζοντας την λογική της χρονοδρομολόγησης σε ένα φορτωμένο VM container, βελτιώνεται η συνολική απόδοση του συστήματος. Προτείνεται ένα πλαίσιο το οποίο προσφέρει διάφορες πολιτικές χρονοδρομολόγησης που είναι προσαρμοσμένες στις ανάγκες του workload. Συγκεκριμένα υλοποιείται το ακόλουθο σενάριο: ο Domain 0 δεν τρέχει στις φυσικές CPUs Που τρέχουν τα εικονικά μηχανήματα και δεν διακόπτεται. Επιπρόσθετα, τα εικονικά μηχανήματα οργανώνονται σε ομάδες και τρέχουν σε CPU sets ανάλογα με το workload τους, προσφέροντας απομόνωση και αποδοτική χρησιμοποίηση των πόρων του συστήματος. Η προσέγγιση αυτή πετυχαίνει 2.3 φορές ταχύτερη Ε/Ε.

3.2.3 Βελτιστοποιήσεις Διεπαφών Δικτύου

Xen2MX: High-performance communication in Virtualized Environments

Η διδακτορική διατριβή [8] έχει στόχο την βελτιστοποίηση της διεπαφής δικτύου. Σε αυτή τη μελέτη σχεδιάζεται και υλοποιείται το Xen2MX, ένα πρωτόκολλο ανταλλαγής μηνυμάτων υψηλής επίδοσης για εικονικά περιβάλλοντα. Τα χαρακτηριστικά του Xen2MX ελαχιστοποιούν και, σε περιπτώσεις, εξαλείφουν προβλήματα που σχετίζονται με τους κοινούς οδηγούς συσκευών που χρησιμοποιούνται σε υποδομές cloud στο πλαίσιο του HPC. Πιο συγκεκριμένα, ελαχιστοποιεί την επιβάρυνση του χειρισμού γεγονότων για ανταλλαγή μηνυμάτων και βελτιώνει σημαντικά τη ρυθμαπόδοση με: (a) χρήση τεχνικών μηδενικών αντιγραφών (zero-copy) για μεγάλα μηνύματα, (b) επαναχρησιμοποίηση αντιστοιχίσεων μνήμης μεταξύ εικονικών μηχανών και του ελεγκτή, (c) αποδέσμευση των μηνυμάτων ελέγχου από τη μεταφορά δεδομένων, χάρη στη βελτιστοποιημένη έκδοση του μηχανισμού καταναλωτή–παραγωγού για την επικοινωνία με στρώματα του λειτουργικού συστήματος και του ελεγκτή εικονικών μηχανών. Ο σχεδιασμός του Xen2MX μπορεί να εφαρμοστεί σε οποιοδήποτε ελεγκτή υποστηρίζει δυνατότητες paravirtualization.

MyriXen: Message Passing in Xen Virtual Machines over Myrinet and Ethernet

Στη μελέτη [7] παρουσιάζεται το MyriXen, ένα πλαίσιο με το οποίο οι εικονικές μηχανές μοιράζονται αποδοτικά τις συσκευές δι-

κτύου μειώνοντας τις επιβαρύνσεις που οφείλονται στον ελεγκτή εικονικών μηχανών και τον driver domain. Το MyriXen επιτρέπει στα εικονικά μηχανήματα να ανταλλάσσουν, με βέλτιστο τρόπο, μηνύματα με το δίκτυο μέσω μιας υψηλής επίδοσης NIC, αφήνοντας τα ζητήματα ασφάλειας και απομόνωσης στα επίπεδα εικονοποίησης. Με το MyriXen, πολλές εικονικές μηχανές ανταλλάσσουν μηνύματα, χρησιμοποιώντας το πρωτόκολλο MX πάνω από κάρτα δικτύου Myri-10G, σαν η κάρτα δικτύου να ανήκει αποκλειστικά σε κάθε ένα από αυτά.

A Smart HPC interconnect for clusters of Virtual Machines

Στη μελέτη [9] παρουσιάζεται η σχεδίαση ενός VM-aware, υψηλής επίδοσης δικτύου διασύνδεσης πάνω από κάρτα δικτύου 10Gbps Ethernet. Το πλαίσιο αυτό προσφέρει ένα απευθείας μονοπάτι επικοινωνίας με την κάρτα δικτύου, για εφαρμογές που τρέχουν σε εικονικά περιβάλλοντα, αφήνοντας την διαχείριση των μη κρίσιμων μονοπατιών (όπως αυτό του ελέγχου) στα ενδιάμεσα επίπεδα εικονοποίησης. Αυτή η προσέγγιση επιτρέπει την πολύπλεξη των εικονικών μηχανών στην πρόσβαση του δικτύου. Αυτό το πλαίσιο επιτρέπει στα εικονικά μηχανήματα να επικοινωνούν χρησιμοποιώντας ένα απλό user-level RDMA πρωτόκολλο.

Κεφάλαιο 4

V4Vsockets: Σχεδιασμός και υλοποίηση μηχανισμού ενδοεπικοινωνίας εικονικών μηχανών

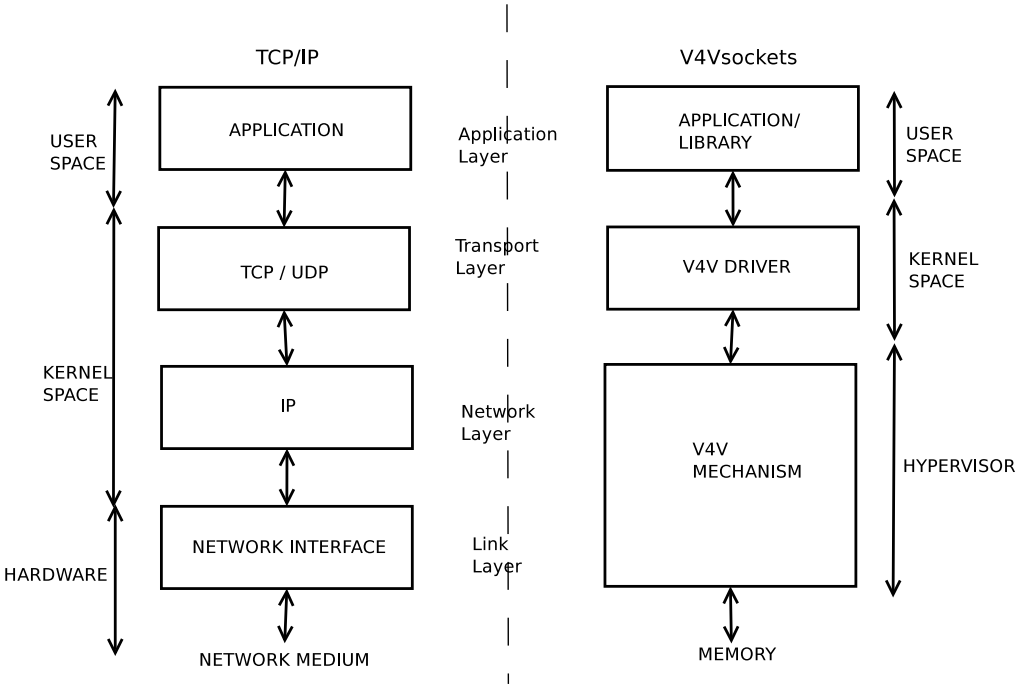
4.1 Εισαγωγή

Σε αυτό το κεφάλαιο περιγράφουμε αναλυτικά το V4Vsockets και τα βασικά στοιχεία που το απαρτίζουν. Ουσιαστικά πρόκειται για ένα πλαίσιο (framework) που επιτρέπει την επικοινωνία μεταξύ εικονικών μηχανών που βρίσκονται στο ίδιο φυσικό μηχάνημα χρησιμοποιώντας ευρέως γνωστά επικοινωνιακά πρωτόκολλα (TCP/IP).

Το V4Vsockets απαρτίζεται από μια βιβλιοθήκη χώρου χρήστη και έναν μηχανισμό ενδοεπικοινωνίας εικονικών μηχανών που ονομάζεται V4V. Το V4V δεν έχει ενσωματωθεί ακόμα στο Xen καθώς είναι υπό εξέλιξη. Ακολουθώντας την σχεδιαστική αρχή διάκρισης μηχανισμού και πολιτικής το V4V αποτελείται από δύο μέρη. Το πρώτο μέρος είναι υλοποιημένο μέσα στον hypervisor και παρέχει έναν μηχανισμό για την μεταφορά δεδομένων από μία εικονική μηχανή σε μία άλλη. Το δεύτερο μέρος είναι υλοποιημένο ως module στον πυρήνα των εικονικών μηχανών και υλοποιεί το ευρέως γνωστό δικτυακό πρωτόκολλο επικοινωνίας TCP/IP(πολιτική).

Όπως φαίνεται και στο παρακάτω σχήμα (Σχήμα **??**, παραλληλίζοντας το V4Vsockets με τα επίπεδα του πρωτοκόλλου TCP/IP θα μπορούσαμε να πούμε ότι το επίπεδο εφαρμογής αποτελείται από την δικτυακή εφαρμογή και την βιβλιοθήκη V4Vsockets, το επίπεδο μεταφοράς αποτελείται από το μέρος του V4V που είναι υλοποιημένο στον

πυρήνα και επί της ουσίας είναι ένας οδηγός χαρακτήρων και τα επίπεδα δικτύου και συνδέσμου αντιστοιχούν στο μέρος του V4V που είναι υλοποιημένο στο Xen.



Σχήμα 4.1: TCP/IP vs V4Vsockets

Στις ενότητες που ακολουθούν παρουσιάζουμε τους βασικούς πυλώνες της σχεδίασης και υλοποίησης του μηχανισμού V4Vsockets.

4.1.1 Επίπεδο Εφαρμογής

Σε αυτό το επίπεδο ανήκουν η εφαρμογή χώρου χρήστη και η βιβλιοθήκη V4Vsockets. Η εφαρμογή χώρου χρήστη μπορεί να είναι οποιοδήποτε πρόγραμμα που χρησιμοποιεί sockets προκειμένου να ανταλλάξει δεδομένα με κάποιο άλλο πρόγραμμα.

Όπως αναφέρεται και παραπάνω μία από τις βασικότερες απαιτήσεις αυτής της μελέτης είναι η διατήρηση της προγραμματιστικής διεπαφής. Συγκεκριμένα, στόχος είναι οποιεσδήποτε εφαρμογές φιλοξενούνται σε εικονικά μηχανήματα που βρίσκονται στο ίδιο φυσικό μηχάνημα και χρησιμοποιούν sockets για την επικοινωνία τους να μπορούν να χρησιμοποιήσουν το V4V χωρίς να απαιτείται κάποια αλλαγή στον πηγαίο κώδικά τους ή η επαναμεταγλώτισή τους.

Η απαίτηση αυτή ικανοποιείται από την βιβλιοθήκη V4Vsockets. Ο ρόλος της βιβλιοθήκης V4Vsockets είναι να φορτώνεται αντί των κλασσικών βιβλιοθηκών που είναι απαραίτητες προκειμένου να χρησιμοποιηθούν τα sockets και να κάνει δυνατή τη χρήση του V4V από

τις εφαρμογές του χώρου χρήστη. Η βιβλιοθήκη αντιστοιχεί τις κλήσεις που γίνονται στα sockets (socket, bind, connect κτλ) στις αντίστοιχες κλήσεις προς το V4V. Επιπλέον, επειδή για την λειτουργία του V4V είναι απαραίτητη η γνώση του domain-id των εικονικών μηχανών η βιβλιοθήκη αναλαμβάνει την “μετάφραση” των IP διευθύνσεων πηγής και προορισμού στα αντίστοιχα domain-id.

Συνοπτικά, μπορούμε να πούμε ότι η προσφορά της βιβλιοθήκης V4Vsockets είναι η συμβατότητα μεταξύ του V4V και της προγραμματιστικής διεπαφής sockets. Δηλαδή, η βιβλιοθήκη αποτελεί τη διαπροσωπία μέσω της οποίας καθίσταται εφικτή η χρήση του μηχανισμού V4V με τρόπο διαφανή προς τον χρήστη χωρίς να απαιτείται τροποποίηση των εφαρμογών.

4.1.2 Επίπεδο Μεταφοράς

Το επίπεδο μεταφοράς στην δική μας προσέγγιση αποτελείται από τον driver του V4V και βρίσκεται στον πυρήνα των εικονικών μηχανημάτων. Εισάγεται ως module και στην πραγματικότητα πρόκειται για έναν οδηγό χαρακτήρων. Πρόκειται για το μέρος του V4Vsockets που υλοποιεί το πρωτόκολλο επικοινωνίας (TCP/UDP) πάνω από το φυσικό μέσο (Xen).

Όσο αφορά το επίπεδο χρήστη δέχεται κλήσεις από την βιβλιοθήκη, τις μεταφράζει και κάνει όλες τις απαραίτητες ενέργειες ώστε να επιτευχθεί η επικοινωνία. Πιο συγκεκριμένα αναλαμβάνει:

- να δημιουργήσει μια εικονική σύνδεση μεταξύ των μηχανημάτων που θέλουν να επικοινωνήσουν
- είναι υπεύθυνο για την αποστολή και λήψη των πακέτων TCP/UDP, το οποίο το επιτυγχάνει μέσω κλήσεων προς τον hypervisor (hypercalls)
- και τέλος ελέγχει και ειδοποιεί τον χώρο χρήστη για τυχόν λάθη κατά την επικοινωνία.

4.1.3 Επίπεδο Δικτύου/Συνδέσμου

Το μέρος του V4V που βρίσκεται μέσα τον hypervisor (hypervisor patch) προσφέρει υπηρεσίες του επιπέδου δικτύου και συνδέσμου. Όπως αναφέρεται στο Κεφάλαιο 2, οι υπηρεσίες που προσφέρονται από το επίπεδο συνδέσμου είναι:

- ο κατακερματισμός των πακέτων μικρότερα,
- η δρομολόγηση των πακέτων στον προορισμό και

- η προσθήκη επικεφαλίδων στα πακέτα έτσι ώστε η πηγή και ο προορισμός των πακέτων να είναι γνωστός.

Το hypervisor patch προσφέρει αυτές τις υπηρεσίες καθώς επίσης και τις υπηρεσίες που παρέχονται από το επίπεδο του Συνδέσμου. Η βασική υπηρεσία αυτού του επιπέδου είναι η μεταφορά των δεδομένων στο φυσικό μέσο. Στην δική μας περίπτωση το φυσικό μέσο είναι η μνήμη του μηχανήματος. Το hypervisor patch διεκπεραιώνει την μεταφορά των δεδομένων αντιγράφοντας δεδομένα από την μνήμη ενός εικονικού μηχανήματος στην μνήμη κάποιου άλλου.

4.2 Υλοποίηση

Σε αυτό το κεφάλαιο παρουσιάζουμε αναλυτικά τον σχεδιασμό και την υλοποίηση του V4Vsockets. Αρχικά παρουσιάζουμε την βασική αρχιτεκτονική του V4V, πάνω στο οποίο στηριζόμαστε και χτίζουμε το V4Vsockets. Στη συνέχεια παραθέτουμε ένα τυπικό παράδειγμα επικοινωνίας τύπου client - server, όπου οι εφαρμογές βρίσκονται σε διαφορετικές εικονικές μηχανές στο ίδιο φυσικό μηχάνημα. Μέσα από το παράδειγμα παρουσιάζονται λεπτομερώς όλες οι ενέργειες που γίνονται από από το χώρο χρήστη μέχρι τον hypervisor και πως τα συστατικά στοιχεία του V4Vsockets (βιβλιοθήκη χώρου χρήστη, driver, hypervisor patch) αλληλεπιδρούν ώστε οι εφαρμογές να μπορούν να επικοινωνήσουν χωρίς να απαιτείται κάποια αλλαγή στον πηγαίο κώδικά τους ή επαναμεταγλώτισή τους.

4.2.1 Αρχιτεκτονική V4V

Το μέρος του V4V που είναι υλοποιημένο μέσα στο Xen (hypervisor patch) παρέχει τον μηχανισμό για την μεταφορά δεδομένων από μία εικονική μηχανή σε μία άλλη. Στο V4V η διεξαγωγή της επικοινωνίας βασίζεται στην ανταλλαγή μηνυμάτων μέσω αντιγραφής μνήμης, σε αντίθεση με τις μεθόδους που απαντώνται στην σχετική βιβλιογραφία οι οποίες στην πλειοψηφία τους χρησιμοποιούν κοινή μνήμη. Επιπλέον, η διεξαγωγή της επικοινωνίας γίνεται μέσω του hypervisor, ο οποίος είναι υπεύθυνος για την μεταφορά των δεδομένων και για την ενημέρωση των εικονικών μηχανών ότι υπάρχουν νέα δεδομένα.

Η βασική δομή που χρησιμοποιεί το V4V για την ανταλλαγή των μηνυμάτων είναι κυκλικοί απομονωτές (ring buffers, στο εξής v4v_ring), οι οποίοι υπάρχουν στο χώρο μνήμης που βλέπουν οι πυρήνες των εικονικών μηχανών. Η δομή v4v_ring, όπως φαίνεται και παρακάτω, περιέχει τόσο το ίδιο το ring, δηλαδή τον χώρο στον οποίο υπάρχουν τα δεδομένα προς αποστολή, όσο και άλλα πεδία που μας δίνουν πληροφορίες για το ring. Τα σημαντικότερα από αυτά τα πεδία είναι η δομή v4v_ring_id_t που λειτουργεί ως η ταυτότητα του

v4v_ring και αποτελεί το αναγνωριστικό του μέσα την πλατφόρμα, το μέγεθος του ring, καθώς και δύο δείκτες οι tx_ptr και rx_ptr. Ο δείκτης tx_ptr δείχνει μέχρι που έχουν γραφτεί δεδομένα μέσα στο ring και ο δείκτης rx_ptr μέχρι που έχουν διαβαστεί. Ο tx_ptr μεταβάλλεται μόνο από τον hypervisor, γιατί μόνο αυτός μπορεί να εισάγει δεδομένα στο ring και ο rx_ptr μεταβάλλεται μόνο από τον v4v driver που βρίσκεται στον πυρήνα των εικονικών μηχανών, γιατί μέσω του driver γίνεται η ανάγνωση των δεδομένων.

```
typedef struct v4v_ring
{
    uint64_t magic;
    struct v4v_ring_id id;
    uint32_t len;
    uint32_t rx_ptr;
    uint32_t tx_ptr;
    uint64_t reserved[4];
    uint8_t ring[0];
} v4v_ring_t;
```

Στο V4V ένα άκρο επικοινωνίας καθορίζεται από τη δομή v4v_addr_t η οποία αποτελείται από ένα port number και το domain id του εικονικού μηχανήματος. Η δομή v4v_ring_id_t αναγνωρίζει μοναδικά τα v4v_ring χρησιμοποιώντας την διεύθυνση ενός άκρου επικοινωνίας (v4v_addr_t) και το domain id του εικονικού μηχανήματος με το οποίο πρόκειται να επικοινωνήσει.

```
typedef struct v4v_addr
{
     uint32_t port;
     domid_t domain;
} v4v_addr_t;

typedef struct v4v_ring_id
{
    struct v4v_addr_t addr;
    domid_t partner;
}v4v_ring_id_t;
```

Επειδή η επικοινωνία γίνεται μέσω του Hypervisor έχει αναπτυχθεί ένα νέο Hypercall το οποίο παρέχει υπηρεσίες προς τον v4v driver. Αυτές οι υπηρεσίες είναι:

- η δήλωση (register) ενός v4v_ring προς το Xen
- η αφαίρεση (Unregister) ενός v4v_ring από το Xen
- η μεταφορά δεδομένων από μία εικονική μηχανή σε μία άλλη και
- η ενημέρωση των εικονικών μηχανών σχετικά με την κατάσταση του κάθε v4v_ring.

Συνοπτικά η διαδικασία αποστολής και λήψης δεδομένων μέσω του V4V έχει ως εξής: Κάθε εικονική μηχανή που επιθυμεί να επικοινωνήσει με κάποια άλλη οφείλει να δηλώσει ένα v4v_ring στο Xen καθώς και να δημιουργήσει ένα event channel μεταξύ αυτού και του Xen. Το event channel είναι απαραίτητο γιατί μέσω αυτού ο Hypervisor ειδοποιεί τις εικονικές μηχανές που φιλοξενεί σχετικά με την διαθεσιμότητα καινούριων δεδομένων. Στη συνέχεια, ο αποστολέας ζητάει από το Xen να μεταφέρει τα δεδομένα στο ring του αποδέκτη. Το Xen αντιγράφει τα δεδομένα από το ring του αποστολέα στο ring του αποδέκτη και αφού ολοκληρώσει την μεταφορά ειδοποιεί τον αποδέκτη μέσω του μηχανισμού των event channels. Τέλος ο αποδέκτης μπορεί να διαβάσει τα δεδομένα από το ring του.

Στο επίπεδο του v4v_driver χρησιμοποιούνται δύο επιπλέον δομές για την διαχείριση των rings. Η δομή v4v_private και η δομή ring.

```
struct ring {
        struct list_head node;
        atomic_t refcnt;
        spinlock_t lock;
        struct list_head privates;
        struct v4v_private *sponsor;
        v4v_rtype type;
        v4v_ring_t *ring;
        v4v_pfn_t *pfn_list;
        size_t pfn_list_npages;
        int order;
};

struct v4v_private {
        struct list_head node;
        v4v_state state;
        v4v_ptype ptype;
        uint32_t desired_ring_size;
        struct ring *r;
        wait_queue_head_t readq;
        wait_queue_head_t writeq;
        v4v_addr_t peer;
        uint32_t conid;
        spinlock_t pending_recv_lock;
        struct list_head pending_recv_list;
        atomic_t pending_recv_count;
        int pending_error;
        int full;
        int send_blocked;
        int rx;
};
```

Κάθε v4v_ring σχετίζεται με μία δομή ring και μία δομή v4v_private. Στη δομή ring περιέχεται ένας δείκτης προς το πραγματικό v4v_ring, μία λίστα όπου αποθηκεύονται οι ψευδοφυσικές σελίδες

που καταλαμβάνει το v4v_ring (pfn_list), ο αριθμός αυτών των σελίδων (pfn_list_npages) και ένα πεδίο τύπου v4v_rtype (type). Αυτό το πεδίο υπάρχει επειδή ο driver υλοποιεί τα πρωτόκολλα TCP και UDP και χρησιμοποιείται για να διακρίνει τις εξής περιπτώσεις:

- το ring χρησιμοποιείται για datagram επικοινωνία (V4V_RTYPE_DGRAM)
- το ring χρησιμοποιείται για stream επικοινωνία και μπορεί να δεχτεί εισερχόμενες συνδέσεις, δηλαδή στην ορολογία των sockets πρόκειται για παθητικό socket (V4V_RTYPE_LISTENER) και
- το ring χρησιμοποιείται για stream επικοινωνία και μπορεί να κάνει αίτηση για συνδέσεις, δηλαδή στην ορολογία των sockets πρόκειται για ενεργό socket (V4V_RTYPE_CONNECTOR)

Όσο αφορά τη δομή v4v_private το πεδίο state δηλώνει την κατάσταση του ring και σημασιολογικά αναφέρεται στις διάφορες καταστάσεις που μπορεί να βρίσκεται ένα socket. Οι τιμές που μπορεί να πάρει είναι οι εξής:

- V4V_STATE_IDLE είναι η αρχική κατάσταση κάθε ring
- V4V_STATE_BOUND σε αυτήν την κατάσταση μεταβαίνει το ring όταν εκτελεστεί επιτυχώς η κλήση bind() και σημαίνει ότι το ring έχει αντιστοιχηθεί με μία συγκεκριμένη διεύθυνση επικοινωνίας
- V4V_STATE_LISTENING το ring περιμένει να δεχτεί κάποια εισερχόμενη αίτηση για σύνδεση
- V4V_STATE_ACCEPTED το ring δέχτηκε κάποια αίτηση για σύνδεση και την αποδέχτηκε
- V4V_STATE_CONNECTING αυτή η κατάσταση σημαίνει ότι είναι υπό εξέλιξη η αίτηση για σύνδεση με το άλλο άκρο επικοινωνίας
- V4V_STATE_CONNECTED σε αυτή τη κατάσταση έχει επιτευχθεί η σύνδεση μεταξύ δύο άκρων επικοινωίας
- V4V_STATE_DISCONNECTED σε αυτήν την κατάσταση δεν υπάρχει σύνδεση

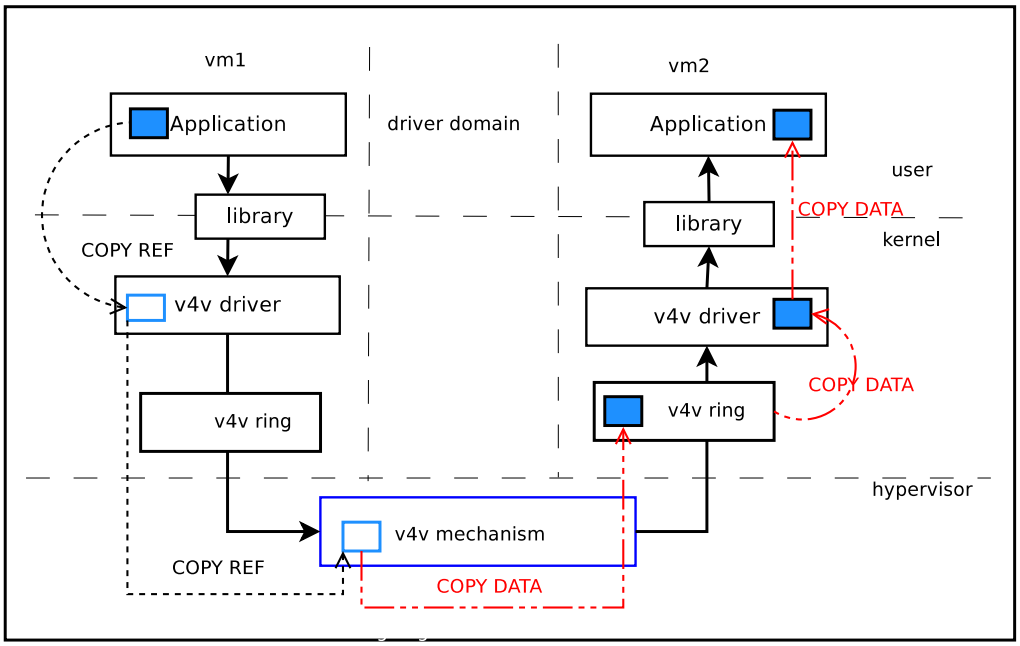
Το πεδίο ptype κάνει διάκριση μεταξύ stream και datagram επικοινωνίας, το πεδίο desired_ring_size περιέχει το επιθυμητό μέγεθος ring που όρισε ο χρήστης, το πεδίο peer αναφέρεται στην διεύθυνση του άλλου άκρου επικοινωνίας, ενώ το πεδίο conid χρησιμοποιείται

στην περίπτωση της stream επικοινωνίας και αποτελεί τον διακριτικό αριθμό της σύνδεσης. Τέλος οι ουρές writeq και readq είναι ουρές αναμονής, έχουν σημαντικό ρόλο κατά την διεξαγωγή της επικοινωνίας και ο τρόπος και ο λόγος που χρησιμοποιούνται θα φανεί παρακάτω.

4.2.2 Παράδειγμα client - server

Σε αυτή την ενότητα παραθέτουμε ένα τυπικό παράδειγμα επικοινωνίας δύο εφαρμογών που φιλοξενούνται σε διαφορετικές εικονικές μηχανές μέσα στο ίδιο φυσικό μηχάνημα. Στο παράδειγμα χρησιμοποιούμε την ευρέως γνωστή προγραμματιστική διεπαφή socket και δείχνουμε πως το V4Vsockets χρησιμοποιεί το V4V προσφέροντας τη δυνατότητα στις εφαρμογές να τρέχουν χωρίς αλλαγές. Το V4Vsockets μας επιτρέπει να χρησιμοποιούμε τόσο stream sockets όσο και dgram sockets. Επειδή στα περισσότερα σημεία οι διεργασίες που επιτελούνται είναι κοινές και για τους δύο τύπους sockets, περιγράφουμε από κοινού και τις δύο περιπτώσεις αναφέροντας ρητά τα σημεία όπου υπάρχει διαφοροποίηση.

Στο παρακάτω σχήμα φαίνεται το μονοπάτι των δεδομένων του V4Vsockets για stream επικοινωνία (Σχήμα 4.2).



Σχήμα 4.2: Μονοπάτι δεδομένων για V4Vsockets stream

Όταν γνωρίζουμε ότι η επικοινωνία λαμβάνει χώρα μεταξύ εικονικών μηχανών που φιλοξενούνται στο ίδιο φυσικό μηχάνημα και

επιθυμούμε να χρησιμοποιήσουμε το V4V αντί της κλασσικής υλοποίησης των socket που παρέχεται από την πλατφόρμα, κάνουμε preload την βιβλιοθήκη V4Vsockets. Με αυτόν τον τρόπο όλες οι κλήσεις των socket οδηγούνται προς τον v4v driver, ο οποίος με τη σειρά του καλεί τον hypervisor και τελικά επιτυγχάνεται η επικοινωνία μέσω αντιγραφής μνήμης και παρακάμπτοντας τη στοίβα του TCP/IP.

Socket

Η πρώτη κλήση που γίνεται από οποιαδήποτε εφαρμογή χρησιμοποιεί την προγραμματιστική διεπαφή Socket είναι η κλήση συστήματος:

```
int socket(int domain, int type, int protocol);
```

Βιβλιοθήκη χώρου χρήστη: Η κλήση socket στο επίπεδο της εφαρμογής αντιστοιχεί στην κλήση open προς τον v4v driver. Σε αυτό το σημείο το πεδίο domain μας είναι αδιάφορο γιατί γνωρίζουμε ότι τα εικονικά μηχανήματα βρίσκονται στο ίδιο φυσικό μηχάνημα όπως επίσης ότι θα χρησιμοποιηθεί το V4V. Το πεδίο type παραμένει ως έχει δηλαδή SOCK_STREAM ή SOCK_DGRAM και λαμβάνεται υπόψη από την βιβλιοθήκη ώστε να κάνει την ανάλογη κλήση προς τον driver. Όσο αφορά το πεδίο protocol παραβιάζουμε την σημασιολογία του και δίνουμε την δυνατότητα στον προγραμματιστή να το χρησιμοποιήσει για να δηλώσει το επιθυμητό μέγεθος του v4v_ring. Σε περίπτωση που ο χρήστης έχει χρησιμοποιήσει το πεδίο protocol για να θέσει το επιθυμητό μέγεθος του ring η βιβλιοθήκη κάνει κλήση προς τον driver μέσω ioctl. Τέλος, η βιβλιοθήκη καλεί τον v4v driver κάνοντας open().

V4V Driver - Kernel Space: Η κλήση open() ανοίγει την συσκευή χαρακτήρων και επί της ουσίας επιστρέφει στον εφαρμογή χρήστη τον περιγραφητή αρχείου (fd), τον οποίο θα χρησιμοποιήσει στην συνέχεια για τις περαιτέρω κλήσεις συστήματός. Ο driver στο εσωτερικό του κρατάει μια συνδεδεμένη λίστα από δομές v4v_private που αντιστοιχούν στα v4v_ring που ανήκουν στην εικονική μηχανή. Κατά την κλήση open ο driver προσθέτει έναν κόμβο στην λίστα με τα v4v_private και αρχικοποιεί τα πεδία της.

Ανάλογα με τη παράμετρο type της κλήσης socket ανοίγεται διαφορετική συσκευή χαρακτήρων. Στο επίπεδο όμως των rings αυτή η διάκριση δεν υπάρχει. Ο driver γνωρίζει για τον τύπο της επικοινωνίας που προορίζεται το κάθε ring μέσω του πεδίου ptype της δομής v4v_private το οποίο το αρχικοποιεί ανάλογα με την περίπτωση. Επίσης σε αυτό το σημείο τίθεται και το πεδίο state της δομής v4v_private σε κατάσταση V4V_STATE_IDLE.

Στην περίπτωση που ο χρήστης επιθυμεί κάποιο συγκεκριμένο μέγεθος για το ring γίνεται η αντίστοιχη κλήση μέσω ioctl και το μέγεθος του ring τίθεται στην τιμή που έχει επιλέξει ο χρήστης. Στην

περίπτωση που χρήστης δεν δηλώσει κάποια συγκεκριμένη τιμή για το μέγεθος του ring τίθεται στην προκαθορισμένη τιμή που ισούται περίπου με 32 σελίδες μνήμης.

Bind

Μετά από την κλήση socket() ακολουθεί η κλήση bind() ώστε να δεθεί η το socket με κάποια διεύθυνση επικοινωνίας. Η εφαρμογή του χρήστη εκτελεί κανονικά την κλήση:

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Βιβλιοθήκη χώρου χρήστη: Στα πλαίσια του V4V το δέσιμο του socket με κάποια διεύθυνση έχει την έννοια της εύρεσης του partner για το v4v_ring, δηλαδή η εύρεση του άλλου άκρου επικοινωνίας. Οι πληροφορίες που χρειάζεται το V4V για να το κάνει αυτό είναι το domain id του εικονικού μηχανήματος στο οποίο φιλοξενείται η εφαρμογή με την οποία θα διεξαχθεί η επικοινωνία καθώς και το port number. Το port number δίνεται ως παράμετρος από την ίδια την εφαρμογή χρήστη. Όσο αφορά το domain id του partner έχουμε υλοποιήσει μέσα στην βιβλιοθήκη μία μικρή συνάρτηση που αντιστοιχίζει την IP address που είναι διαθέσιμη μέσω της εφαρμογής στο domain id της εικονικής μηχανής. Οπότε τελικά η βιβλιοθήκη εκτελεί μια ioctl κλήση προς τον v4v driver με την οποία ζητάει να κάνει bind.

V4V Driver - Kernel Space: Κατά την διαδικασία της bind() ο driver φτιάχνει το καινούριο v4v_ring, δηλαδή δεσμεύει τον απαραίτητο χώρο μνήμης και αρχικοποιεί όλα τα πεδία της δομής v4v_ring. Όσο αφορά την δέσμευση μνήμης, εκτός από το ring, ο driver σε αυτό το σημείο δεσμεύει και μία λίστα με τις ψευδοφυσικές σελίδες μνήμης (pfn_list) από τις οποίες αποτελείται το ring. Στη συνέχεια μεταβάλλει το state του ring σε V4V_STATE_BOUND και τέλος κάνει ένα Hypercall για να δηλώσει (register) το ring στον hypervisor. Τα ορίσματα που περνιούνται στον Hypervisor μέσω του hypercall δείκτες που αναφέρονται στο v4v_ring που μόλις δημιουργήθηκε, την λίστα με τις ψευδοφυσικές σελίδες μνήμης καθώς και ο αριθμός αυτών των σελίδων.

Επίπεδο hypervisor: Ο Hypervisor αποθηκεύει πληροφορίες για όλα τα rings που υπάρχουν στην πλατφόρμα σε μία συνδεδεμένη λίστα από δομές v4v_ring_info_t. Στην πραγματικότητα δεν φτιάχνεται κάποιο ring στο επίπεδο του hypervisor. Τα ring δημιουργούνται και υπάρχουν στους πυρήνες των εικονικών μηχανημάτων. Όπως φαίνεται και από τη δομή v4v_ring_info_t, ο hypervisor κρατάει κάποιες πληροφορίες σχετικά με τα rings, όπως την ταυτότητά τους (v4v_ring_id_t), έναν δείκτη προς το ίδιο το ring, το μέγεθός τους, τον αριθμό των σελίδων που καταλαμβάνουν, τον δείκτη tx_ptr, τις φυσικές σελίδες που καταλαμβάνουν (mfns) καθώς και έναν πίνακα που

περιέχει την αντιστοίχηση μεταξύ φυσικών και ψευδοφυσικών σελίδων μνήμης.

```
struct v4v_ring_info_t
{
    struct hlist_node node;
    v4v_ring_id_t id;
    spinlock_t lock;
    uint32_t len;
    uint32_t npage;
    uint32_t tx_ptr;
    XEN_GUEST_HANDLE(v4v_ring_t) ring;
    uint8_t **mfn_mapping;
    mfn_t *mfns;
    struct hlist_head pending;
};
```

Σε αυτό το σημείο γίνεται η δήλωση του v4v_ring στον hypervisor. Επί της ουσίας αφού γίνουν όλοι οι απαραίτητοι έλεγχοι από το Xen σχετικά με τις παραμέτρους που δόθηκαν από τον driver, δεσμεύεται μία δομή v4v_ring_info_t, αρχικοποιούνται τα πεδία της και γίνεται η αντιστοίχηση μεταξύ των φυσικών και ψευδοφυσικών σελίδων. Αυτή η διαδικασία είναι αρκετά σημαντική γιατί μέσω αυτής της αντιστοίχησης γνωρίζει το Xen γνωρίζει ποιες σελίδες πρέπει να αντιγράψει κατά την μεταφορά των δεδομένων.

Listen

Βιβλιοθήκη χώρου χρήστη: Η κλήση listen() χρησιμοποιείται μόνο σε περίπτωση stream socket. Η εφαρμογή χρήστη εκτελεί την προκαθορισμένη κλήση:

```
int listen(int sockfd, int backlog);
```

Το όρισμα backlog είναι αδιάφορο για το V4Vsockets οπότε η βιβλιοθήκη εκτελεί απλώς ένα ioctl προς τον v4v driver για να εκτελεστεί η listen().

V4V Driver - Kernel Space: Η κλήση listen είναι η πιο απλή κλήση. Επί της ουσίας το μόνο που κάνει είναι να ελέγξει αν το συγκεκριμένο v4v_ring για το οποίο καλείται προορίζεται για stream επικοινωνία καθώς επίσης και την κατάστασή του. Για να εκτελεστεί επιτυχώς, η κατάσταση του v4v_ring πρέπει να είναι V4V_STATE_BOUND, δηλαδή απαιτείται να έχει εκτελεστεί προηγουμένως η bind(). Στη συνέχεια αλλάζει το type του ring σε V4V_RTYPE_LISTENER και την κατάστασή του σε V4V_STATE_LISTENING και έπειτα επιστρέφει.

Επίπεδο hypervisor: Κατά την διάρκεια αυτής της κλήσης δεν γίνεται κάποιο hypercall. Ο λόγος είναι ότι ο v4v driver υλοποιεί αυτήν την κλήση καθώς και τις άλλες προκειμένου να υλοποιήσει το

πρωτόκολλο TCP/IP και να είναι συνεπείς ως προς την προγραμματιστική διεπαφή Socket, δηλαδή υλοποιεί μια πολιτική. Όπως έχουμε αναφέρει και παραπάνω, ο hypervisor παρέχει μόνο έναν μηχανισμό μεταφοράς δεδομένων, οπότε δεν υπάρχει ανάγκη να γνωρίζει τι είδους πρωτόκολλο υλοποιεί ο driver και κατά συνέπεια δεν χρειάζεται να γίνει κάποιο hypercall.

Accept

Βιβλιοθήκη χώρου χρήστη: Όπως συμβαίνει και με τη κλήση listen() έτσι και η κλήση accept() έχει νόημα μόνο στο πλαίσιο stream επικοινωνίας. Η εφαρμογή εκτελεί την προκαθορισμένη από την προγραμματιστική διεπαφή Socket, κλήση

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

Όπως αναφέρεται και στο Κεφάλαιο 2, στα ορίσματα addr και addrlen επιστρέφονται πληροφορίες σχετικά με τη διεύθυνση του peer. Οπότε στο επίπεδο της βιβλιοθήκης μας είναι αδιάφορα και εκτελείται το αντίστοιχο ioctl προς τον v4v driver ο οποίος υλοποιεί την σημασιολογία της accept(). Εκτός από τον file descriptor, sockfd, που περνιέται στον driver μέσω της ioctl, περνιέται επίσης και ένα δεύτερο όρισμα που έχει τον ρόλο των ορισμάτων addr και addrlen. Πρόκειται για έναν δείκτη, *peer, σε μία δομή τύπου v4v_addr_t όπου επιστρέφεται η διεύθυνση του peer (πρόκειται ουσιαστικά για την διεύθυνση του “γειτονικού” v4v_ring - του άλλου άκρου επικοινωνίας), ακριβώς όπως συμβαίνει και με τα συμβατικά sockets.

V4V Driver - Kernel Space: Στο επίπεδο του driver αρχικά γίνεται έλεγχος αν πρόκειται για stream επικοινωνία και αν η κατάσταση του v4v_ring είναι V4V_STATE_LISTENING. Αυτές είναι οι δύο συνθήκες που πρέπει να ικανοποιούνται ώστε να μπορεί να εκτελεστεί η accept(). Στη συνέχεια η κλήση μπλοκάρει μέχρις ότου φτάσει κάποια αίτηση για σύνδεση.

Όπως αναφέρεται και παραπάνω το σημαντικό στοιχείο για την accept() είναι ότι δημιουργεί ένα νέο socket. Ανάλογα στην δική μας περίπτωση, όταν έρθει κάποια αίτηση για σύνδεση, η accept() δημιουργεί μία επιπλέον δομή v4v_private και την προσθέτει στην λίστα με τις δομές v4v_private. Κάνει όλες τις απαραίτητες αρχικοποιήσεις για την δομή αυτή, όπως συμβαίνει στην κλήση socket, με τη διαφορά ότι πεδίο r δείχνει στο ήδη υπάρχον ring, δηλαδή δεν δημιουργείται κάποιο νέο ring και η κατάσταση τίθεται σε V4V_STATE_ACCEPTED. Επιπλέον, η accept() ανοίγει και ένα νέο αρχείο. Ο file descriptor αυτού του αρχείου επιστρέφεται τελικά στην εφαρμογή. Από εδώ και στο εξής όταν η εφαρμογή επιχειρεί να στείλει ή να λάβει δεδομένα το κάνει μέσω του νέου file descriptor. Η accept(), επίσης, γράφει στη μνήμη όπου δείχνει ο δείκτης *peer, που περάστηκε ως όρισμα από τον χώρο χρήστη, και με αυτόν τον τρόπο δίνει πληροφορίες στον

χώρο χρήστη σχετικά με την διεύθυνση του άλλου άκρου επικοινωνίας.

Η τελευταία διαδικασία που επιτελεί η accept() προτού επιστρέψει είναι να στείλει στον peer ένα μήνυμα γνωστοποίησης, ότι δηλαδή έλαβε την αίτηση του για σύνδεση και ότι την αποδέχτηκε. Για να γίνει η αποστολή της γνωστοποίησης χρησιμοποιείται ο μηχανισμός που παρέχεται από το V4V, ο οποίος εξηγείται αναλυτικά σε επόμενη ενότητα.

Επίπεδο hypervisor: Όπως συμβαίνει και με την listen(), η κλήση accept() υλοποιείται επειδή είναι απαίτηση της προγραμματιστικής διεπαφής Socket και ως εκ τούτου δεν αφορά τον Hypervisor. Ο hypervisor εμπλέκεται μόνο για σταλθεί το μήνυμα γνωστοποίησης στον peer, αλλά πέρα από αυτό δεν απαιτείται να υλοποιήσει οτιδήποτε προκειμένου να επιτευχθεί η σωστή λειτουργία της accept().

Connect

Βιβλιοθήκη χώρου χρήστη: Η κλήση connect() αν και συνήθως χρησιμοποιείται όταν έχουμε stream sockets, μπορεί να χρησιμοποιηθεί και στα πλαίσια των datagram sockets. Η εφαρμογή χώρου χρήστη εκτελεί την κανονικά την κλήση

```
int connect(int sockfd, struct sockaddr *addr, socklen_t addrlen);
```

Η βιβλιοθήκη χρησιμοποιεί τις απαραίτητες πληροφορίες από την δομή addr και παραμετροποιεί την αντίστοιχη δομή v4v_addr_t. Συγκεκριμένα χρησιμοποιεί το port number καθώς και την διεύθυνση του peer, που εν προκειμένω είναι το id της εικονικής μηχανής στην οποία φιλοξενείται η εφαρμογή με την οποία θα γίνει η επικοινωνία, και στην συνέχεια εκτελεί ioctl προς τον v4v driver.

V4V Driver - Kernel Space: Στην περίπτωση που η connect καλείται για datagram socket το μόνο που αλλάζει είναι η κατάσταση του ring σε V4V_RING_CONNECTED και επίσης στο πεδίο peer της αντίστοιχης δομής v4v_private αντιγράφεται η διεύθυνση του peer.

Στην περίπτωση των stream sockets η διαδικασία είναι λίγο πιο πολύπλοκη. Αρχικά το type του ring παίρνει την τιμή V4V_RTYPE_CONNECTOR, η κατάστασή του γίνεται V4V_STATE_CONNECTING, αντιγράφεται στο πεδίο peer η διεύθυνση του peer και στη συνέχεια δημιουργείται ο stream_header. Ο stream_header είναι μία δομή που χρησιμοποιείται όταν έχουμε stream επικοινωνία.

```
struct v4v_stream_header
{
    uint32_t flags;
    uint32_t conid;
};
```

Όοως φαίνεται, περιέχει τον διακριτικό αριθμό της σύνδεσης καθώς και μία μεταβλητή flags η οποία λαμβάνει μη μηδενική τιμή κατά τη

διάρκεια εγκαθίδρυσης της σύνδεσης μεταξύτ ων δύο άκρων επικοινωνίας.

Συνεχίζοντας την περιγραφή της κλήσης, σε αυτό το σημείο απαιτείται η δημιουργία ενός αναγνωριστικού για την σύνδεση (sh.conid) και το πεδίο flag του stream header τίθεται στην τιμή V4V_SHF_SYN. Αυτό σημαίνει ότι ο client αιτείται στον server σύνδεση. Ακολούθως στέλνεται ο stream header ως μήνυμα στην άλλη εφαρμογή που έχει τον ρόλο του server, μέσω του μηχανισμού V4V. Στη συνέχεια, η connect() μπλοκάρει έως ότου καταφτάσει το μήνυμα γνωστοποίησης από τον server (accept()). Μόλις ο client λάβει την γνωστοποίηση τίθεται η κατάσταση σε V4V_STATE_CONNECTED και η connect() επιστρέφει επιτυχώς. Από αυτό το σημείο και εμπρός μπορεί να διεξαχθεί η stream επικοινωνία.

Επίπεδο hypervisor: Όπως συμβαίνει με τις accept() και listen(), έτσι και στην connect() δεν απαιτείται η εμπλοκή του hypervisor στην υλοποίησή της. Η connect() χρησιμοποιεί τον hypervisor μόνο για να στείλει στον server την αίτηση για σύνδεση.

Αποστολή Δεδομένων

Βιβλιοθήκη χώρου χρήστη: Η αποστολή δεδομένων στην προγραμματιστική διεπαφή Socket μπορεί να γίνει με την βοήθεια αρκετών συναρτήσεων, όπως send(), sendto(), sendmsg() ανάλογα με τις εκάστοτε ανάγκες. Επίσης η αποστολή δεδομένων μπορεί να γίνει και μέσω της κλήσης συστήματος write(). Στην βιβλιοθήκη υποστηρίζεται προς το παρόν μόνο η κλήση sendto(). Για την κλήση write() δεν χρειάζεται να παρέχουμε υποστήριξη σε επίπεδο βιβλιοθήκης. Επειδή ο v4v driver πρόκειται για οδηγό χαρακτήρων και η write() είναι μία από τις βασικές file operations, όταν η εφαρμογή καλεί την συγκεκριμένη συνάρτηση, η κλήση προωθείται βάση του file descriptor στον driver χωρίς να απαιτείται η παρεμβολή της βιβλιοθήκης. Όσο αφορά την sendto() ο τρόπος κλήσης της είναι

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen);
```

Το όρισμα buf πρόκειται για το μήνυμα που θα σταλθεί, το όρισμα len αναφέρεται στο μέγεθος του μηνύματος, το flag προσδιορίζει διάφορες παραμέτρους τις επικοινωνίας που στην περίπτωση του v4v είναι αδιάφορες και τα ορίσματα dest_addr και addrlen αναφέρονται στην διεύθυνση του παραλήπτη. Σε αυτό το σημείο η βιβλιοθήκη παίρνει τις παραμέτρους sockfd, buf, και len και εκτελεί την write().

```
ssize_t write(int sockfd, const void *buf, size_t len);
```

Από εδώ και πέρα αναλαμβάνει ο v4v driver και ο Hypervisor την αποστολή των δεδομένων στον προορισμό τους.

V4V Driver - Kernel Space: Σε αυτό το σημείο γίνεται διαχωρισμός μεταξύ stream και datagram επικοινωνίας. Αν και η διαδικασία η οποία ακολουθείται είναι σε γενικές γραμμές ίδια, κάποιες μικρές διαφορές που πηγάζουν από τα διαφορετικά πρωτόκολλα επιβάλουν το διαφορετικό χειρισμό των stream και datagram πακέτων.

Όπως έχουμε αναφέρει η τελική αποστολή των μηνυμάτων γίνεται με την μεσολάβηση του hypervisor. Προτού προχωρήσουμε στην αναλυτική περιγραφή της διαδικασίας αποστολής, περιγράφουμε τις δομές που πρέπει να φτιάξει ο driver και οι οποίες θα περαστούν μέσω του Hypercall στον Hypervisor, καθώς είναι ίδιες είτε πρόκειται για stream είτε για datagram επικοινωνία.

Ο driver αποστέλλει τα δεδομένα υπό την μορφή μίας λίστας από δομής v4v_iov_t

```
typedef struct v4v_iov
{
    uint64_t iov_base;
    uint32_t iov_len;
    uint32_t pad;
} v4v_iov_t;
```

Το πεδίο iov_base δείχνει στην διεύθυνση που αρχίζει το μήνυμα και το πεδίο iov_len αναφέρεται στο μέγεθος του μηνύματος. Οπότε ο δείκτης buf που δόθηκε ως παράμετρος από τον χρήστη αντιγράφεται στο πεδίο iov_base και το len στο πεδίο iov_len. Επιπλέον, για το Hypercall απαιτούνται και οι διευθύνσεις παραλήπτη και αποστολέα. Για τον σκοπό αυτό χρησιμοποιείται η δομή v4v_send_addr_t,

```
typedef struct
{
    v4v_addr_t src;
    v4v_addr_t dst;
} v4v_send_addr_t;
```

που όπως φαίνεται περιέχει δύο δομές τύπου v4v_addr_t, μία για την διεύθυνση του αποστολέα και μία για την διεύθυνση του παραλήπτη. Τέλος, το Hypercall χρειάζεται και μία ακόμα παράμετρο την protocol. Αυτή περιέχει την πληροφορία σχετικά με τον τύπο της επικοινωνίας, δηλαδή stream ή datagram. Αν και η πληροφορία αυτή είναι αδιάφορη για τον Hypervisor, δίδεται, όπως θα δούμε παρακάτω, έτσι ώστε να είναι διαθέσιμη στην εικονική μηχανή που θα λάβει τα δεδομένα.

DATAGRAM

Βασική προϋπόθεση για να ξεκινήσει η διαδικασία αποστολής δεδομένων για datagram επικοινωνία είναι η κατάσταση του ring να είναι V4V_STATE_BOUND ή V4V_STATE_CONNECTED. Επίσης επειδή στα datagram sockets εμπεριέχεται η έννοια του μηνύματος συγκεκριμένου μεγέθους γίνεται έλεγχος αν το μήνυμα που επιθυμούμε

να στείλουμε είναι μικρότερο ή ίσο του μεγέθους του v4v_ring που έχουμε δηλώσει. Σε περίπτωση που το μήνυμα είναι μεγαλύτερο από το ring η αποστολή δεν μπορεί να εκτελεστεί και επιστρέφεται στον χρήστη μήνυμα λάθους.

Στη συνέχεια ο driver επιχειρεί να στείλει το μήνυμα. Πρώτα φτιάχνει τις δομές που περιγράφονται παραπάνω. Για τα datagram πακέτα χρησιμοποιείται μόνο μία δομή τύπου v4v_iov_t. Αξίζει αναφέρουμε ότι αυτό που αντιγράφεται είναι μόνο ο δείκτης προς τον buffer όπου υπάρχουν τα δεδομένα, δηλαδή τα δεδομένα δεν αντιγράφονται στον χώρο μνήμης του πυρήνα προκειμένου να αποσταλούν. Αυτό σημαίνει ότι δεν υπάρχει επιβάρυνση από άποψη επίδοσης. Στη συνέχεια κάνει κλήση προς τον Hypervisor, ενώ ταυτόχρονα μπλοκάρει σε μία ουρά έως ότου ολοκληρωθεί η μεταφορά, δηλαδή περιμένει μέχρι ο Hypervisor να αντιγράψει τα δεδομένα στον παραλήπτη και να τον ενημερώσει για την έκβαση της μεταφοράς. Σε περίπτωση που η μεταφορά των δεδομένων ολοκληρώθηκε επιτυχώς, η δουλειά του driver τελειώνει και επιστρέφει στον χρήστη τον αριθμό των bytes που στάλθηκαν. Στην περίπτωση όμως, που για τον οποιοδήποτε λόγο ο Hypervisor δεν κατάφερε να κάνει την μεταφορά των δεδομένων επιστρέφεται μήνυμα λάθους στον driver. Σε αυτό το σημείο ο driver βάζει το μήνυμα σε μία λίστα όπου υπάρχουν όλα τα μηνύματα για τα οποία εκκρεμεί η αποστολή και μπλοκάρει ξανά στην ουρά, δηλαδή η αντίστοιχη διεργασία κοιμάται. Ο χειρισμός των εκκρεμών αποστολών περιγράφεται σε επόμενη ενότητα.

STREAM

Στην stream επικοινωνία απαραίτητη προϋπόθεση είναι η κατάσταση του ring να είναι V4V_STATE_CONNECTED (αν στέλνει ο client) ή V4V_STATE_ACCEPTED (αν στέλνει ο server), οπότε γίνονται και οι ανάλογοι έλεγχοι. Ένα σημείο που έχει ενδιαφέρον εδώ είναι το εξής: από τη μία πλευρά η stream επικοινωνία έχει την έννοια ότι μπορεί να στείλει μια ροή δεδομένων χωρίς να περιορίζεται από συγκεκριμένο μέγεθος μηνύματος. Ωστόσο, το V4V είναι σχεδιασμένο έτσι ώστε να μπορεί να στείλει το πολύ τόσα δεδομένα όσο είναι το μέγεθος του ring. Προκειμένου να ξεπεράσει αυτήν την αντίφαση, ο driver φροντίζει να κατακερματίζει το μήνυμα σε μικρότερα μηνύματα και να τα στέλνει ένα ένα με τη σειρά. Αυτό γίνεται μέσω της μεταβλητής write_lump που προσδιορίζει το μέγεθος του μηνύματος που θα σταλεί και που όπως θα φανεί στο επόμενο κεφάλαιο, το μέγεθος στο οποίο θα γίνει ο κατακερματισμός επηρεάζει την επίδοση.

Στην περίπτωση της stream επικοινωνίας στέλνεται μία λίστα από δύο δομές v4v_ionv_t. Η πρώτη δομή χρησιμοποιείται για τον stream_header του μηνύματος ο οποίος κατασκευάζεται εκείνη τη στιγμή, ενώ η δεύτερη δομή χρησιμοποιείται για το ίδιο το μήνυμα. Στη συνέχεια η διαδικασία αποστολής των δεδομένων είναι παρό-

μοια με αυτή που ακολουθείται για τα datagram πακέτα. Ο driver επιχειρεί να στείλει το μήνυμα κάνοντας Hypercall και περιμένει σε μία ουρά για την διεκπεραίωση της αποστολής. Σε περίπτωση επιτυχίας επιστρέφεται στον χώρο χρήστη ο αριθμός των bytes που στάλθηκαν ενώ σε περίπτωση αποτυχίας το μήνυμα προστίθεται στην λίστα με τα εκκρεμή προς αποστολή μηνύματα και παράλληλα η αντίστοιχη διεργασία μπλοκάρει στην ουρά αναμονής.

Επίπεδο hypervisor: Σε αυτό το σημείο ο hypervisor καλείται να μεταφέρει δεδομένα από μία εικονική μηχανή σε μία άλλη. Όπως έχουμε αναφέρει ξανά, ο hypervisor δεν κάνει διάκριση μεταξύ πρωτοκόλλων. Το μόνο που χρειάζεται να ξέρει είναι ποια δεδομένα πρέπει να στείλει, το μέγεθος αυτών των δεδομένων καθώς και τις διευθύνσεις παραλήπτη και αποστολέα. Στο πλαίσιο του V4V διεύθυνση σημαίνει το id της εικονικής μηχανής καθώς και το port number με το οποίο σχετίζεται το εκάστοτε v4v_ring.

Όταν η ροή εκτέλεσης φτάσει στον Hypervisor, το πρώτο πράγμα που γίνεται είναι ο έλεγχος της ορθότητας των παραμέτρων που δόθηκαν από τον driver. Στη συνέχεια, ο Hypervisor με βάση τις διευθύνσεις που του δόθηκαν βρίσκει τον παραλήπτη του μηνύματος μέσα από τη λίστα με τις δομές v4v_ring_info. Αφού βρεθεί ο παραλήπτης του μηνύματος, κατασκευάζεται και προτίθεται ο message_header του μηνύματος. Ο message_header είναι μία δομή τύπου v4v_ring_-message_header

```
struct v4v_ring_message_header
{
    uint32_t len;
    uint32_t pad0;
    v4v_addr_t source;
    uint32_t message_type;
    uint8_t data[0];
};
```

Αυτή η δομή παρέχει πληροφορίες στον παραλήπτη σχετικά με τον αποστολέα (source), το μέγεθος του μηνύματος (len) και τον τύπο του μηνύματος (message_type). Ο Hypervisor γράφει αυτές τις πληροφορίες στην δομή την οποία τελικά την αντιγράφει στο ring του παραλήπτη. Συγκεκριμένα στο πεδίο message_type αντιγράφεται η τιμή του protocol που περάστηκε από τον driver. Ο hypervisor δεν αλλάζει την συμπεριφορά του ανάλογα με την τιμή αυτής της μεταβλητής, που είναι stream ή datagram, απλώς την αντιγράφει γιατί είναι απαραίτητη για τον παραλήπτη όταν επιχειρήσει να διαβάσει τα δεδομένα.

Στη συνέχεια αρχίζει η αντιγραφεί του κυρίως μηνύματος. Ο hypervisor εξετάζει ένα ένα τα v4v_iov_t και τα προωθεί στην συνάρτηση που εκτελεί την αντιγραφή. Η αντιγραφή γίνεται σε επίπεδο σελίδας. Αφού πρώτα ο Hypervisor συμβουλευτεί τον πίνακα που κρα-

τάει στην δομή v4v_ring_info σχετικά με την αντιστοίχηση ψευδοφυσικών και φυσικών σελίδων (pfns και mfns) παίρνει τα δεδομένα από το ring του αποστολέα και τα αντιγράφει στο ring του παραλήπτη. Αυτό γίνεται διαδοχικά για κάθε μία σελίδα που καταλαμβάνουν τα δεδομένα. Προφανώς αν τα δεδομένα καταλαμβάνουν λιγότερο από μία σελίδα η διαδικασία αντιστοίχησης - αντιγραφής γίνεται μόνο μία φορά. Κατά τη διάρκεια όλης της διαδικασίας ο Hypervisor μεταβάλει τον tx_pointer, που όπως έχουμε αναφέρει δείχνει μέχρι ποιο σημείο έχουν γραφτεί δεδομένα στο ring, και ταυτόχρονα κάνει ελέγχους σχετικά με τον εναπομένοντα χώρο ώστε να μην υπάρξει αλλοίωση των δεδομένων (data corruption).

Αφού ολοκληρωθεί επιτυχώς η μεταφορά των δεδομένων, αποστέλλεται στον παραλήπτη μια εικονική διακοπή μέσω του μηχανισμού των event channels . Ουσιαστικά, ο hypervisor ενημερώνει τον παραλήπτη ότι υπάρχουν διαθέσιμα νέα δεδομένα και κατ' αυτόν τον τρόπο ξεκινάει η διαδικασία ανάγνωσης των δεδομένων από το άλλο άκρο επικοινωνίας.

Το παραπάνω σενάριο αναφέρεται στην επιτυχή μεταφορά δεδομένων. Οι περιπτώσεις που αποτυγχάνει η μεταφορά οφείλονται είτε σε λάθη που προέρχονται από τον χώρο χρήστη και κυρίως λόγω λανθασμένων ορισμάτων, είτε στην περίπτωση όπου δεν υπάρχει αρκετός διαθέσιμος χώρος στο ring για την αντιγραφή των δεδομένων. Η πρώτη περίπτωση αντιμετωπίζεται είτε σε επίπεδο driver είτε σε επίπεδο Hypervisor, ανάλογα με την περίπτωση, και επιστρέφεται σχετικό μήνυμα λάθους. Η δεύτερη περίπτωση εξετάζεται αναλυτικά σε ξεχωριστή ενότητα παρακάτω.

Ανάγνωση Δεδομένων

Βιβλιοθήκη χώρου χρήστη: Αντίστοιχα με την αποστολή δεδομένων, έτσι και η λήψη μπορεί να γίνει με διάφορες συναρτήσεις recv(), recvfrom(), recvmsg() και read(). Για την read() ισχύει ότι ισχύει για την κλήση write() παραπάνω, δηλαδή δεν χρειάζεται να παρέχουμε κάποιου είδους υποστήριξη στο επίπεδο της βιβλιοθήκης. Παρέχουμε υποστήριξη για την κλήση recvfrom

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);
```

Όπως και στην περίπτωση της sendto() το όρισμα buf πρόκειται για τον χώρο όπου θα αντιγραφεί το μήνυμα, το όρισμα len αναφέρεται στο μέγεθος του μηνύματος, το flag προσδιορίζει διάφορες παραμέτρους τις επικοινωνίας που στην περίπτωση του v4v μας είναι αδιάφορες και τα ορίσματα src_addr και addlen αναφέρονται στην διεύθυνση του αποστολέα. Όπως και πριν, η βιβλιοθήκη παίρνει τις παραμέτρους sockfd, buf, και len και εκτελεί την read().

```
ssize_t read(int sockfd, const void *buf, size_t len);
```

Η κλήση προωθείται προς τον v4v driver ο οποίος είναι υπεύθυνος για την ανάγνωση των δεδομένων καθώς και για την παράδοσή τους στον χώρο χρήστη.

V4V Driver - Kernel Space

DATAGRAM

Στην περίπτωση των datagram sockets η διαδικασία της ανάγνωσης είναι αρκετά απλή. Καθώς ξεκινάει η εκτέλεση στο επίπεδο του driver δεν υπάρχει καμία εγγύηση ότι υπάρχουν δεδομένα προς ανάγνωση, δηλαδή μπορεί για παράδειγμα ο server να έχει καλέσει την read() αλλά δεν ξέρουμε αν στην πλευρά του client έχει ολοκληρωθεί η διαδικασία της εγγραφής. Οπότε αυτό που κάνει ο πυρήνας είναι να μπλοκάρει την τρέχουσα διεργασία σε μία ουρά αναμονής. Για να συνεχίσει η εκτέλεση πρέπει κάποιος να ξυπνήσει την μπλοκαρισμένη διεργασία. Όπως αναφέραμε παραπάνω, όταν η διαδικασία μεταφοράς των δεδομένων ολοκληρωθεί, ο Hypervisor στέλνει μια εικονική διακοπή στον παραλήπτη. Όταν φτάσει αυτή η διακοπή, η εκτέλεση περνάει στον διαχειριστή των διακοπών του driver. Ο διαχειριστής των διακοπών εξετάζει τον τύπο του ring και στην περίπτωση που είναι V4V_RTYPE_DGRAM, δηλαδή πρόκειται για datagram επικοινωνία ξυπνάει όλε τις διεργασίες που περιμένουν για αυτό το ring.

Αφού ξυπνήσει η διεργασία, η εκτέλεση συνεχίζει από εκεί που σταμάτησε. Πλέον είναι σίγουρο ότι υπάρχουν δεδομένα προς ανάγνωση μέσα στο ring του παραλήπτη και το μόνο που μένει είναι να διαβαστούν. Αρχικά ο driver διαβάζει τον message_header, από τον οποίο γίνεται γνωστός και ο αποστολέας του μηνύματος. Ωστόσο δεν γίνεται κάποιος έλεγχος σχετικά με τον αποστολέα γιατί δεν απαιτείται από το πρωτόκολλο UDP. Στην συνέχεια αντιγράφονται στην διεύθυνση μνήμης που δείχνει η παράμετρος buf, που δόθηκε από τον χρήστη, δεδομένα μεγέθους len, το οποίο επίσης δόθηκε από τον χρήστη. Τέλος, ο driver επιστρέφει τον αριθμό τον των bytes που διαβάστηκαν και ο έλεγχος περνάει στην εφαρμογή του χρήστη.

Όπως γίνεται φανερό, στην περίπτωση των datagram πακέτων τα μηνύματα αντιγράφονται από το ring που βρίσκεται στον πυρήνα της εικονικής μηχανής κατευθείαν στον χώρο μνήμης που βλέπει η εφαρμογή, χωρίς να μεσολαβεί κάποιος επιπλέον έλεγχος ή κάποια άλλη διαδικασία γιατί κάτι τέτοιο δεν απαιτείται από το πρωτόκολλο UDP. Αν η εφαρμογή απαιτεί κάποιου είδους αξιοπιστία τότε είναι ευθύνη της εφαρμογής να την εξασφαλίσει.

STREAM

Η περίπτωση της stream επικοινωνίας είναι αρκετά πιο πολύπλοκη όσο αφορά την ανάγνωση των δεδομένων. Αρχικά γίνονται κάποιοι έλεγχοι σχετικά με την κατάσταση του ring. Οι επιτρεπτές καταστάσεις για να γίνει η ανάγνωση είναι V4V_STATE_CONNECTED

ή V4V_STATE_ACCEPTED. Έπειτα, όπως συμβαίνει και στην περίπτωση των datagrams, η διαδικασία μπλοκάρει σε μια ουρά αναμονής έως ότου την ενεργοποιήσει πάλι κάποιος.

Όταν φτάνει η εικονική διακοπή, την εκτέλεση αναλαμβάνει ο διαχειριστής διακοπών του V4V. Σε αυτό το σημείο γίνεται διάκριση ανάλογα με το type του ring, δηλαδή αν είναι V4V_RTYPE_CONNECTOR ή V4V_RTYPE_LISTENER. Αυτός ο διαχωρισμός υπάρχει για να μπορεί ο driver να διαχείζεται τα μηνύματα που ανταλλάσσονται μεταξύ client και server κατά την διάρκεια εγκαθίδρυσης της σύνδεσης. Αν ο driver εκτελείται εκ μέρους του server, οπότε πρόκειται V4V_RTYPE_LISTENER περιμένει μήνυμα αίτησης για σύνδεση. Αν πάλι εκτελείται εκ μέρους του client, πρόκειται V4V_RTYPE_CONNECTOR, και περιμένει μήνυμα γνωστοποίησης από τον server ότι δέχτηκε την αίτηση του. Επειδή αυτές οι δύο περιπτώσεις απαιτούν διαφορετικό χειρισμό έχουν υλοποιηθεί διαφορετικές συναρτήσεις για την κάθε μία. Ωστόσο, όταν πλέον έχει γίνει η σύνδεση και κατά συνέπεια ανταλλάσσονται τα πραγματικά δεδομένα, η διαδικασία που ακολουθείται και στις δύο περιπτώσεις είναι η ίδια. Για αυτό το λόγο δεν εστιάζουμε περισσότερο στη διαφορά μεταξύ connector και listener και προχωράμε στην περιγραφή της ανάγνωσης των δεδομένων.

Αρχικά ο driver διαβάζει τον stream_header και ελέγχει αν όντως το πρωτόκολλο είναι V4V_PROTO_STREAM, δηλασή αν όντως πρόκειται για stream επικοινωνία όπως επίσης ελέγχει αν ο διακριτικός αριθμός της σύνδεσης που υπάρχει στον stream_header συμφωνεί με τον αριθμό της σύνδεσης που κρατάει ο ίδιος στις δομές του. Ακολούθως, αντί να αντιγράψει το μήνυμα στον χώρο χρήστη, το τοποθετεί σε μία λίστα από δομές τύπου pending_recv.

```
struct pending_recv {
        struct list_head node;
        v4v_addr_t from;
        size_t data_len, data_ptr;
        struct v4v_stream_header sh;
        uint8_t data[0];
}
```

Δηλαδή, κάθε μήνυμα που καταφτάνει στον παραλήπτη πρώτα αντιγράφεται στην λίστα με τα pending_recv. Αφού ολοκληρωθεί αυτή η αντιγραφή, ο driver ξυπνάει την διεργασία που ήταν μπλοκαρισμένη στην ουρά αναμονής ώστε να φτάσουν τα δεδομένα στον χώρο χρήστη.

Αφού ξυπνήσει η διεργασία, ο driver, που εκτελείται εκ μέρους της, παίρνει μία μία τις δομές pending_recv και τις αντιγράφει (copy_to_user()) στον χώρο μνήμης του χρήστη, εκεί που του υποδεικνύει η παράμετρος buf και αντιγράφει τόσα δεδομένα όσα απαιτούνται από την παράμετρο len. Επειδή, στην διαδικασία της αποστολής είναι πολύ πιθανό το μήνυμα να έχει κατακερματιστεί σε μικρότερα,

στις περισσότερες περιπτώσεις χρειάζονται να αντιγραφούν παραπάνω από μία δομές pending_recv προκειμένου να γράψει ο driver len δεδομένα. Τέλος, ο driver επιστρέφει τον συνολικό αριθμό bytes που έγγραψε και παραδίδει τον έλεγχο εκτέλεσης στον χώρο χρήστη.

Επίπεδο hypervisor: Ο hypervisor δεν λαμβάνει μέρος στην διαδικασία ανάγνωσης των δεδομένων. Αυτό γίνεται επειδή τα δεδομένα μεταφέρονται από το ένα ring στο άλλο χωρίς να αντιγραφούν μέσα στον hypervisor. Οπότε ο driver μπορεί να διαβάσει και να αντιγράψει τα δεδομένα στον χρηστή κατευθείαν, χωρίς την συμβολή του hypervisor, γιατί ούτως ή άλλως υπάρχουν στον δικό του χώρο μνήμης.

Close

Η κλήση close() δεν παρουσιάζει κάποιο ιδιαίτερο ενδιαφέρον, ωστόσο την αναφέρουμε για λόγους πληρότητας. Στο επίπεδο της βιβλιοθήκης δεν απαιτείται υποστήριξη γιατί, όπως και με τις κλήσης read() και write(), προωθείται απευθείας στον driver. Στο επίπεδο τόσο του driver όσο και του hypervisor αυτό που συμβαίνει κατά την close() είναι η αποδέσμευση όλων των πόρων, δηλαδή η ελευθέρωση του χώρου μνήμης που είχε καταληφθεί από τις διάφορες δομές.

Χειρισμός Εκκρεμών Αποστολών

Σε αυτήν την ενότητα περιγράφουμε πως το V4V διαχειρίζεται την περίπτωση όπου το ring είναι γεμάτο και δεν μπορεί να εκτελεστεί η μεταφορά των δεδομένων. Όπως είχε αναφερθεί στην υποενότητα αποστολής δεδομένων, όταν ο driver ενημερωθεί ότι η μεταφορά δεν μπορεί να γίνει, προσθέτει το μήνυμα σε μία λίστα όπου κρατούνται όλες οι εκκρεμείς αποστολές για το συγκεκριμένο εικονικό μηχάνημα. Πρόκειται για μια λίστα από δομές τύπου pending_xmit, που όπως φαίνεται παρακάτω υπάρχουν όλες οι πληροφορίες που χρειάζονται για την επαναμετάδοση.

```
struct pending_xmit {
        struct list_head node;
        enum v4v_pending_xmit_type type;
        uint32_t conid;
        struct v4v_ring_id from;
        v4v_addr_t to;
        size_t len;
        uint32_t protocol;
        uint8_t data[0];
};
```

Στο επίπεδο του hypervisor η πληροφορία που αποθηκεύεται αφορά τις εκκρεμείς λήψεις. Δηλαδή ενώ στο επίπεδο του driver κάθε εικονική μηχανή γνωρίζει ποια μηνύματα δεν εστάλησαν, στο επίπεδο

του Hypervisor κάθε ring γνωρίζει ποιες λήψεις δεν κατάφεραν να πραγματοποιηθούν. Η λίστα pending της δομής v4v_ring_info υπάρχει για να αποθηκεύονται οι αποτυχημένες λήψεις για κάθε ring. Ωστόσο, στο επίπεδο του hypervisor οι μόνες πληροφορίες που χρειάζεται να αποθηκευτούν είναι το id του αποστολέα και το μέγεθος των δεδομένων προς λήψη.

Η διαδικασία της επαναμετάδοσης ξεκινάει έπειτα από κάθε εικονική διακοπή. Κατά τη διάρκεια αυτής της διαδικασίας χρειάζονται δύο επιπλέον δομές, η v4v_ring_data_t και v4v_ring_data_ent_t.

```
typedef struct v4v_ring_data_ent
{
    v4v_addr_t ring;
    uint16_t flags;
    uint16_t pad;
    uint32_t space_required;
    uint32_t max_message_size;
} v4v_ring_data_ent_t;

typedef struct v4v_ring_data
{
    uint64_t magic;
    uint32_t nent;
    uint32_t pad;
    uint64_t reserved[4];
    v4v_ring_data_ent_t data[0];
} v4v_ring_data_t;
```

Κάθε μία δομή τύπου v4v_ring_data_ent_t αναφέρεται σε ένα μήνυμα που εκκρεμεί. Όπως φαίνεται, οι πληροφορίες που χρειάζονται είναι η διεύθυνση του αποστολέα (ring), ο χώρος που απαιτείται (space_required), το μέγιστο μέγεθος μηνύματος (max_message_size) καθώς και ένα πεδίο flags. Οι τιμές που μπορεί να πάρει αυτό το πεδίο είναι V4V_RING_DATA_F_EMPTY, που σημαίνει ότι το ring είναι άδειο, V4V_RING_DATA_F_EXISTS, που σημαίνει ότι το ring υπάρχει και V4V_RING_DATA_F_SUFFICIENT, που σημαίνει ότι υπάρχει επαρκής χώρος στο ring για να γίνει η αντιγραφή. Η δομή v4v_ring_data_t περιέχει τον αριθμό των δομών v4v_ring_data_ent_t που υπάρχουν στην εικονική μηχανή (nent) καθώς και έναν πίνακα που δείχνει στις δομές αυτές (data[0]).

Στο επίπεδο του driver, αφού φτάσει η διακοπή και αφού διαβαστούν τα δεδομένα καλείται η συνάρτηση v4v_notify(). Αυτή η συνάρτηση για κάθε pending_xmit που υπάρχει κατασκευάζει μία δομή v4v_ring_data_ent_t καθώς και την δομή v4v_ring_data_t και στην συνέχεια κάλει τον Hypervisor δίνοντας ως παράμετρο έναν δείκτη στη δομή v4v_ring_data_t.

Στο επίπεδο του Hypervisor η notify() επιτελεί δύο λειτουργίες. Πρώτον, ελέγχει αν μπορούν να γίνουν οι εκκρεμείς λήψεις που υπάρχουν για κάθε ring της εικονικής μηχανής και δεύτερον, να δώσει

στον driver πληροφορίες σχετικά με τον εναπομένοντα χώρο των rings των άλλων εικονικών μηχανών, για τα οποία υπάρχουν εκκρεμείς αποστολές. Αρχικά, ο hypervisor βρίσκει τον διαθέσιμο χώρο που υπάρχει για κάθε ring της εικονικής μηχανής που έκανε το hypercall. Στη συνέχεια για κάθε ring του εικονικού μηχανήματος, μέσω της λίστας pending της δομής v4v_ring_info, βρίσκει ποιες εκκρεμείς λήψεις υπάρχουν και αν υπάρχει ο απαιτούμενος χώρος ειδοποιεί τις αντίστοιχες εικονικές μηχανές μέσω εικονικών διακοπών, ώστε να γίνει η αποστολή. Όσο αφορά την δεύτερη λειτουργία, για κάθε μία δομή v4v_ring_data_ent_t, δηλαδή για κάθε εκκρεμή αποστολή, βρίσκει τον διαθέσιμο χώρο που υπάρχει στο ring του partner και θέτει ανάλογα το πεδίο flags της δομής, το οποίο αντιγράφεται στον driver. Σε αυτό το σημείο το Hypercall τελειώνει και η ροή εκτέλεσης επιστρέφει στον driver.

Ο driver πλέον γνωρίζει για ποια pending_xmits υπάρχει διαθέσιμος χώρος ώστε να γίνει επιτυχώς η αποστολή. Υπενθυμίζουμε ότι κατά την διαδικασία αποστολής η διεργασία που επιθυμούσε να στείλει δεδομένα αλλά δεν μπορούσε λόγω μη επαρκούς χώρου στο ring, μπλόκαρε σε μία ουρά αναμονής και κοιμόταν. Τώρα ο driver ελέγχοντας το πεδίο flags, για κάθε pending_xmit για το οποίο το flags έχει τιμή V4V_RING_DATA_F_SUFFICIENT, ξυπνάει την αντίστοιχη διεργασία που ήταν κολλημένη πάνω στην ουρά αναμονής ώστε να γίνει ξανά προσπάθεια μετάδοσης των δεδομένων.

Η κλήση v4v_notify() γίνεται μετά από κάθε διακοπή για τον παρακάτω λόγο. Το γεγονός ότι έχει φτάσει κάποια διακοπή σημαίνει ότι υπάρχουν δεδομένα προς ανάγνωση τα οποία τελικά θα διαβαστούν με αποτέλεσμα να αδειάσει το ring. Οπότε υπάρχει σίγουρα διαθέσιμος χώρος και συνεπώς μπορεί να γίνει λήψη νέων δεδομένων που ενδεχομένως είχε μπλοκαριστεί νωρίτερα (πρώτη λειτουργία της notify()). Επιπλέον, είναι αρκετά βολικό τόσο από άποψη υλοποίησης όσο και επίδοσης στο ίδιο Hypercall να γίνεται έλεγχος αν μπορούν να σταλούν εκκρεμείς αποστολές (δεύτερη λειτουργία της notify()). Συμφέρει από άποψη υλοποίησης γιατί δεν χρειάζεται να υλοποιηθεί ένα επιπλέον Hypercall που θα αφορά τις εκκρεμείς αποστολές αποκλειστικά και από άποψη επίδοσης συμφέρει γιατί μειώνονται οι κλήσεις προς τον hypervisor.

Κεφάλαιο 5

Evaluation

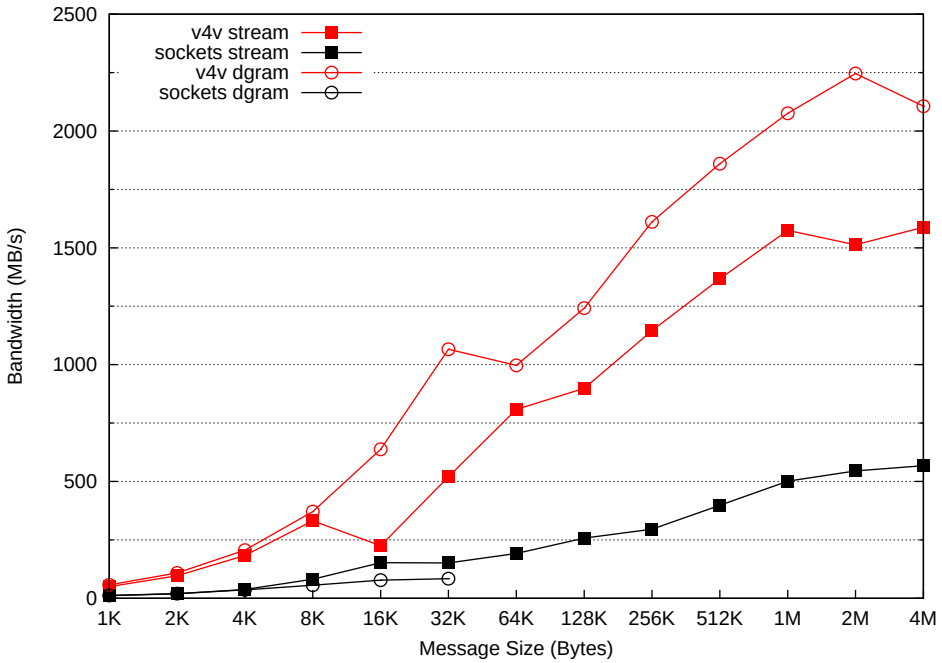
Σε αυτό το κεφάλαιο παρουσιάζουμε την πειραματική αποτίμηση του V4Vsockets. Για την αξιολόγηση της απόδοσης φτιάξαμε ένα micro-benchmark, το οποίο εκτελεί την μεταφορά των δεδομένων από τον χώρο χρήστη μιας εικονικής μηχανής στον χώρο χρήστη μιας άλλης. Για να ελέγξουμε την ακεραιότητα των δεδομένων χρησιμοποιούμε την μέθοδο του ping-pong. Δηλαδή, ο client στέλνει τα δεδομένα στον server, ο server τα διαβάζει και έπειτα τα στέλνει πίσω στον client όπου γίνεται ο έλεγχος για την ακεραιότητά τους.

Τα πειράματα εκτελέστηκαν σε μηχάνημα με τις εξής προδιαγραφές: 2 επεξεργαστές Intel Xeon X5650@2.677GHz, chipset Intel 5520 και κεντρική μνήμη 12x4GB ECC@1333MHz. Επίσης χρησιμοποιήθηκε η έκδοση του Xen 4.5-unstable και πυρήνα Linux έκδοσης 3.14.2.

5.1 V4Vsockets vs Socket

Σε αυτή την ενότητα παρουσιάζουμε την επίδοση του V4Vsockets σε σύγκριση με τον κλασσικό τρόπο επικοινωνίας που προσφέρεται από το Xen, χρησιμοποιώντας την προγραμματιστική διεπαφή Socket. Στα διαγράμματα 1 και 2 παρουσιάζεται το bandwidth και το latency αντίστοιχα, για dgram και stream socket, χρησιμοποιώντας το V4Vsockets για την επικοινωνία αλλά και τον συμβατικό τρόπο που προσφέρει το Xen. Οι παράμετροι της εκτέλεσης είναι 512KB ring size, για δεδομένα από 1 byte μέχρι 32KB και write_lump 32KB (έχει νόημα μόνο για stream).

Όπως φαίνεται στο Σχήμα 5.1, με το V4Vsockets για dgram socket πετυχαίνουμε στην καλυτερη περίπτωση περίπου 15 φορές καλύτερο bandwidth (για μέγεθος δεδομένων 32KB) σε σχέση με τα κλσσικά socket, ενώ στην χειρότερη 3 φορές καλύτερο bandwidth. Αντίστοιχα, για stream socket παρατηρούμε ότι πετυχαίνουμε στην χειρότερη περίπτωση 3 φορές καλύτερη επίδοση ενώ στην καλύτερη 5 με 6 φορές ταχύτερη επικοινωνία (για μέγεθος δεδομένων 2KB και 4KB).



Σχήμα 5.1: Bandwidth

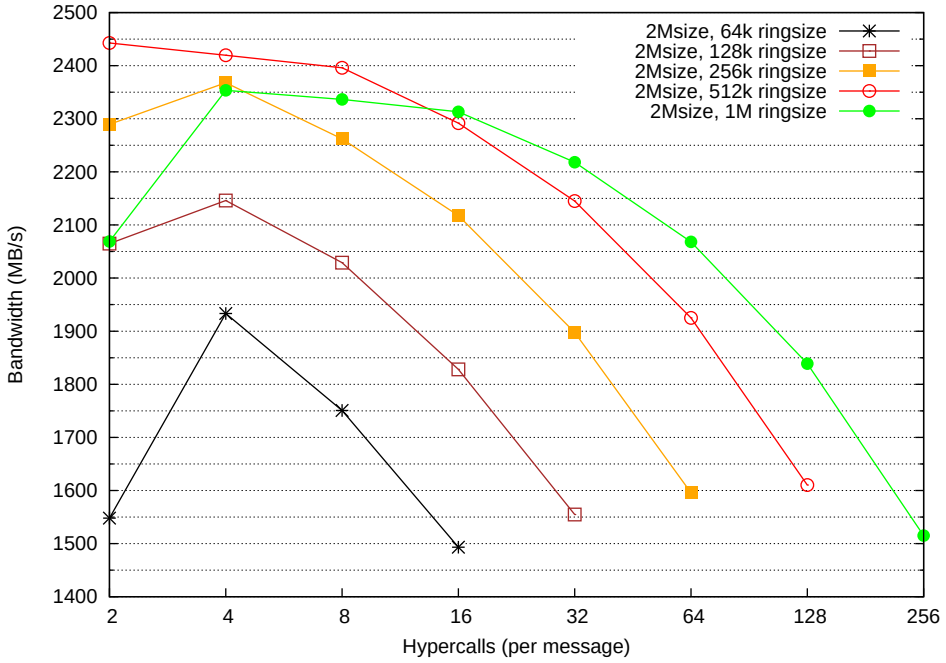
Επιπλέον παρατηρούμε ότι στο V4Vsockets, το dgram είναι ταχύτερο από το stream. Αυτό οφείλεται στο παρακάτω γεγονός: όπως περιγράψαμε στο προηγούμενο κεφάλαιο για την διεξαγωγή της datagram επικοινωνίας απαιτούνται 2 αντιγραφές ενώ στην περίπτωση της stream επικοινωνίας 3 αντιγραφές. Η μία λιγότερη αντιγραφή είναι που προσδίδει στη datagram επικοινωνία καλύτερη επίδοση.

5.2 Παράγοντες που επηρεάζουν την επίδοση

Ένας από τους σημαντικότερους παράγοντες που επηρεάζουν την επίδοση του V4Vsockets είναι η επιβάρυνση που προσδίδει το context/mode switch που γίνεται στο λειτουργικό σύστημα της εικονικής μηχανής (mode switch) καθώς και μεταξύ του λειτουργικού συστήματος και του hypervisor. Σε μια λειτουργία αποστολής/λήψης ενός stream μηνύματος, η εφαρμογή χώρου χρήστη εκτελεί ένα system-call για την αποστολή – παρόλα αυτά, ο V4V driver κατακερματίζει το μήνυμα σε `write_lump` υπο-μηνύματα, για καθένα από τα οποία εκτελεί ένα hypercall στο Xen για να ξεκινήσει η διαδικασία αντιγραφής στο ring του παραλήπτη.

Μελετάμε αυτή τη διαδικασία, εκτελώντας πειράματα για διαφορετικά μεγέθη `ring_size` και `write_lump`. Στο σχήμα 5.2 φαίνεται η

ρυθμαπόδοση του συστήματος για μηνύματα μεγέθους 2MB σε σχέση με τον αριθμό των hypercalls που γίνονται στο Xen. Οι καμπύλες δείχνουν τα διαφορετικά μεγέθη του `ring_size` ενώ στον άξονα των Χ φαίνεται ο λόγος $\frac{ring\_size}{write\_lump}$ που ουσιαστικά δηλώνει τον αριθμό των hypercalls ανά μήνυμα.



Σχήμα 5.2: Ρυθμαπόδοση σε σχέση με τον αριθμό των hypercalls ανά μήνυμα

5.3 Κλιμακωσιμότητα του V4Vsockets

Όπως αναφέρουμε στη σχεδίαση του V4Vsockets, η μεταφορά των δεδομένων ουσιαστικά ανάγεται σε μια λειτουργία αντιγραφής στην κεντρική μνήμη του συστήματος. Σύμφωνα με τα παραπάνω αποτελέσματα (Ενότητα 5.1 και 5.2), θα έπρεπε η επίδοση του συστήματος να είναι συγκρίσιμη με αυτή μιας αντιγραφής μνήμης. Για το λόγο αυτό εκτελούμε το μετροπρόγραμμα stream για την αποτίμηση του συστήματος μνήμης.

Στον παρακάτω πίνακα φαίνεται η επίδοση του συστήματος μνήμης της πειραματικής πλατφόρμας που χρησιμοποιούμε.

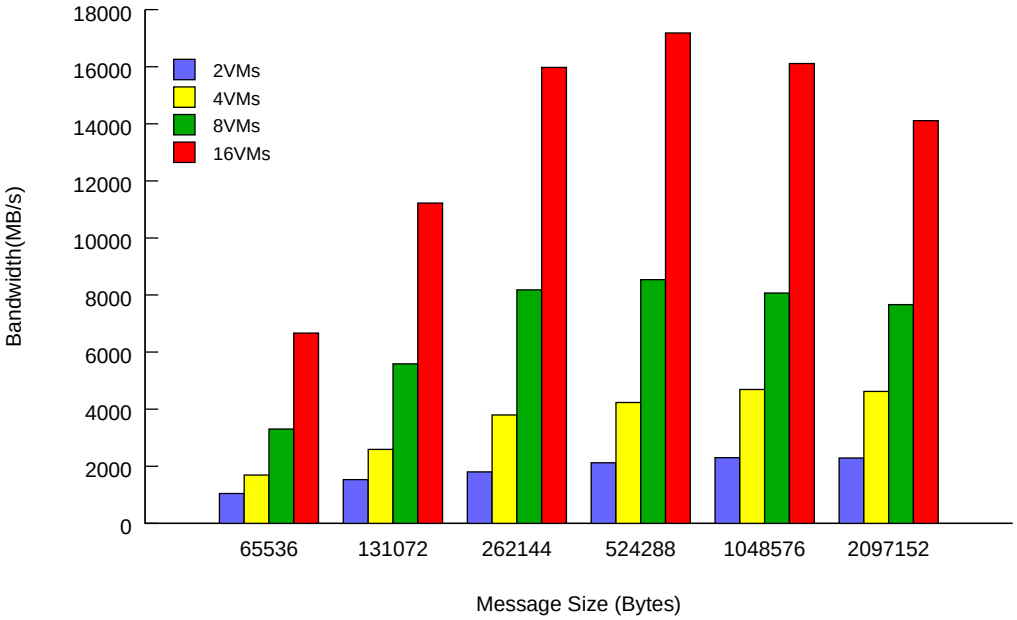
Με δεδομένο ότι το V4Vsockets κάνει 3 αντιγραφές στο κρίσιμο μονοπάτι (στην περίπτωση του Stream) θα έπρεπε η επίδοση του συστήματός μας να είναι 3 φορές πιο αργή. Παρόλα αυτά παρατηρούμε

Threads	1	2	4	8	16
Mem BW (MB/s)	4803	9143	13474	19542	27332

Πίνακας 5.1: Memory Bandwidth (MB/sec)

ότι στην καλύτερη περίπτωση του V4Vsockets η επίδοση είναι 15 φορές μικρότερη (περίπου 2GB/s σε αντίθεση με 27GB/s). Για το λόγο αυτό, εκτελούμε πειράματα σε παραπάνω από 1 ζεύγος εικονικών μηχανών για να μελετήσουμε τόσο το scalability της μεθόδου μας, όσο και την εγκυρότητα του παραπάνω επιχειρήματος.

Για την εκτέλεση του πειράματος χρησιμοποιούμε μέχρι 16 εικονικά μηχανήματα. Στο κάθε εικονικό μηχάνημα δίνουμε 2 cpus καθώς και 2GB μνήμη. Εκτελούμε το micro-benchmark ταυτόχρονα μεταξύ ζευγαριών. Το συνολικό bandwidth (aggregate bandwidth) καθώς και το latency για 2, 4, 8 και 16 εικονικές μηχανές φαίνονται στα Σχήματα 5.3 και 5.4.

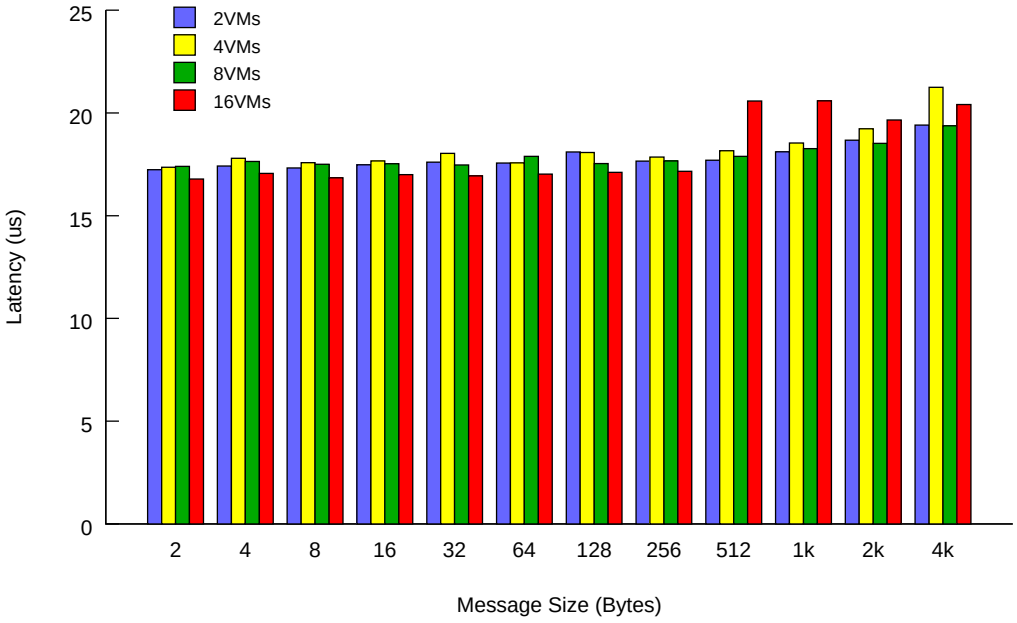


Σχήμα 5.3: V4Vsockets: bandwidth scalability για 2, 4, 8 και 16 VMs

Παρατηρούμε ότι όσο ο αριθμός των εικονικών μηχανών που συμμετέχουν στο πείραμα αυξάνεται, τόσο περισσότερο αυξάνεται και η συνολική ρυθμαπόδοση του συστήματος. Για παράδειγμα για 1 ζεύγος εικονικών μηχανών, για μηνύματα μεγέθους 512KB η ρυθμαπόδοση είναι $\approx$ 2GB/s ενώ για 8 ζεύγη, εκτοξεύεται στο οκταπλάσιο ($\approx$16GB/s).

Θα πρέπει να σημειωθεί ότι σύμφωνα με την παραπάνω υπόθεση για τη μνήμη του μηχανήματος περιμένουμε η συνολική ρυθμαπόδοση να καταλήγει να είναι περίπου το μισό του μετροπρογράμματος stream. Αυτό οφείλεται στο ότι κατά τη διάρκεια της 3ης αντιγραφής στο V4Vsockets, το σύστημα μπορεί να εκμεταλλευτεί την cache των επεξεργαστών και να μην καταλήξει σε χρονοβόρα πρόσβαση στην κεντρική μνήμη.

Συνολικά λοιπόν, η ρυθμαπόδοση του συστήματος για 16 εικονικές μηχανές (8 ζεύγη που ανταλλάσσουν μηνύματα μεταξύ τους και πιέζουν τη μνήμη) αγγίζει το μισό περίπου του μετροπρογράμματος stream (16.5GB/s vs. 27GB/s).



Σχήμα 5.4: V4Vsockets: latency scalability για 2, 4, 8 και 16 VMs

Όσον αφορά στο latency (Σχήμα 5.4) παρατηρούμε ότι ο χρόνος απόκρισης του συστήματος παραμένει σταθερός παρόλο που αυξάνουμε τον αριθμό των εικονικών μηχανών – για 1 ζεύγος εικονικών μηχανών, ο χρόνος που χρειάζεται για να φτάσει 1 byte από τη μία εικονική μηχανή στην άλλη είναι περίπου 17us. Για 2 ζεύγη ο χρόνος αυτός είναι ο ίδιος και για τα δύο ζεύγη ($\approx$ 17us για κάθε ζεύγος) ενώ για 8 ζεύγη ο χρόνος παραμένει $\approx$ 17us. Το γεγονός αυτό αποδεικνύει πως δεν υπάρχουν επιπλέον επιβαρύνσεις που οφείλονται στο hypervisor και αναδυκνύει το σωστό σχεδιασμό του συστήματος.

Κεφάλαιο 6

Επίλογος

6.1 Σύνοψη

Σκοπός της παρούσας εργασίας ήταν μελέτη και σύγκριση διαφορετικών τρόπων ενδοεπικοινωνίας εικονικών μηχανών χωρίς να αλλάζει η προγραμματιστική δοεπαφή. Για τον σκοπό αυτό χρησιμοποιήθηκε ένας μηχανισμός ενδοεπικοινωνίας, ο V4V, ο οποίος επιτρέπει την επικοινωνία μεταξύ δύο εικονικών μηχανών μέσω ανταλλαγής μηνυμάτων και επιπλέον χρησιμοποιεί την σημασιολογία του πρωτοκόλλου TCP/IP. Στηριζόμαστε σε αυτόν τον μηχανισμό και δημιουργούμε το V4Vsockets, ένα πλαίσιο το οποίο είναι συμβατή με την προγραμματιστική διεπαφή Socket, αλλά παρακάμπτει την στοίβα πρωτοκόλλου TCP/IP.

Στη συνέχεια κάνουμε κάποιες σύντομες παρατηρήσεις που αφορούν τον σχεδιασμό και την υλοποίηση του V4Vsockets. Το πρώτο σημείο που αξίζει να αναφέρουμε είναι ότι η επικοινωνία διεξάγεται μέσω του hypervisor. Αυτός είναι υπεύθυνος για την δημιουργία των message_header και για την αντιγραφή των δεδομένων από το ένα άκρο επικοινωνίας στο άλλο. Το γεγονός αυτό, προσδίδει στο V4Vsockets υψηλή αξιοπιστία, αφού ο Hypervisor είναι το πιο “έμπιστο” στοιχείο της πλατφόρμας. Από τη στιγμή που η διαδικασία μεταφοράς δεδομένων ελέγχεται πλήρως από τον hypervisor ο παραλήπτης μπορεί να εμπιστευτεί τόσο τη διαμόρφωση του ring όσο και το περιεχόμενό του. Επιπλέον, η εμπλοκή του Hypervisor στην επικοινωνία μας δίνει την καλύτερη δυνατή απομόνωση μεταξύ των εικονικών μηχανών, αφού καμιά εικονική μηχανή δεν έχει πρόσβαση στο χώρο μνήμης κάποιας άλλης.

Μία δεύτερη παρατήρηση είναι ότι το κομμάτι του V4V που είναι υλοποιημένο μέσα στον Hypervisor παρέχει έναν γενικό μηχανισμό μεταφοράς δεδομένων, ενώ ο v4v driver αναλαμβάνει την υλοποίηση δύο πρωτοκόλλων επικοινωνίας. Το γεγονός ότι σε επίπεδο hypervisor το V4V παρέχει μόνο ένα μηχανισμό ενδοεπικοινωνίας

προσφέρει μεγάλη ευελιξία καθώς μας δίνει την δυνατότητα να υλοποιήσουμε οποιοδήποτε πρωτόκολλο πάνω από το μηχανισμό.

Επίσης, όπως έχει αναφερθεί και προηγούμενα κεφάλαια, το V4V sockets παρέχει συμβατότητα με την κλασσική προγραμματιστική διεπαφή Socket, που σημαίνει ότι οποιαδήποτε εφαρμογή μπορεί να επωφεληθεί από το V4V χωρίς να απαιτούνται αλλαγές στον πηγαίο κώδικα ή επαναμεταγλώττισή της. Παράλληλα, το V4Vsockets πετυχαίνει πολύ καλύτερη επίδοση και κλιμακοστιμότητα σε σχέση με τον κλασσικό μηχανισμό που παρέχεται από το Xen (netfront - netback). Το V4Vsockets παρακάμπτει αποδοτικά το split driver model, με αποτέλεσμα να μην εμπλέκεται καθόλου ο domain O στην διαδικασία της επικοινωνίας.

Τέλος, πιθανές μελλοντικές επεκτάσεις της παρούσας διπλωματικής εργασίας είναι οι ακόλουθες: Από άποψη βελτιστοποιήσεων μπορεί να γίνει α)βελτιστοποίηση της stream επικοινωνίας μειώνοντας τις αντιγραφές από δύο σε στην διαδικασία λήψης δεδομένων και β) ενδελεχής μελέτη της χρησιμοποίησης CPU, έτσι ώστε να βρούμε ενδεχόμενες επιβαρύνσεις που δεν είναι γνωστές μέχρι στιγμής και προσπάθεια εξάλειψης τους. Μία άλλη μελλοντική επέκταση είναι η μελέτη και ο προσδιορισμός των παραμέτρων της αρχιτεκτονικής της πλατφόρμας (NUMA architectures) που επηρεάζουν την επίδοση του V4Vsockets.

Βιβλιογραφία

[1] Dimitris Aragiorgis, Anastassios Nanos, and Nectarios Koziris. Coexisting scheduling policies boosting i/o virtual machines. In *Proceedings of the 2011 International Conference on Parallel Processing - Volume 2*, Euro-Par'11, pages 407–415, Berlin, Heidelberg, 2012. Springer-Verlag.

[2] François Diakhaté, Marc Pérache, Raymond Namyst, and Hervé Jourdren. Efficient shared memory message passing for inter-vm communications. In *4th Workshop on Virtualization in High-Performance Cloud Computing (VHPC '08), Euro-par 2008*, 2008.

[3] Sriram Govindan, Arjun R. Nath, Amitayu Das, Bhuvan Urgaonkar, and Anand Sivasubramaniam. Xen and co.: Communication-aware cpu scheduling for consolidated xen-based hosting platforms, 2007.

[4] Wei Huang, Matthew J. Koop, Qi Gao, and Dhabaleswar K. Panda. Virtual machine aware communication libraries for high performance computing. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, SC '07, pages 9:1–9:12, New York, NY, USA, 2007. ACM.

[5] Kangho Kim, Cheiyol Kim, Sung-In Jung, Hyun-Sup Shin, and Jin-Soo Kim. Inter-domain socket communications supporting high performance and full binary compatibility on xen. In *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '08, pages 11–20, New York, NY, USA, 2008. ACM.

[6] Aravind Menon, Alan L. Cox, and Willy Zwaenepoel. Optimizing network virtualization in xen. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 2–2, Berkeley, CA, USA, 2006. USENIX Association.

[7] Anastassios Nanos and Nectarios Koziris. Myrixen: Message passing in xen virtual machines over myrinet and ethernet. In *Proceedings of the 2009 International Conference on Parallel*

Processing, Euro-Par'09, pages 395–403, Berlin, Heidelberg, 2010. Springer-Verlag.

[8] Anastassios Nanos and Nectarios Koziris. Xen2mx: Towards high-performance communication in the cloud. In *Proceedings of the 18th International Conference on Parallel Processing Workshops*, Euro-Par'12, pages 548–556, Berlin, Heidelberg, 2013. Springer-Verlag.

[9] Anastassios Nanos, Nikos Nikoleris, Stratos Psomadakis, Elisavet Kozyri, and Nectarios Koziris. A smart hpc interconnect for clusters of virtual machines. In *Proceedings of the 2011 International Conference on Parallel Processing - Volume 2*, Euro-Par'11, pages 398–406, Berlin, Heidelberg, 2012. Springer-Verlag.

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

[11] Jian Wang, Kwame-Lante Wright, and Kartik Gopalan. Xenloop: A transparent high performance inter-vm network loopback. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, HPDC '08, pages 109–118, New York, NY, USA, 2008. ACM.

[12] Wikipedia. Osi model — Wikipedia, the free encyclopedia, 2014.

[13] Lamia Youseff, Dmitrii Zagorodnov, and Rich Wolski. Inter-os communication on highly parallel multi-core architectures.

[14] Xiaolan Zhang, Suzanne McIntosh, Pankaj Rohatgi, and John Linwood Griffin. Xensocket: A high-throughput interdomain transport for virtual machines. In *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*, Middleware '07, pages 184–203, New York, NY, USA, 2007. Springer-Verlag New York, Inc.